Desenvolvimento Rápido de Aplicações

Java Server Faces - JSF

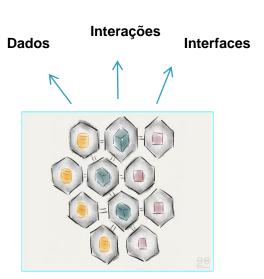
Profa. Joyce Miranda

- O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - Framework
 - □ Abstrair questões sobre arquitetura da aplicação
 - □ Manter foco na lógica de negócios

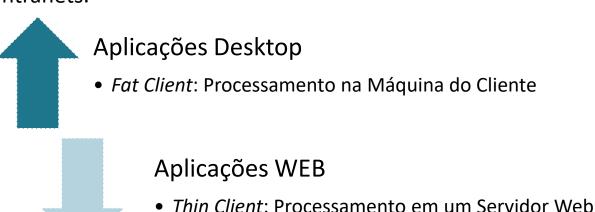


- O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - MVC
 - Padrão de arquitetura que organiza o código da aplicação em camadas com responsabilidades específicas.





- O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - Desenvolvimento web
 - Aplicação WEB
 - □ Software projetado para ser executado em navegadores usando a Internet ou Intranets.



4

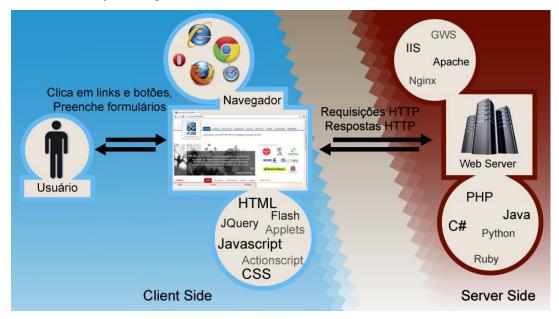
- O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - Desenvolvimento web
 - Arquitetura Aplicação WEB



- Servidor WEB
 - □ Software ou computador capaz de aceitar pedidos HTTP dos clientes e servi-los com respostas HTTP.

- O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - Desenvolvimento web
 - Tecnologias para desenvolvimento WEB
 - □ Front-End
 - ☐ Tecnologias para projetar interfaces e promover interação com o usuário do lado do cliente
 - □ Back-End
 - □ Tecnologias para dinamizar o site através de linguagens de programação que rodam do lado do servidor web.

- ▶ O que é JSF?
 - Framework MVC para desenvolvimento web em JAVA.
- Detalhando..
 - Desenvolvimento web
 - Arquitetura Aplicação WEB



Tecnologias Front-End

HTML Marcação do Conteúdo CSS Formatação e Apresentação JavaScript Comportamento e Inteligência

- Estrutura básica de um documento HTML
 - Os comandos HTML são representados por meio de tags (rótulos).

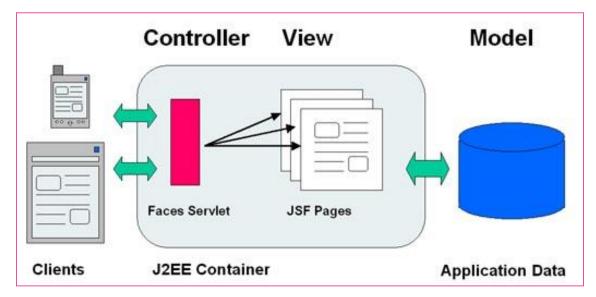
<TAG> Texto marcado pela Tag </TAG>

Características JSF

- ▶ É uma especificação JAVA para a construção de interfaces de usuário <u>baseadas em componentes</u> para aplicações web.
- Component-Based
 - Permite criar aplicações utilizando componentes visuais prédefinidos.
 - □ Capaz de criar uma interface web mais rica e menos complexa de ser implementada.
 - Adota um modelo de programação dirigido a eventos.
 - ☐ As ações dos componentes são baseadas em eventos.
 - Os componentes encapsulam detalhes do protocolo HTTP (requisiçãoxresposta).

Características JSF

- O JSF é fortemente baseado nos padrões MVC (arquitetura) e Front Controller (projeto).
- ▶ Front Controller
 - ▶ Todas as requisições são recebidas pelo mesmo componente.
 - □ Em uma aplicação JSF, toda requisição é recebida pelo Faces Servlet.



Managed Beans

- Classes da camada Model que interagem com componentes JSF.
- São responsáveis por:
 - Receber os dados enviados nas requisições.
 - Fornecer dados que serão exibidos nas telas.
 - Executar tarefas de acordo com as ações dos usuários.
- Os componentes JSF interagem com as classes Managed Beans por meio do acesso a seus métodos.
- São <u>POJOS</u> anotados com <u>@ManagedBean</u>
 - □ POJO (Plain Old Java Object)
 - ☐ Classe JAVA que expõe atributos por meio de métodos *get* e set.

Managed Beans

- Criação
 - Criar uma classe Java com a anotação @ManagedBean.

```
@ManagedBean
public class TesteBean {
   ...
}
```

- ▶ Por padrão, o JSF assume que o nome do Managed Bean é o nome da classe com a primeira letra minúscula.
- Para o exemplo acima, o nome do Managed Bean é testeBean.

- Managed Beans
 - Propriedades

```
@ManagedBean
public class TesteBean {
  private int numero;

public int setNumero(int numero) {
    this.numero = numero;
}

public int getNumero() {
    return numero;
}
```

- Recuperando e exibindo valores na página da aplicação
 - Expression Language (#{})

```
Valor: #{testeBean.numero}
```

- Estrutura Básica de uma página JSF
 - As telas são definidas em arquivos XHTML (HTML + XML)
 - Os componentes visuais que constituem as telas são adicionados por meio de tags.
 - A especificação do JSF define uma grande quantidade de *tags* e as classifica em bibliotecas.
 - HTML (http://java.sun.com/jsf/html)
 - □ Construção de telas gerando o HTML adequado
 - Core (<u>http://java.sun.com/jsf/core</u>)
 - □ Tratadores de eventos e validadores
 - Facelets (http://java.sun.com/jsf/facelets)
 - ☐ Utilização de *templates* e reutilização de código.

