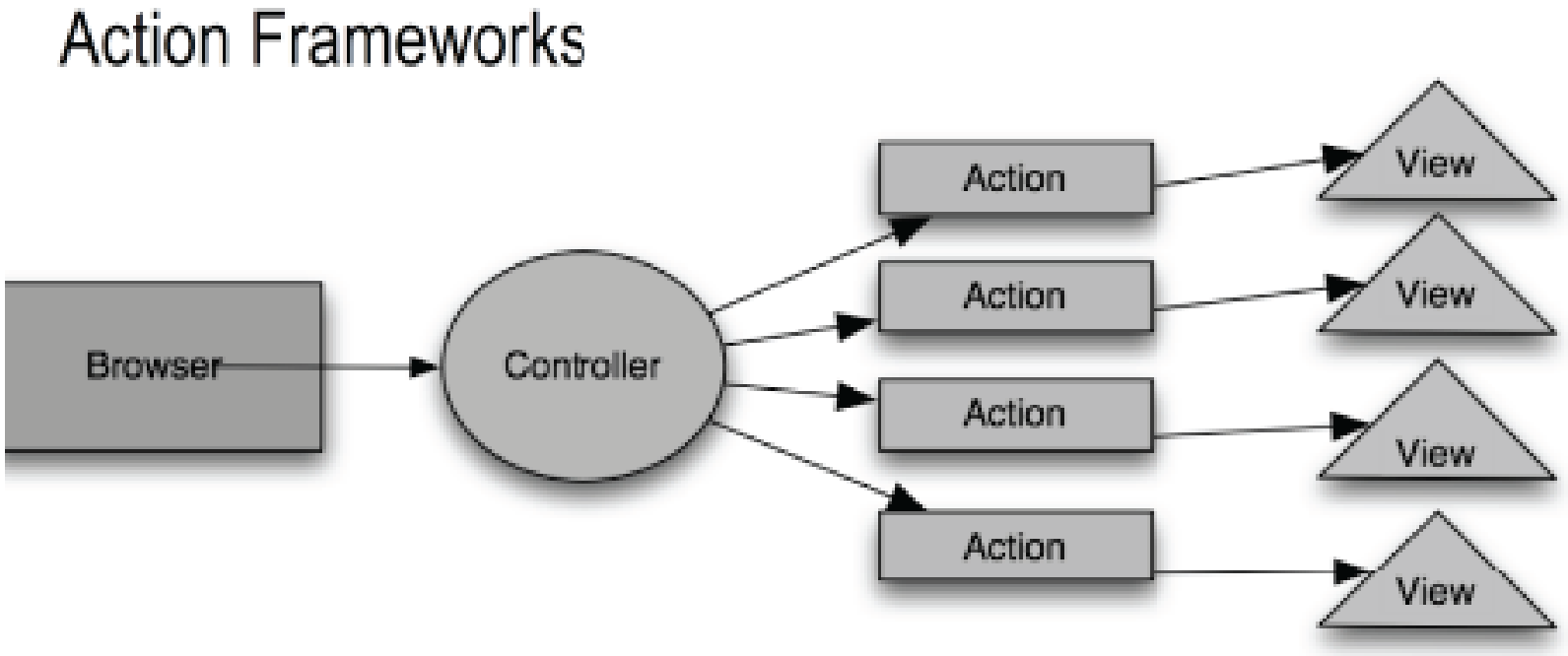
PROGRAMAÇÃO WEB

Java EE com Java Server Faces

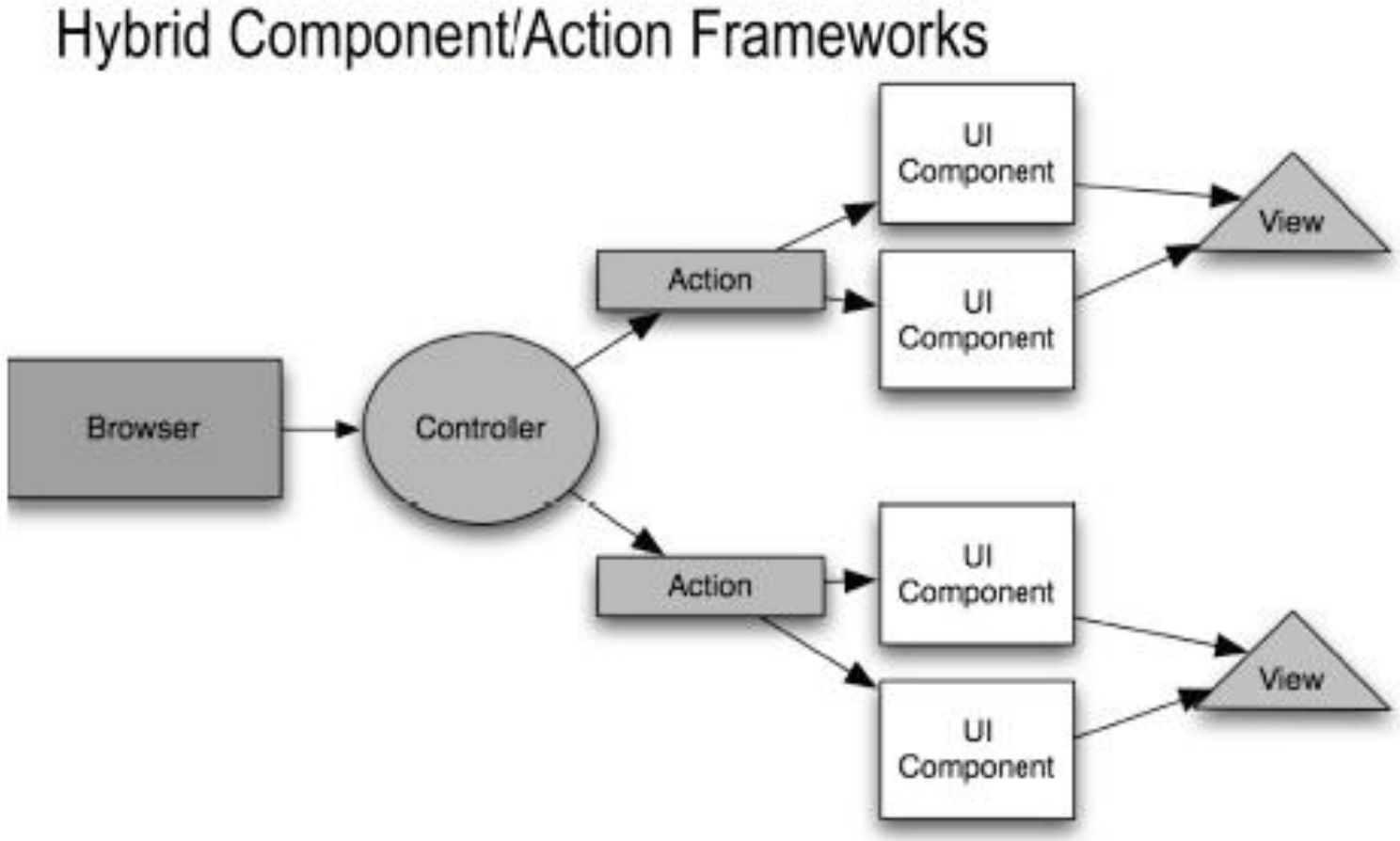
Web Frameworks

- Classificação:*Action Frameworks:Struts, Struts 2, Rails.*
- *Hybrid Frameworks:Tapestry, Wicket.*
- *UI Component Frameworks:JSF.*

Web Frameworks

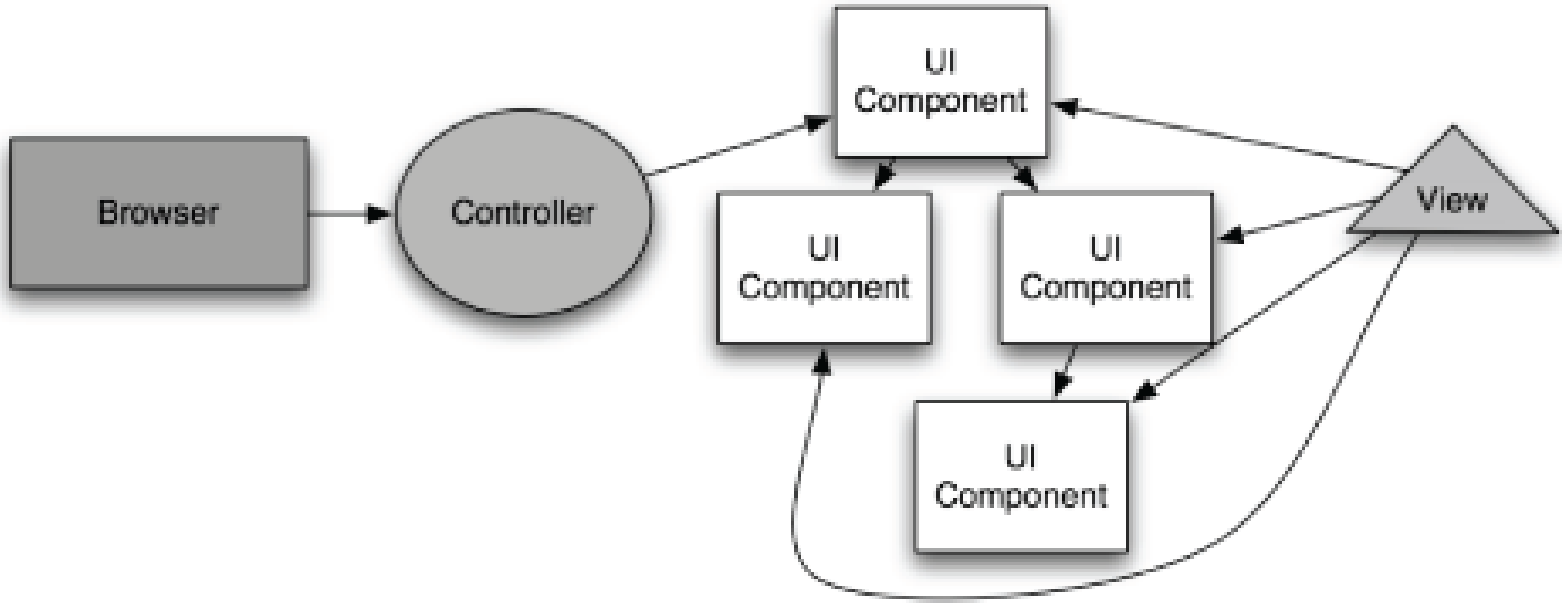


Web Frameworks



Web Frameworks

True UI Component Frameworks



JSF

- O que é o JavaServer Faces?
  - *Um arcabouço* para o desenvolvimento de aplicações para a web.
  - O que é arcabouço?
    - Um conjunto de mecanismos.

JSF

- O JSF, então, por definição é:Um arcabouço, **padrão arquitetural**, que fornece um gabarito extensível àaplicações para web, definindo aspectos estratégicos quanto a separação de **componentes**em pacotes lógicos ...

JSF

- Open source.
- Padrão JavaEE para desenvolvimento de aplicações web.
- It's a RAD-ical world!
  - Rapid Application Development: construir aplicações através de componentes de reuso.

RAD

- RAD →Produtividade, desenvolvimento *drag and drop*!
- Como?
  - RAD *tools*.
- RAD *toolslayers*, camadas:
  - Definição básica de componentes arquiteturais.
  - Um conjunto de componentes visuais.
  - Um arcabouço infraestrutural para aplicação.
  - A ferramenta (*tool*) em si.

JSF RAD

- JSF: Padrão Java RAD *web framework*!
- JSF RAD *layers*:
  - Definição básica de componentes arquiteturais.
  - Um conjunto de componentes visuais.
  - Um arcabouço infraestrutural para aplicação.

JSF

- JSF provê:
  - Componentes de interface de usuário (*UI*) padrões.
  - Definições arquiteturais para uso de componentes de terceiros.

JSF

- Componentes UI, qual seria o diferencial?
  - Orientados a eventos!
  - Suporte **nativo** a manutenção de estado entre múltiplas requisições.
- Épossível manter o sincronismo entre os componentes UI e **objetos Java**, responsáveis por coletar dados de entrada de usuário e por processar os eventos, os chamados ***backing beans***.

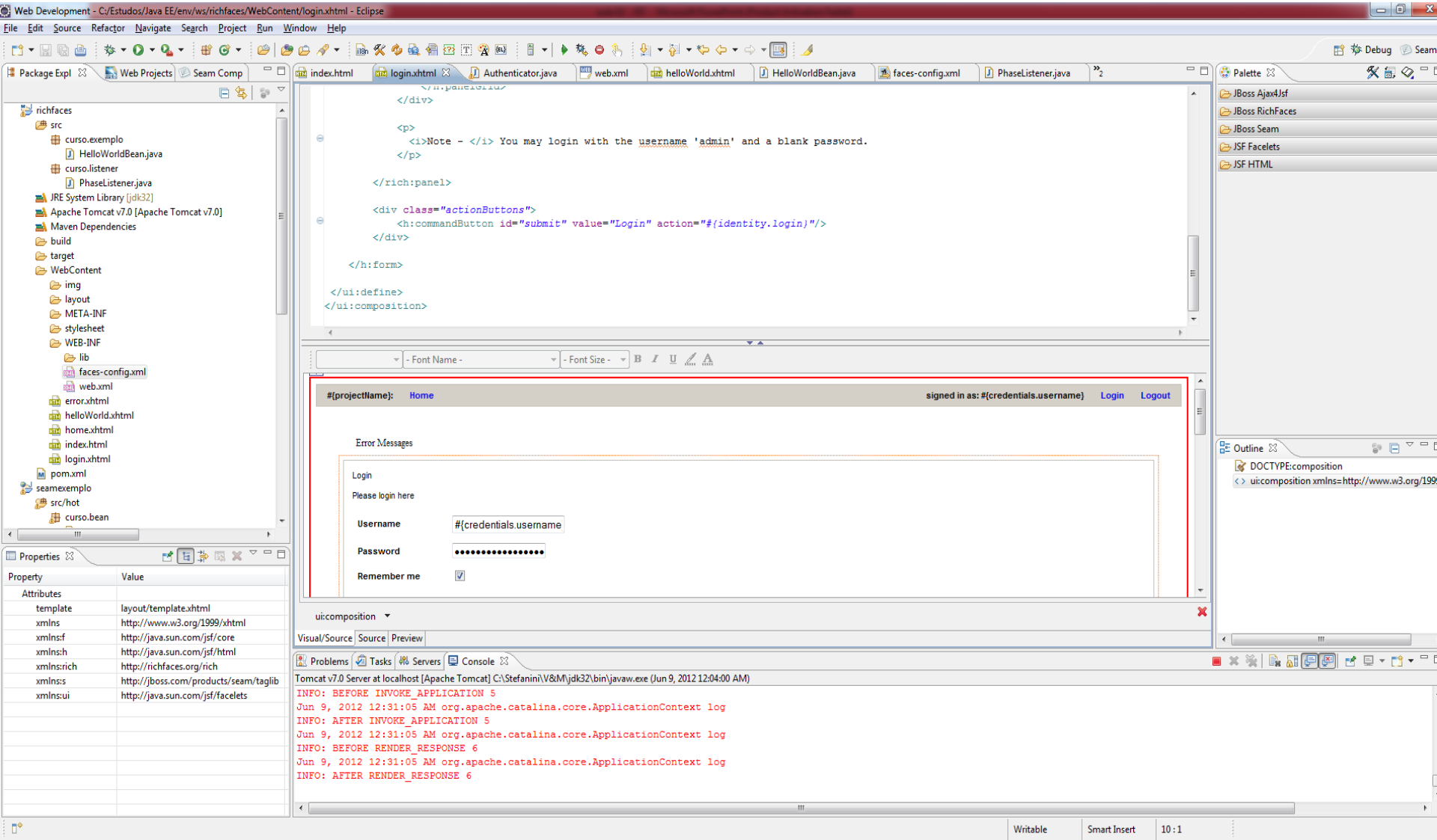
JSF

- JSF provê, ainda:
  - Componentes para validação e conversão de dados de entrada do usuário.
  - *Portlets*.
  - Mecanismos para definição e controle de navegação.
  - Suporte completo a internacionalização.

JSF

- JSF **não** provê:
  - As ferramentas (*tools*) de apoio ao desenvolvimento *drag and drop*. Isto fica a cargo de terceiros (*JBoss Tools*, por exemplo).