- Estrutura Básica de uma página JSF
 - ▶ Importando *tags* de componentes
 - As bibliotecas de tags que serão utilizadas para construir a página devem ser "importadas" através do pseudo-atributo xmlns aplicado à tag <html>.

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
```

- Componentes Visuais
 - Formulário: <h:form>
 - Comporta componentes que permitem a interação com os usuários:
 - □ caixas de texto, caixas de seleção, rótulos, botões e links.
 - Campo de Texto: <h:inputText>
 - ☐ Permite que o usuário digite uma linha de texto.
 - Rótulo: <h:outputLabel>
 - ▶ Indica o que deve ser preenchido em uma caixa de texto.
 - Botão: <h:commandButton>
 - ☐ Enviam os dados de um formulário HTML para o servidor.

Digite seu nome: Joyce Miranda Exibe Mensagem

- Componentes Visuais
 - Caixas de Texto
 - Binding
 - Associação de uma caixa de texto a uma propriedade de um *Managed* Bean através do atributo value.

```
@ManagedBean
public class TesteBean {
  private int numero;

  public int setNumero(int numero) {
    this.numero = numero;
  }

  public int getNumero() {
    return numero;
  }
}
```

<h:inputText value="#{testeBean.numero}" />

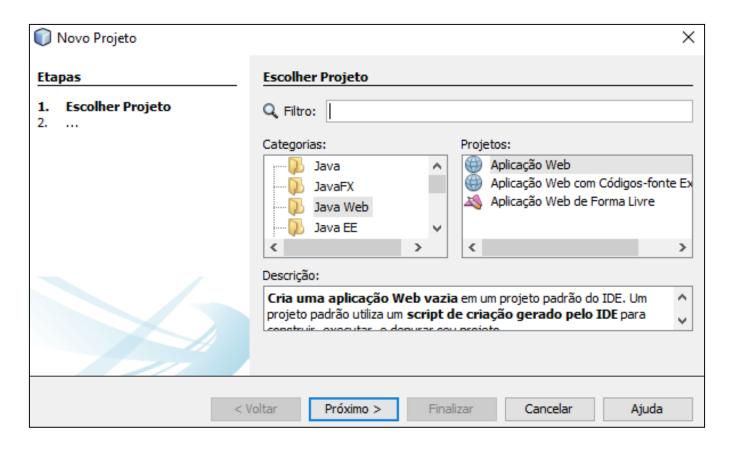
Pratique!

Crie um projeto JSF de forma a garantir que o comportamento abaixo seja devidamente interpretado em um navegador web.

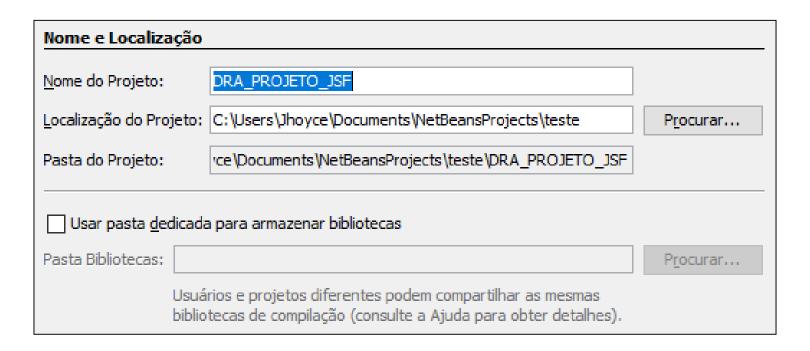
Digite seu nome: Joyce Miranda Exibe Mensagem

Olá, Joyce Miranda!

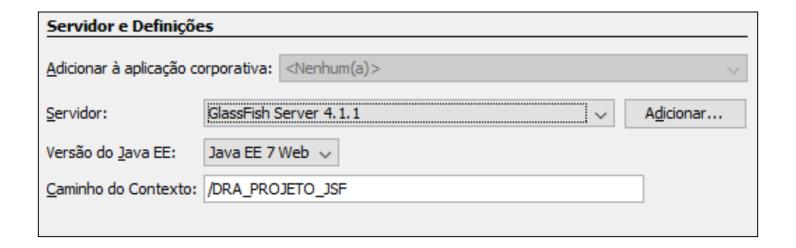
- Pratique!
 - Criando um Projeto WEB...



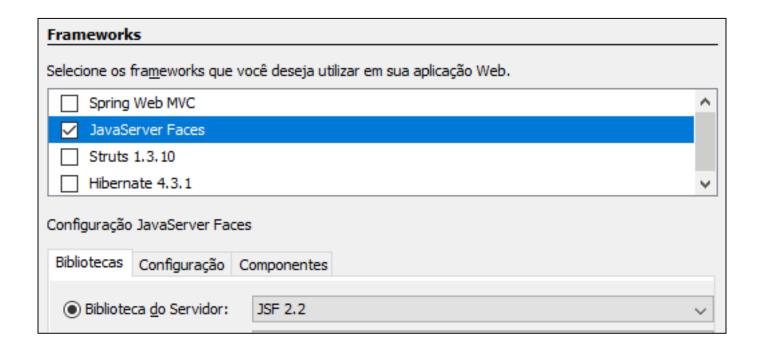
- Pratique!
 - Definindo Nome do Projeto...



- Pratique!
 - Escolhendo Servidor WEB...



- Pratique!
 - Escolhendo Framework JSF...



Pratique!

Criando Managed Bean: HelloJSF...

```
@ManagedBean
public class HelloJSF {
   private String nome;
   private String mensagem;
    public String getNome() {...3 linhas }
   public void setNome(String nome) {...3 linhas }
    public String getMensagem() {...3 linhas }
    public void setMensagem(String mensagem) {...3 linhas }
    public void exibeMensagem() {
        mensagem = "Olá" + this.nome;
```

Pratique!

Criando formulário de interação com o usuário...

```
<h:form>
     <h:outputLabel for="nome" value="Digite seu nome:"/>
     <h:inputText id="nome" value="#{helloJSF.nome}"/>
     <h:commandButton value="Exibe Mensagem" action="#{helloJSF.exibeMensagem}"/>
     <h:outputLabel for="nome" value="#{helloJSF.mensagem}"/>
     </h:form>
```

Digite seu nome: Exibe Mensagem

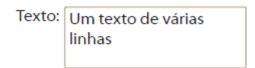


Pratique!

Crie um projeto JSF de forma a garantir que o comportamento abaixo seja devidamente interpretado em um navegador web.

Valor 1: 2.0
Valor 2:4.0
Somar
6.0

- Outros Componentes Visuais
 - https://github.com/joyceMiranda/codigosDeExemplo/blob/master/D RA PROJETO JSF/web/componentesVisuais.xhtml
 - Caixas de Texto
 - <h:inputTextarea>
 - ☐ Permite que o usuário digite várias linhas de texto.



- <h:inputSecret>
 - □ Oculta o que foi digitado.

Senha:	••••	
ocimia.		

- Outros Componentes Visuais
 - Campos de Seleção
 - <h:selectOneRadio>

```
<h:outputLabel value="Gênero " for="genero" />
<h:selectOneRadio value="#{componentesBean.genero}" id="genero" >
        <f:selectItem itemValue ="F" itemLabel="Feminino" />
        <f:selectItem itemValue ="M" itemLabel="Masculino" />
</h:selectOneRadio>
```

Gênero

Feminino Masculino

- Componentes Visuais
 - Campos de Seleção
 - <h:selectBooleanCheckbox>
 - □ Seleção Binária

```
<h:outputLabel value="Ex-Aluno: " for="exAluno" />
<h:selectBooleanCheckbox value="#{componentesBean.exAluno}" id="exAluno" />
```

Ex-Aluno: 🗹

□ Caixas desse tipo devem ser vinculadas a propriedades booleanas.

- Componentes Visuais
 - Campos de Seleção
 - <h:selectManyCheckbox>
 - □ Seleção Múltipla
 - Opção Estática

```
<h:selectManyCheckbox value="#{componentesBean.cursosEscolhidosE}" id="cursosE" >
        <f:selectItem itemValue ="C1" itemLabel="P00 - Estático" />
        <f:selectItem itemValue ="C2" itemLabel="DRA - Estático" />
        </h:selectManyCheckbox>
```

```
Cursos:
```

POO - Estático
DRA - Estático

- Componentes Visuais
 - Campos de Seleção
 - <h:selectManyCheckbox>
 - □ Seleção Múltipla
 - Opção Dinâmica

```
<h:selectManyCheckbox

value="#{componentesBean.cursosEscolhidosD}" id="cursosD" >
    <f:selectItems value="#{componentesBean.cursos}"
        var="curso" itemValue="#{curso.codigo}" itemLabel="#{curso.nome}"/>
        </h:selectManyCheckbox>
```

```
Cursos:

POO - Dinâmico

DRA - Dinâmico
```

- Componentes Visuais
 - Campos de Seleção
 - <h:selectOneMenu>
 - □ Seleção Única
 - Opção Dinâmica

```
<h:selectOneMenu

value="#{componentesBean.cursosEscolhidos}" id="cursosSOM" >
        <f:selectItems value="#{componentesBean.cursos}"
        var="curso" itemValue="#{curso.codigo}" itemLabel="#{curso.nome}"/>
        </h:selectOneMenu>
```

```
Cursos: POO - Dinâmico ▼
POO - Dinâmico
DRA - Dinâmico
```

- Componentes Visuais
 - Botões e Links
 - <h:commandButton> e <h:commandLink>
 - □ Enviam os dados de um formulário HTML para o servidor através do método POST do HTTP.
 - □ Podem ser associados a métodos de ação de um *Managed Bean* através dos atributos *action* e *actionListener*.

```
<h:commandButton value="Adiciona curso" action="#{cursosBean.adicionaCurso}"/>
<h:commandLink value="Remove curso" action="#{cursosBean.removeCurso}"/>
```

Adiciona curso

Remove curso

- Componentes Visuais
 - Botões e Links
 - <h:commandButton> e <h:commandLink>
 - □ *action*: deve ser usado para executar uma lógica de negócio ou indicar navegação entre páginas.

```
@ManagedBean
public class TesteBean {
   private int numero;

   public void incrementaNumero() {
     this.numero = numero + 1;
   }

   // GETTERS E SETTERS
}
```

```
<h:form>
    <h:commandButton value="Incrementa" action="#{testeBean.incrementaNumero}" />
    </h:form>
```

- Componentes Visuais
 - Botões e Links
 - <h:commandButton> e <h:commandLink>
 - □ *action*: deve ser usado para executar uma lógica de negócio ou indicar navegação entre páginas.

<h:commandButton value="Salva" action="#{produtoBean.salva}" />

```
@ManagedBean
public class ProdutoBean {
  private Produto produto = new Produto();
  private List<Produto> produtos = new ArrayList<Produto>();

  public String salva() {
    this.produtos.add(this.produto);
    this.produto = new Produto();
    return "lista-produtos";
  }

// GETTERS E SETTERS
}
```

- Componentes Visuais
 - Botões e Links
 - <h:button> e <h:link>
 - □ Realizam requisições HTTP do tipo GET.
 - □ Utiliza o atributo *outcome* para definir a página de destino.
 - ☐ Esses componentes não são utilizados para submeter formulários HTML.

```
<h:button value="Lista de cursos" outcome="lista-cursos" />
<h:link value="Home" outcome="home" />
```

Lista de cursos

<u>Home</u>

- Componentes Visuais
 - Botões e Links
 - <h:outputLink>
 - □ Permite apenas a criação de links HTML que realizam requisições do tipo GET.
 - ☐ A URL da requisição é definida explicitamente no atributo *value*.