JBoss Tools


Web Development - C:/Estudos/Java EE/env/ws/richfaces/WebContent/login.xhtml - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Expl
Web Projects
Seam Comp
richfaces
src
curso.exemplo
HelloWorldBean.java
curso.listener
PhaseListener.java
JRE System Library [jdk32]
Apache Tomcat v7.0 [Apache Tomcat v7.0]
Maven Dependencies
build
target
WebContent
img
layout
META-INF
stylesheet
WEB-INF
lib
faces-config.xml
web.xml
error.xhtml
helloWorld.xhtml
home.xhtml
index.html
login.xhtml
pom.xml
seamexemplo
src/hot
curso.bean
Properties
Property
Value
Attributes
template
layout/template.xhtml
xmlns
http://www.w3.org/1999/xhtml
xmlns:f
http://java.sun.com/jsf/core
xmlns:h
http://java.sun.com/jsf/html
xmlns:rich
http://richfaces.org/rich
xmlns:s
http://jboss.com/products/seam/taglib
xmlns:ui
http://java.sun.com/jsf/facelets
index.html
login.xhtml
Authenticator.java
web.xml
helloWorld.xhtml
HelloWorldBean.java
faces-config.xml
PhaseListener.java

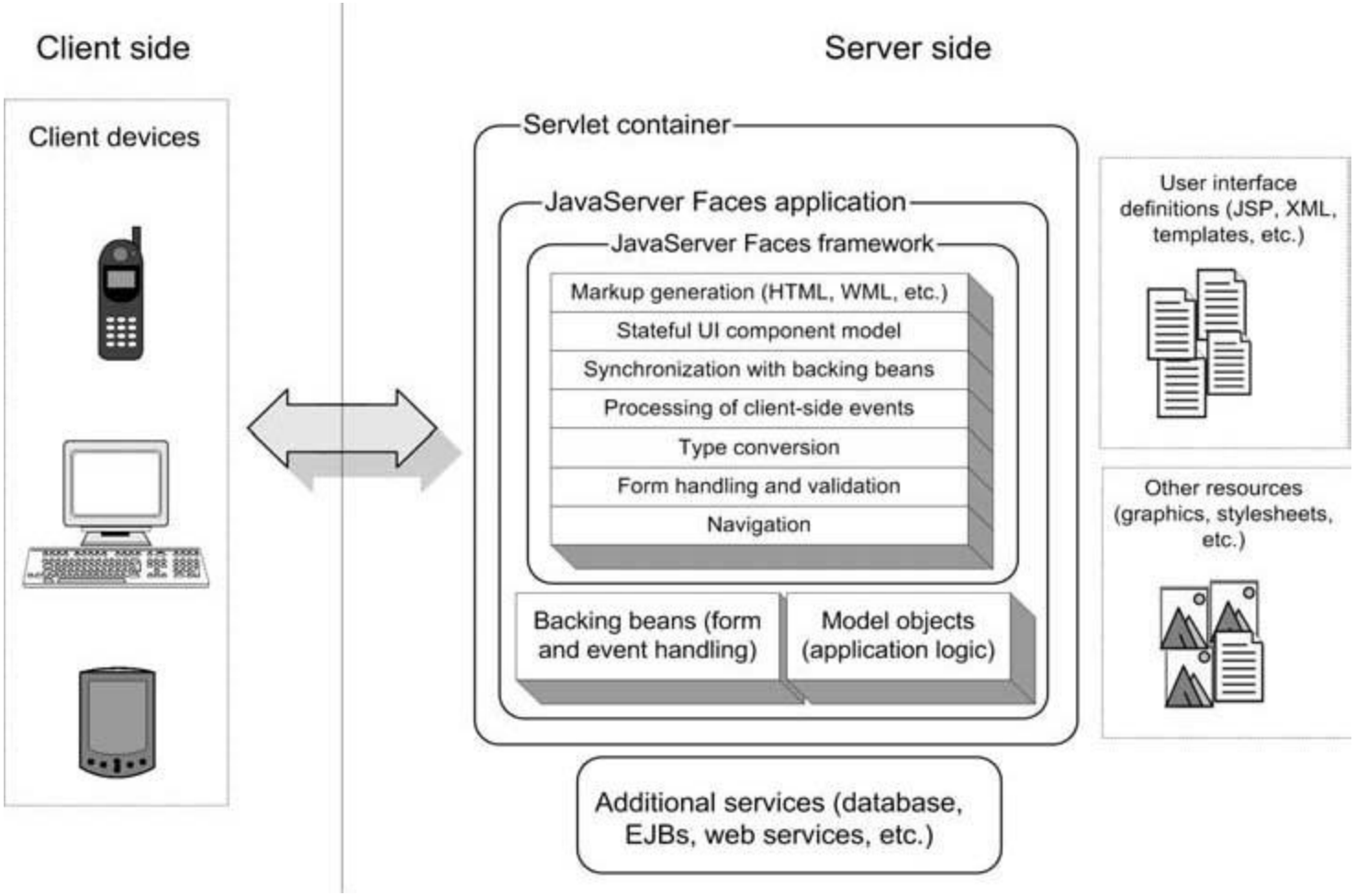
Note - You may login with the username 'admin' and a blank password.

</rich:panel>

<h:commandButton id="submit" value="Login" action="#{identity.login}"/>

</h:form>
</ui:define>
</ui:composition>
- Font Name -
- Font Size -
#{projectName}:
Home
signed in as: #{credentials.username}
Login
Logout
Error Messages
Login
Please login here
Username
#{credentials.username
Password
Remember me
ui:composition
Visual/Source
Source
Preview
Palette
JBoss Ajax4Jsf
JBoss RichFaces
JBoss Seam
JSF Facelets
JSF HTML
Outline
DOCTYPE:composition
ui:composition xmlns=http://www.w3.org/199
Problems
Tasks
Servers
Console
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Stefanini\V&M\jdk32\bin\javaw.exe (Jun 9, 2012 12:04:00 AM)
INFO: BEFORE INVOKE_APPLICATION 5
Jun 9, 2012 12:31:05 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER INVOKE_APPLICATION 5
Jun 9, 2012 12:31:05 AM org.apache.catalina.core.ApplicationContext log
INFO: BEFORE RENDER_RESPONSE 6
Jun 9, 2012 12:31:05 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER RENDER_RESPONSE 6
Writable
Smart Insert
10 : 1

Camadas do JSF



JSF

- JSF, uma nova tecnologia?
  - **Não.**
- Quais seriam as tecnologias envolvidas?
  - HTTP.
  - Servlets.
  - Portlets.
  - JavaBeans.
  - JSP.
  - Padrões de projeto.