```
<h:outputLink value="http://www.ifam.edu.br">
    Página do IFAM
</h:outputLink>
```

Página do IFAM

- Componentes Visuais
 - **Imagens**
 - <h:graphicImage>

```
<h:graphicImage value="img/download.png" width="50px" height="50px" />
```

- Componentes Visuais
 - Tabelas
 - <h:dataTable>

- CódigoNomeAdicionar turma0POO DinâmicoAdicionar turma1DRA DinâmicoAdicionar turma
- □ Atributo value: associa uma lista de elementos à tabela

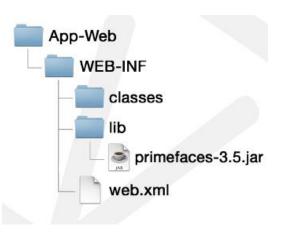
```
<h:dataTable value="#{componentesBean.cursos}" var="curso">
<h:column>
   <f:facet name="header">Código </f:facet> #{curso.codigo}
</h:column>
<h:column>
   <f:facet name="header">Nome </f:facet> #{curso.nome}
</h:column>
<h:column>
   <f:facet name="header">Adicionar turma </f:facet>
   <h:commandLink value=" Adicionar turma " action="#{componentesBean.adicionarTurma(curso)}" />
</h:column >
</h:dataTable >
```

- Componentes Visuais
 - Pratique!

Email: joyce@ Senha: ••• Cadastrar **Email** Senha Ação jms@ 123 Remover mds@ 456 Remover

PrimeFaces

- Biblioteca de componentes ricos para aplicações criadas com Java Server Faces.
- Instalação

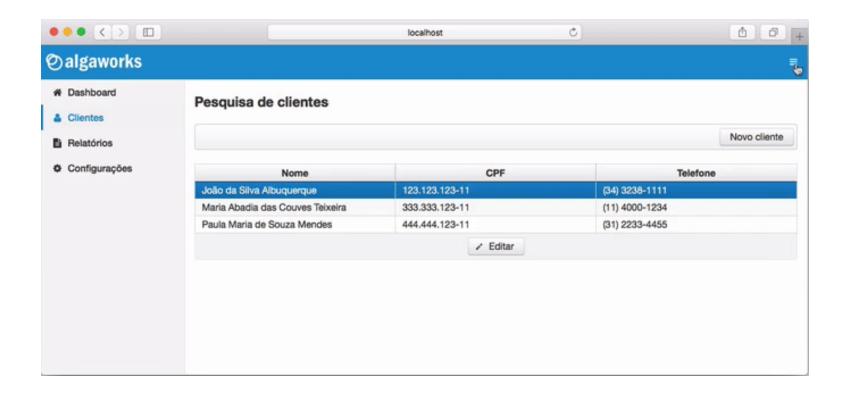


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui">
<h:head>
    <title>K19 Treinamentos</title>
</h:head>
<h:body>
    ...
</h:body>

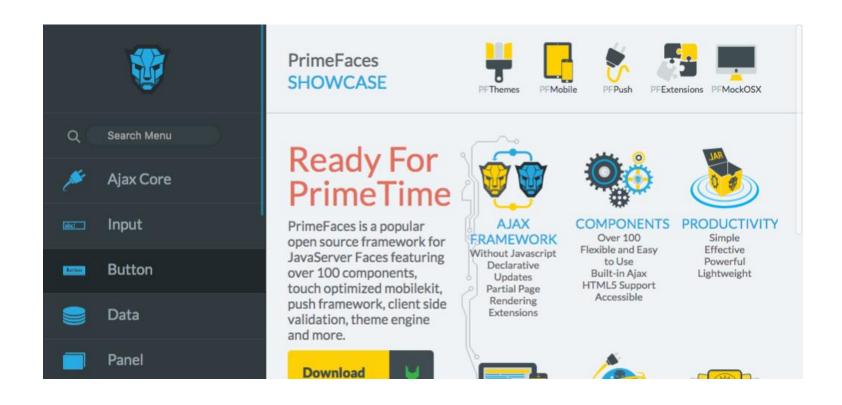
    ...
</h:hbody>
</html>
```

PrimeFaces

Responsividade

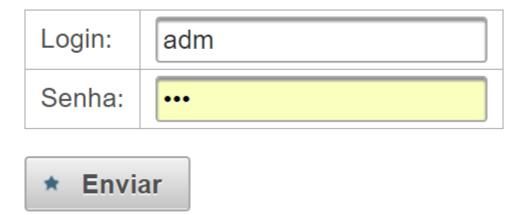


- PrimeFaces
 - https://www.primefaces.org/showcase/

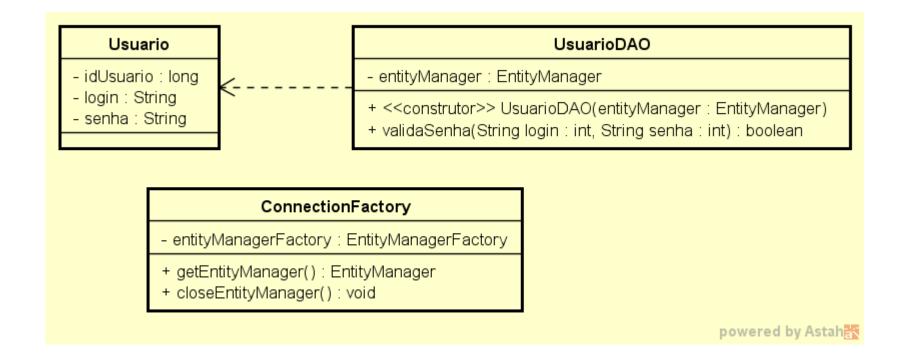


- Desenvolvimento com JSF + PrimeFaces + JPA
 - Tutorial adaptado do conteúdo disponível em:
 - http://www.devmedia.com.br/java-web-criando-uma-tela-de-login-com-jpa-jsf-primefaces-e-mysql/32456
 - Pré-requisito
 - Serviço do MySQL ativo.
 - Schema de banco de dados criado (Ex. sysControleAcademico)
 - Tabela criada no schema de banco de dados referenciado (Ex. usuario)

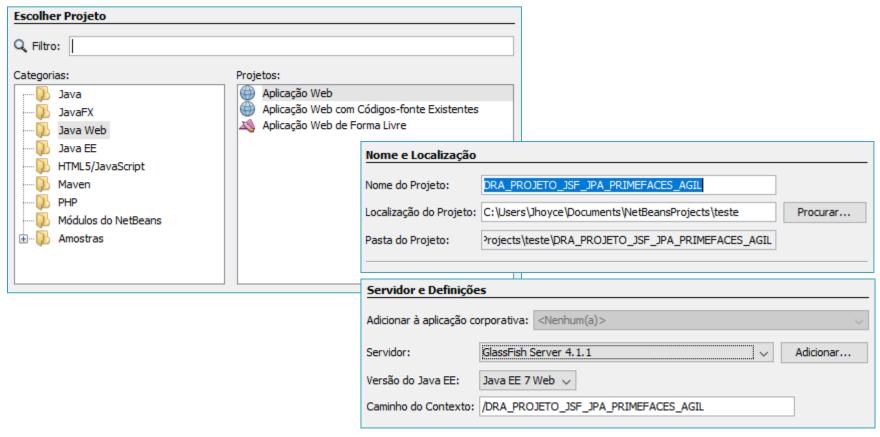
- Desenvolvimento com JSF + PrimeFaces + JPA
 - Formulário de Login



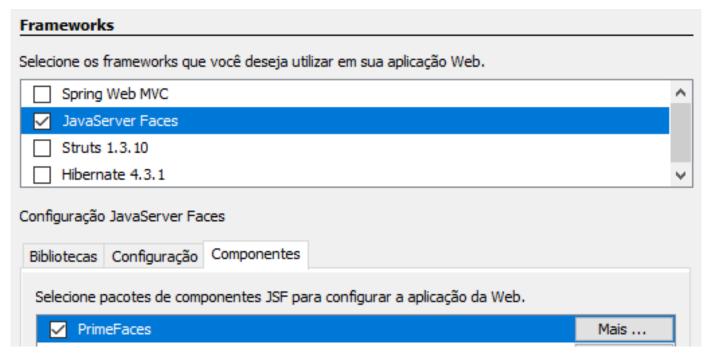
- Desenvolvimento com JSF + PrimeFaces + JPA
 - Formulário de *Login*
 - Classes Principais



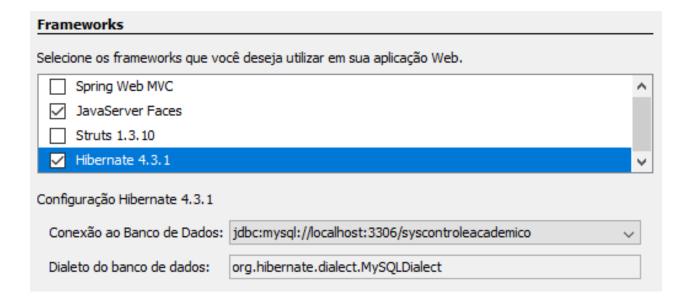
- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando o Projeto



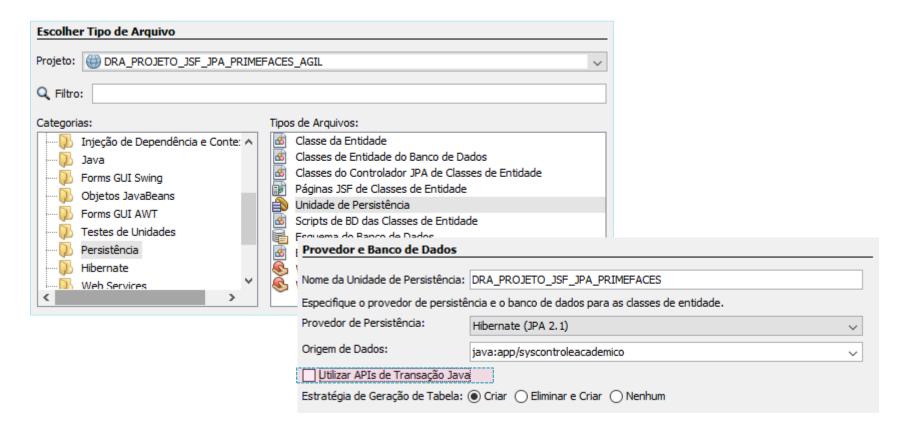
- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando o Projeto



- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando o Projeto



- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando Unidade de persistência (persistence.xml)



- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando Unidade de persistência (persistence.xml)

ConnectionFactory

- entityManagerFactory : EntityManagerFactory
- + getEntityManager(): EntityManager
- + closeEntityManager(): void

Desenvolvimento com JSF + PrimeFaces + JPA

Criando Classe de Conexão (ConnectionFactory.java)

```
public class ConnectionFactory {
    private static EntityManagerFactory entityManagerFactory;
    public static EntityManager getEntityManager() {
        if (entityManagerFactory==null|| !entityManagerFactory.isOpen()) {
            System.out.println("Criando EntityManager....");
            entityManagerFactory =
            Persistence.createEntityManagerFactory("DRA PROJETO JSF JPA PRIMEF
        return entityManagerFactory.createEntityManager();
    public static void closeEntityFactory(){
        entityManagerFactory.close();
```

- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando Classe de Entidade (<u>Usuario.java</u>)

```
@Entity
public class Usuario {
    @Id
    @GeneratedValue
    private Long idUsuario;
    @Column(unique=true)
    private String login;
    private String senha;

//gets e sets
```

Usuario

- idUsuario : long
- login : String
- senha : String

- entityManager : EntityManager

- + <<construtor>> UsuarioDAO(entityManager : EntityManager)

UsuarioDAO

+ validaSenha(usuario : Usuario) : boolean

- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando Classe de Entidade DAO (UsuarioDAO.java)

```
public class UsuarioDAO {
    private final EntityManager entityManager;
    UsuarioDAO (EntityManager entityManager) {
        this.entityManager = entityManager;
    public boolean validaSenha(Usuario usuario) {
        //monta consulta
        TypedQuery<Long> query = entityManager.createQuery(
            "select count(u) from Usuario as u "
              + "where u.login = :login "
              + " and u.senha = :senha ", Long.class);
        query.setParameter("login", usuario.qetLogin());
        query.setParameter("senha", usuario.getSenha());
        long quantidade = query.getSingleResult();
        if(quantidade > 0) {return true;}else{return false;}
```