HTTP

- O que éHTTP?
  - Protocolo formal de comunicação entre computadores em rede.
  - Basicamente em formato textual.
  - Dos sete métodos de solicitação, os mais usados são o GET e o POST.

HTTP

- **Não mantém estado.**
- Reescrita de URL e cookies são utilizados para controlar os usuários entre as solicitações.
- **HTTP não trafega conteúdo dinâmico.**

Servlets

- Qual seria o papel de Servlets em aplicações para a web? O que seria um Servlet?Classes de intermédio entre as requisições de um *browsere* o uso de aplicativos de um servidor.
- JSF torna possível o desenvolvimento de aplicações para web sem o uso direto de especificidades de HTTP e/ou Servlets.

Portlets

- Componentes visuais responsáveis por exibir informações de diferentes *datasources*em um **Portal**.Especificação JavaEE para *Portlet*, *Sun Portlet*, de 2003.

- JSF é aderente a *Portlet*API, ou seja, componentes JSF podem ser usados em *portlets*.

Java Beans

- Classes Java com propriedades acessíveis via métodos get e set?
  - **Não é somente isso!**

- JavaBeans é todo um arcabouço arquitetural especificado em vistas a um amplo suporte ferramental.

JSP

- JSP (*Java Server Pages*):
  - Documentos JSP incluem tanto partes estáticas quanto “dinâmicas”.
  - *Servlets* são classes especiais que implementam alguns serviços pré-definidos e que rodam em *containers* JSP.
  - Documentos JSP são convertidos em *servlets* quando são requisitados pela primeira vez.

JSP

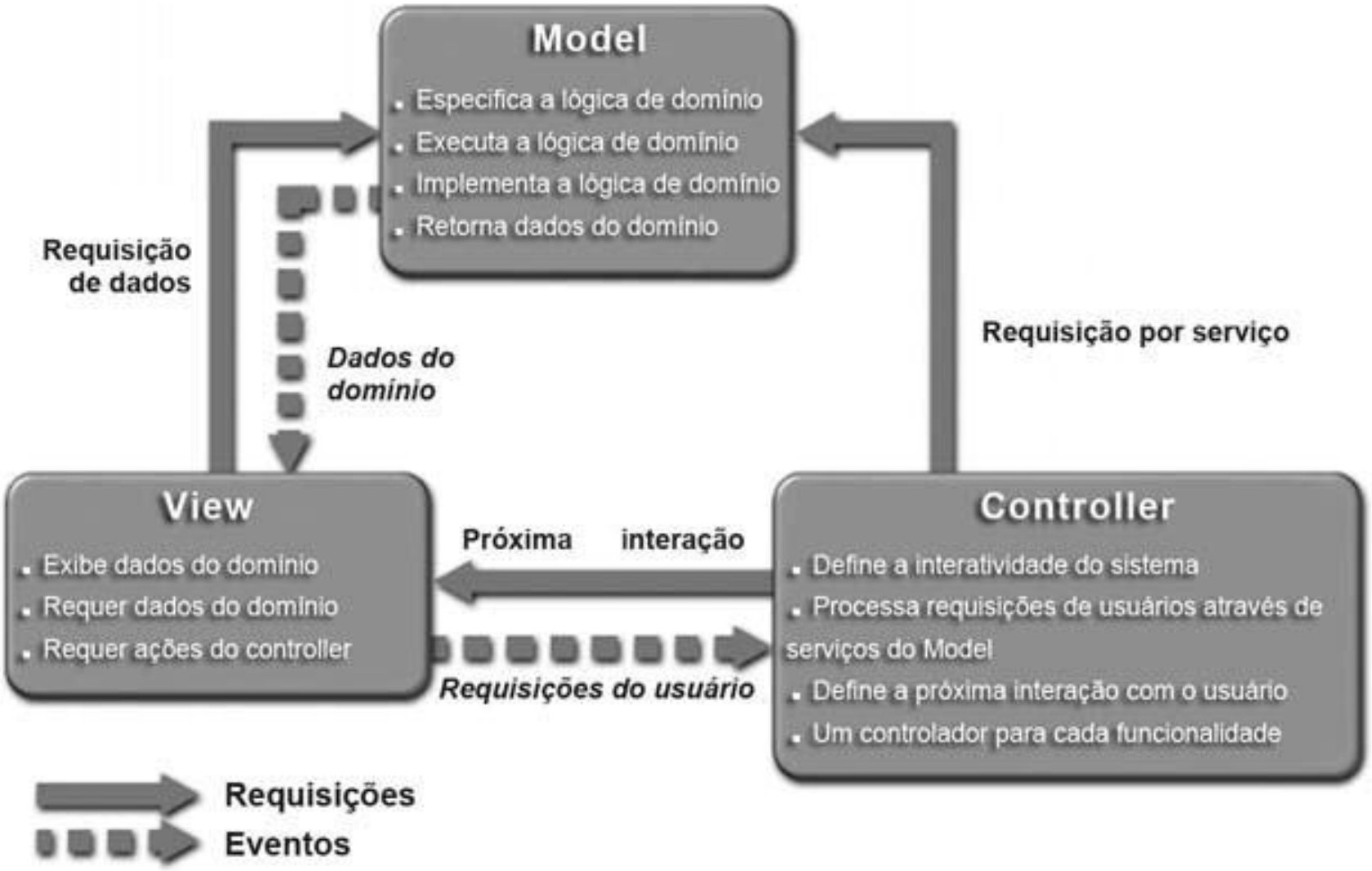
- Por ser um padrão JavaEE, o JSF referencia o padrão JavaEE para apresentação, o JSP.

- Porém, épossível trabalhar com JSF e outras tecnologias de apresentação. JSP não éa única alternativa. No curso iremos utilizar xhtml no lugar de jsps ☺

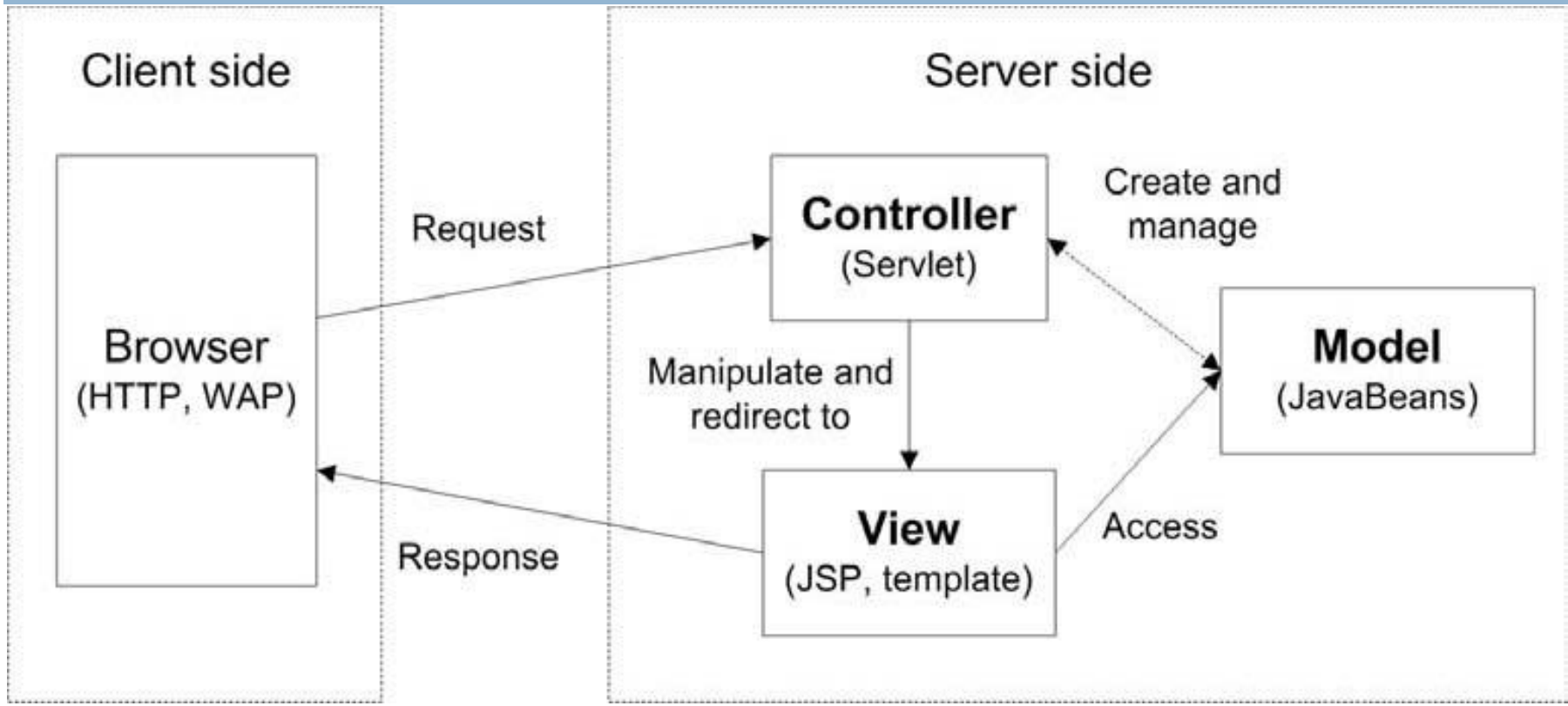
JSF

- JSF é um arcabouço orientado ao padrão MVC 2.

MVC


Model
Especifica a lógica de domínio
Executa a lógica de domínio
Implementa a lógica de domínio
Retorna dados do domínio
Requisição de dados
Dados do domínio
Requisição por serviço
View
Exibe dados do domínio
Requer dados do domínio
Requer ações do controller
Próxima interação
Controller
Define a interatividade do sistema
Processa requisições de usuários através de serviços do Model
Define a próxima interação com o usuário
Um controlador para cada funcionalidade
Requisições do usuário
Requisições
Eventos