Login:	adm
Senha:	•••
* Envi	ar

- Desenvolvimento com JSF + PrimeFaces + JPA
 - ManagedBean (<u>ValidaSenhaBean.java</u>)

```
@ManagedBean
□public class ValidaSenhaBean {
     private Usuario usuario = new Usuario() ;
     private UsuarioDAO dao = new UsuarioDAO(ConnectionFactory.getEntityManager());
     public String validaSenha() {
         boolean validou = dao.validaSenha(usuario);
         if(validou){
              return "/index";
         }else{
              FacesContext.getCurrentInstance().addMessage(
                 null,
                 new FacesMessage (FacesMessage.SEVERITY ERROR,
                          "Usuário não encontrado!",
                          "Erro no Login!"));
                      return "/faces/formLogin";
     public Usuario getUsuario() {
     public void setUsuario (Usuario usuario) {
```

Login:	adm
Senha:	•••
★ Envi	ar

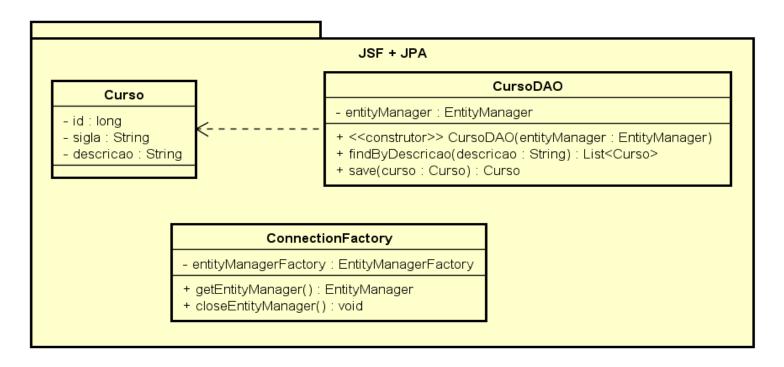
- Desenvolvimento com JSF + PrimeFaces + JPA
 - Criando a interface gráfica (<u>formLogin.xhtml</u>)

```
khtml xmlns="http://www.w3.org/1999/xhtml"
       xmlns:h="http://java.sun.com/jsf/html"
       xmlns:f="http://java.sun.com/jsf/core"
       xmlns:p="http://primefaces.org/ui">
 <h:head><title>Página JSF</title></h:head>
 <h:body>
    <h:form>
        <p:messages id="messages" />
        <p:panelGrid columns="2">
            <p:outputLabel for="login" value="Login:" />
            <p:inputText id="login" value="#{validaSenhaBean.usuario.login}" />
            <p:outputLabel for="senha" value="Senha:" />
            <p:password id="senha" value="#{validaSenhaBean.usuario.senha}" />
            <p:commandButton value="Enviar" icon="ui-icon-star"</pre>
                             action="#{validaSenhaBean.validaSenha}" ajax="false">
            </p:commandButton>
        </p:panelGrid>
    </h:form>
  </h:body>
</html>
```

- Em caso de erro na execução (jandex): Importar biblioteca *jandex.jar*
- https://www.youtube.com/watch?v=GRYkZR2SxfU

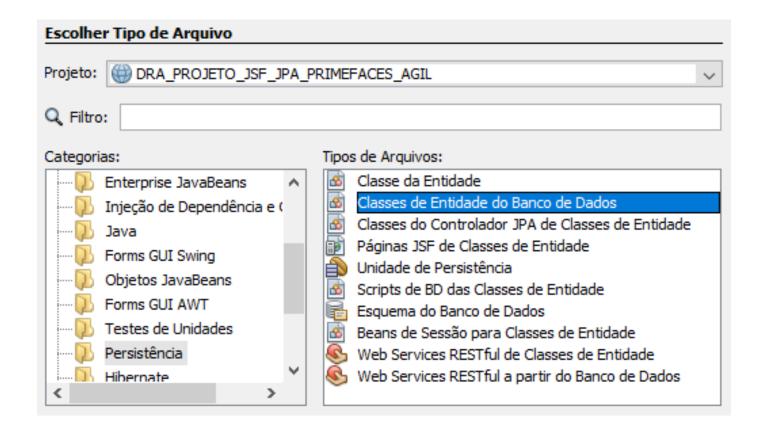


- Tarefa de Implementação
 - Desenvolvimento com JSF + PrimeFaces + JPA
 - ▶ Criar telas de listagem de cursos e cadastro de cursos.

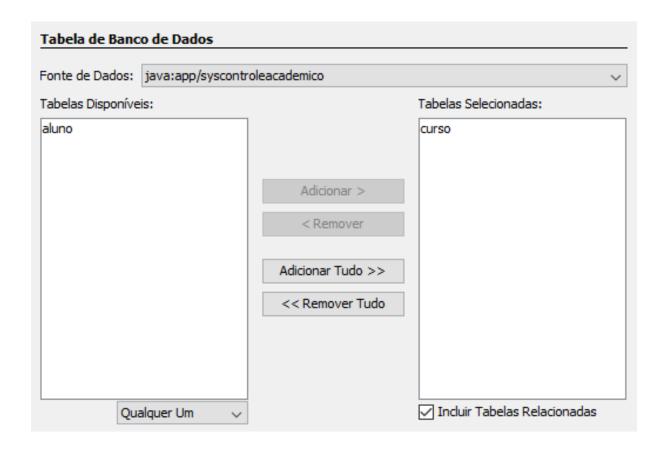


- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Tutorial adaptado do conteúdo disponível em:
 - https://www.scribd.com/document/104338116/Tutorial-Projeto-Webusando-JSF-com-CRUD-em-JPA-Netbeans-7
 - Pré-requisito
 - Serviço do MySQL ativo.
 - Schema de banco de dados criado (Ex. sysControleAcademico)
 - ▶ Tabela criada no schema de banco de dados referenciado (Ex. curso)