MVC 2

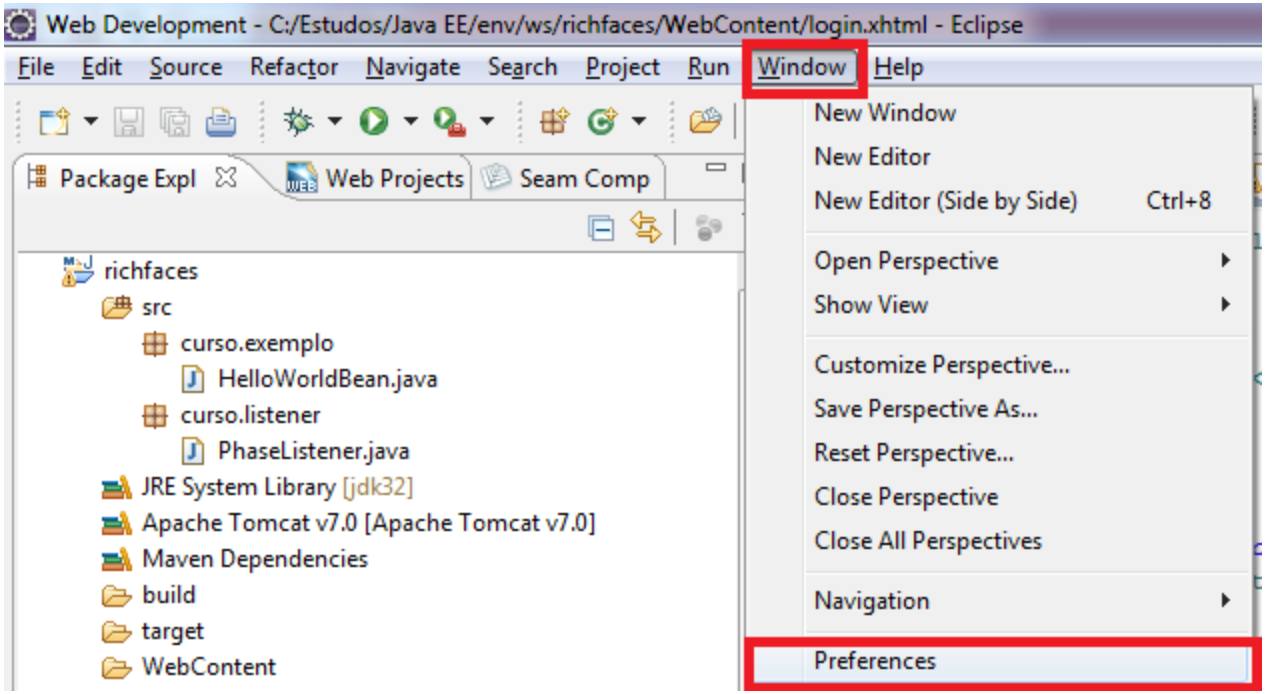

Client side
Server side
Browser
(HTTP, WAP)
Request
Response
Controller
(Servlet)
Create and
manage
Manipulate and
redirect to
Model
(JavaBeans)
View
(JSP, template)
Access

Java EE e JSF

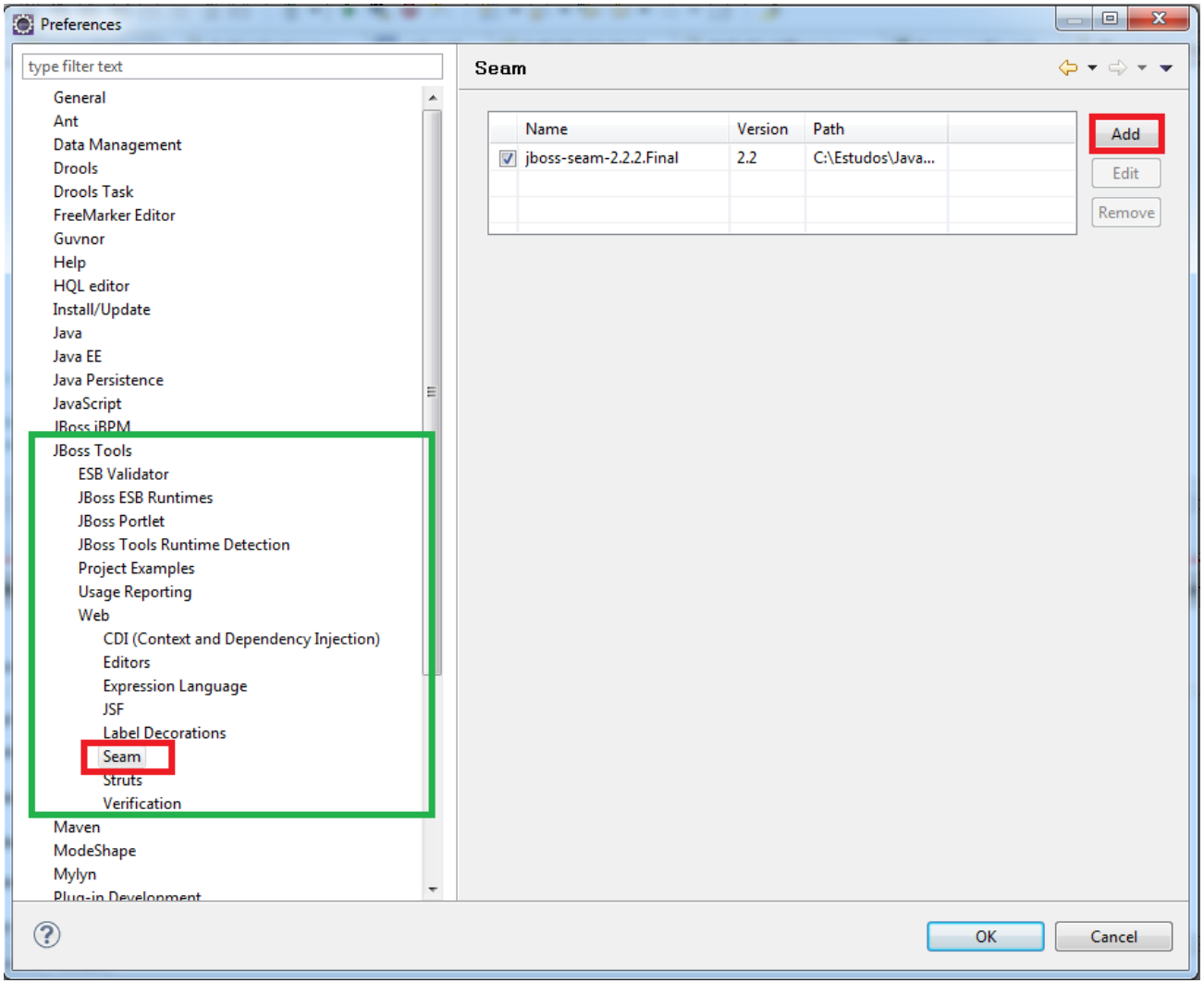
Prática 01

1. Configurar o JBoss Tools para uso.
2. Configurar o repositório Maven.
3. Importar um projeto JSF *template*.
4. Configurar o projeto JSF.
5. Implementar um simples Hello World usando os componentes básicos do JSF.
6. Implantar e executar a aplicação JSF.
7. Acessar o simples Hello World implementado.