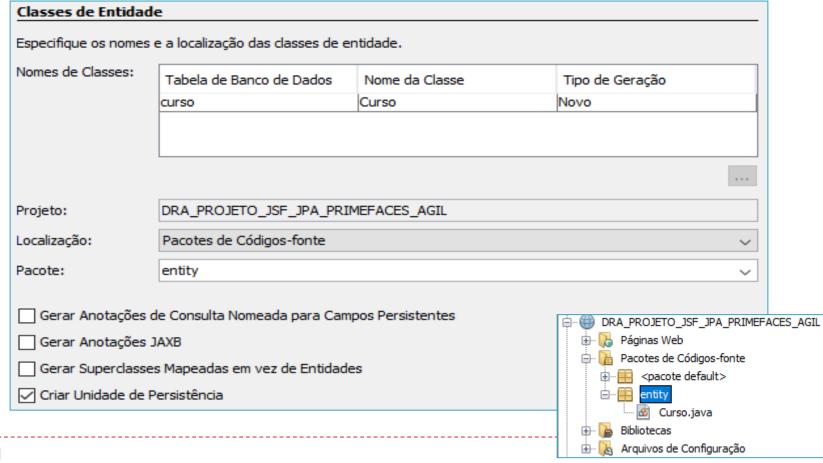
- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Classe de Entidade do Banco de Dados



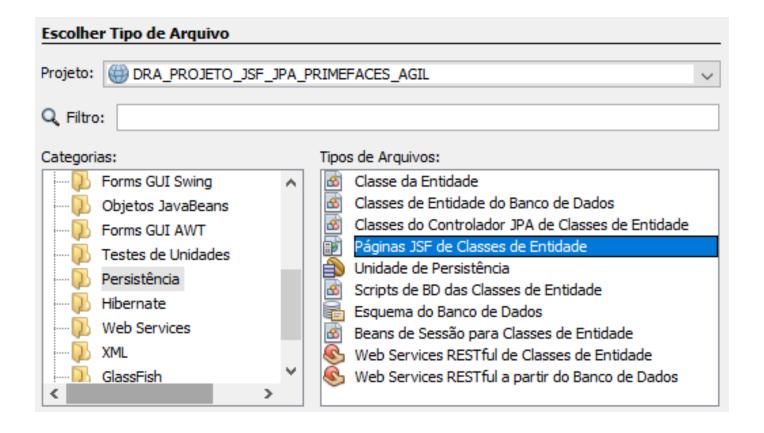
- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Classe de Entidade do Banco de Dados



- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Classe de Entidade do Banco de Dados



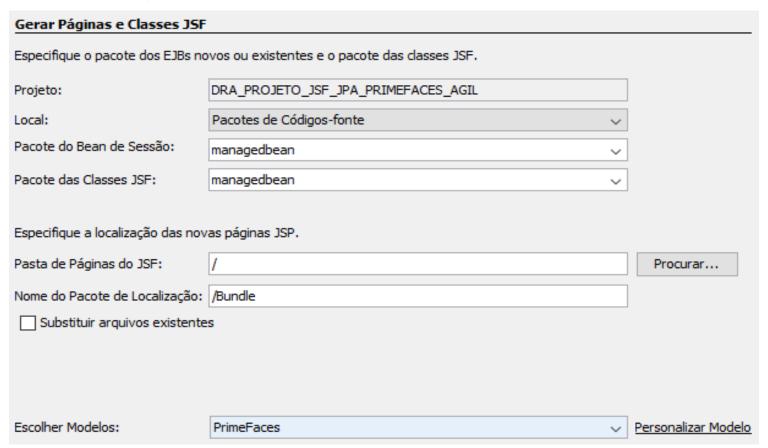
- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Páginas JSF de Classes de Entidade



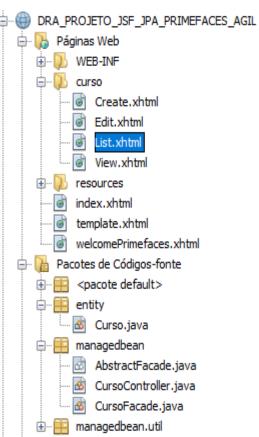
- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Páginas JSF de Classes de Entidade



- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Páginas JSF de Classes de Entidade



- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Criando Páginas JSF de Classes de Entidade

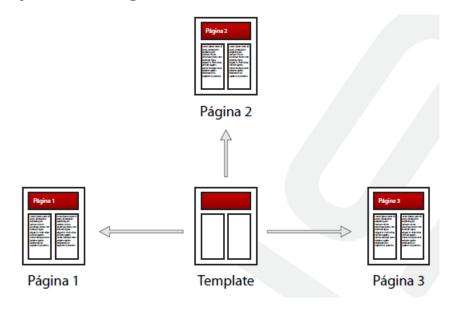


- Desenvolvimento <u>Ágil</u> com JSF + PrimeFaces + JPA
 - Resultado Final



EXTRAS

- Reutilização de Código
 - Facelets
 - Projeto definido para facilitar o processo de desenvolvimento e manutenção/gerenciamento das telas de uma aplicação JSF
 - Templates
 - Reutilização de código de Telas



Templates

- Definindo um template.
 - Criar um arquivo XHTML e definir o posicionamento dos trechos estáticos e dinâmicos.
 - ▶ Importar biblioteca de *tags* do *Facelets*

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
```

- Templates
 - Definindo um template.
 - <ui:insert>
 - ☐ Indica o posicionamento dos trechos dinâmicos.

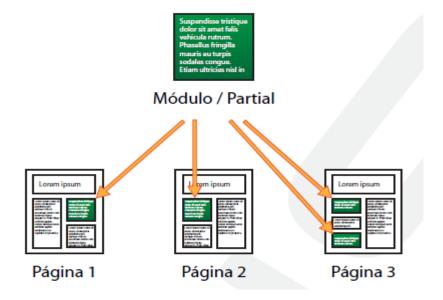
template.xhtml

- Templates
 - Utilizando um template.
 - <ui:composition>
 - □ Indica o posicionamento dos trechos dinâmicos.

```
<ui:composition template="template.xhtml"</pre>
xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">
                                                                      ******Cabeçalho*****
<ui:define name ="conteudo">
   <h:form>
                                                           Nome:
                                                                                Enviar
   <h:outputLabel value ="Nome:" for ="campo-nome" />
   <h:inputText id="campo-nome" />
                                                                       ******Rodapé*****
   <h:commandButton value="Enviar" />
   </h:form>
</ui:define>
</ui:composition>
                                              telaUsandoTemplate.xhtml
```

Modularização

- Os trechos estáticos ou dinâmicos definidos em um *template* possuem <u>posição</u> fixa.
- Em determinadas situações, é necessário tornar flexível o posicionamento de um determinado trecho.
- Nessas situações aplica-se o conceito de Modularização.



Modularização

▶ 1. Coloca-se o conteúdo do módulo em um arquivo XHTML separado.

2. Faz-se a inclusão do arquivo XHTML onde se fizer necessário.

```
<ui:include src="/formulario-de-contato.xhtml"/>
```

Modularização

Passando parâmetros entre módulos.

Adicionando CSS

```
resources
css
css
ch:outputStylesheet name="css/estilos.css" />
```

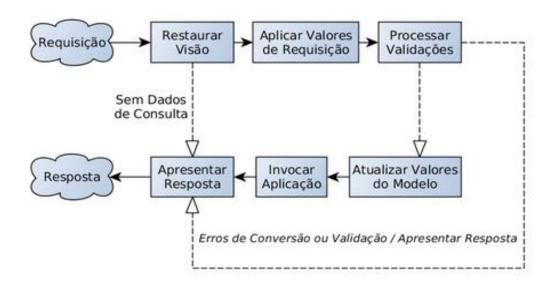
Adicionando JavaScript

```
<h:outputScript name="k19.js" library="javascript" target="head" />
```

Faces Servlet - Processamento de Requisição – Ciclo de Vida

Restaurar Visão

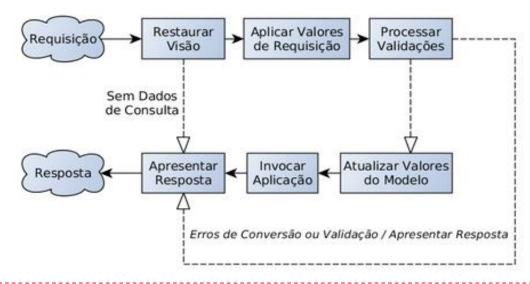
- □ O objetivo principal desta fase é construir/reconstruir a árvore de componentes.
- ☐ Salva o estado da árvore no objeto FacesContext.



Faces Servlet - Processamento de Requisição – Ciclo de Vida

Aplicar Valores de Requisição

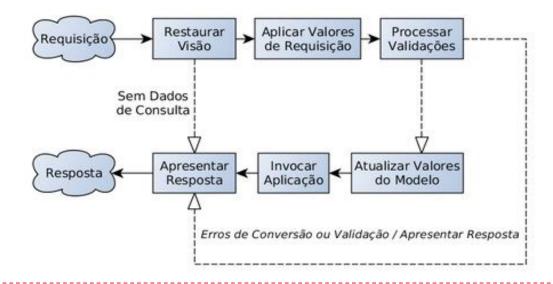
- ☐ A árvore construída na etapa anterior é percorrida e cada um dos seus componentes é "decodificado".
 - □ No processo de decodificação, cada componente extrai da requisição os dados associados a esse componente e se atualiza com essas informações.
 - Os eventos de ação são identificados e registrados para serem tratados posteriormente.



Faces Servlet - Processamento de Requisição – Ciclo de Vida

Processar Validações

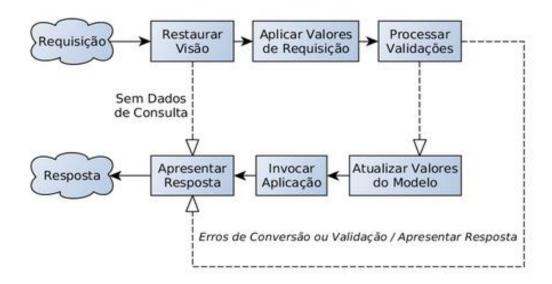
- □ Os componentes com valores submetidos pelo usuário são convertidos e validados, caso haja algum validador registrado para esse componente.
 - □ Se ocorrer algum problema, mensagens de erro são adicionadas ao **contexto** da requisição e o fluxo é redirecionado para a fase "Apresentar Resposta".
 - Caso contrário, processo continua na fase "Atualizar Valores do Modelo".



Faces Servlet - Processamento de Requisição – Ciclo de Vida

4. Atualizar Valores de Modelo

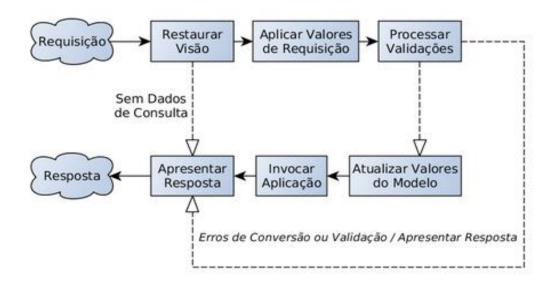
 Os valores contidos em cada componente da árvore, já convertidos e validados na fase anterior, são armazenados em propriedades de objetos definidos pela aplicação (Managed Beans)



Faces Servlet - Processamento de Requisição – Ciclo de Vida

Invocar aplicação

- ☐ As tarefas correspondentes ao evento que disparou a requisição (normalmente um clique em um botão ou link) serão executadas.
- □ A próxima tela a ser apresentada ao usuário é determinada pelo resultado do método que implementa a lógica de negócio executado nesta fase.



Faces Servlet - Processamento de Requisição – Ciclo de Vida

6. Apresentar resposta

- □ A próxima tela é gerada e enviada ao navegador do usuário.
- □ Uma representação desta tela também é armazenada a fim de ser usada na fase "Restaurar Visão" da próxima requisição.