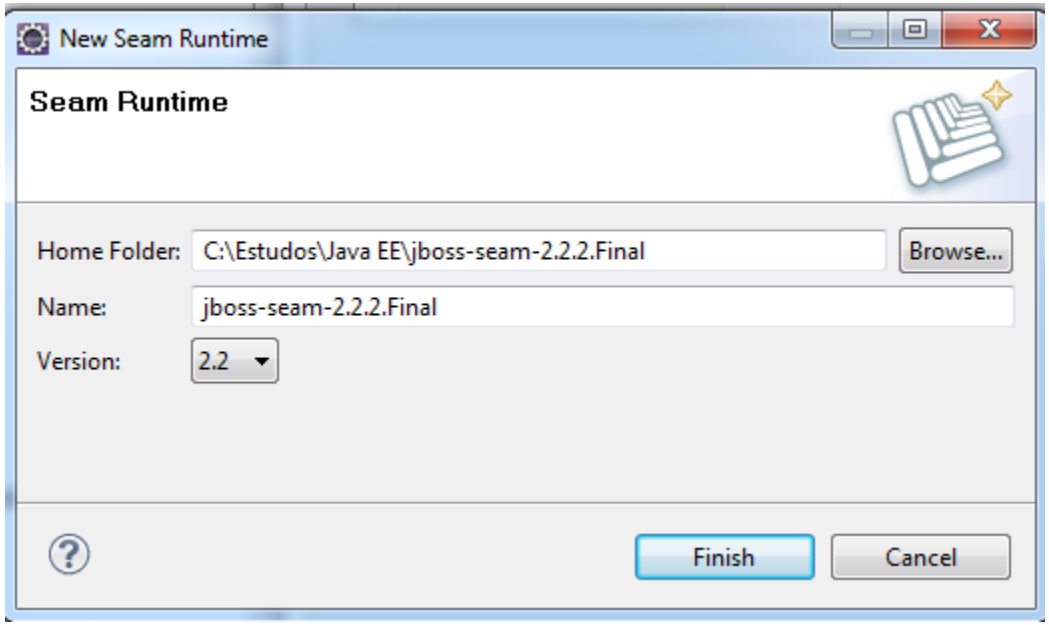
Prática 01 – Atividade 1



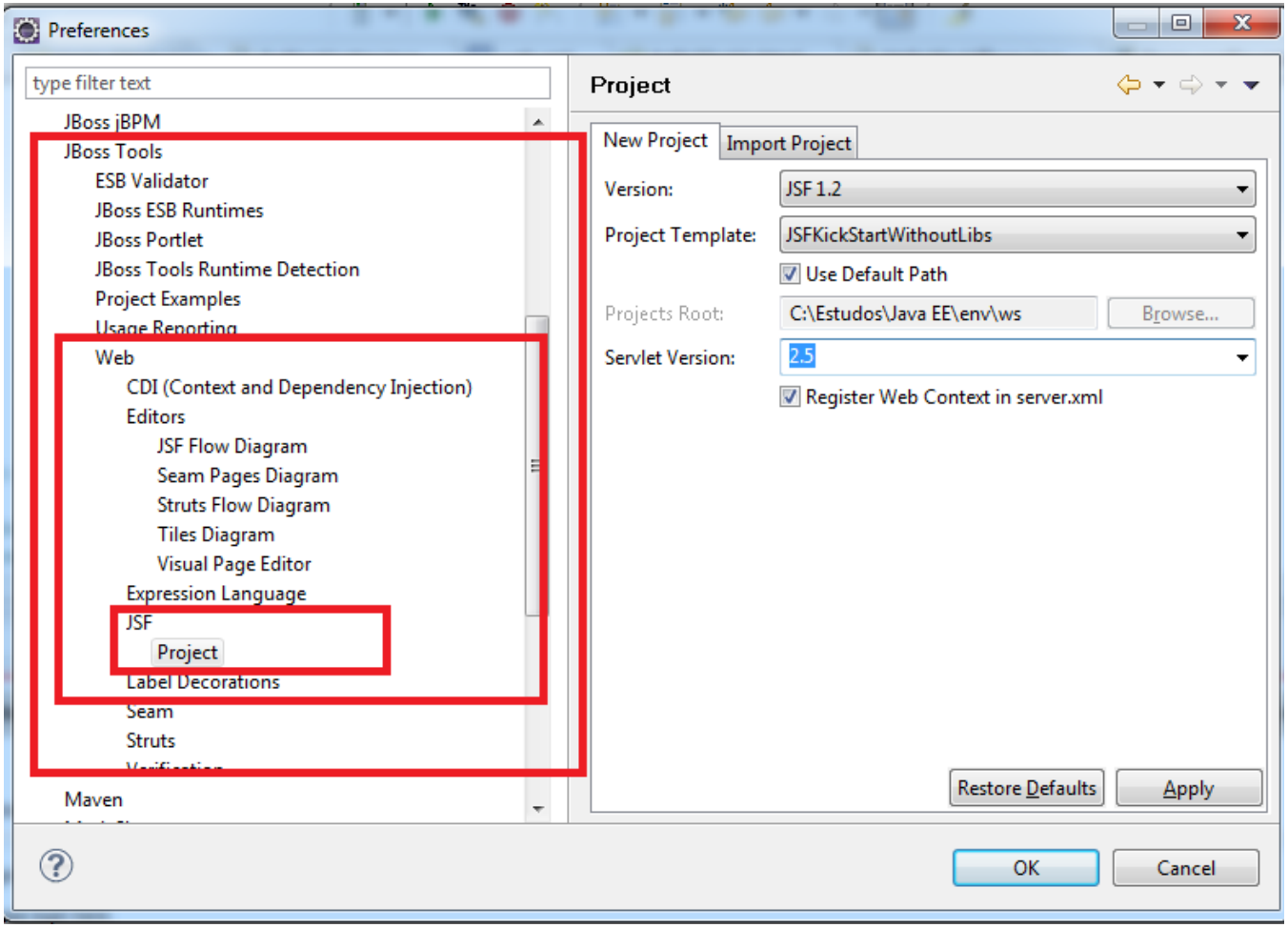
Prática 01 – Atividade 1



Prática 01 – Atividade 1



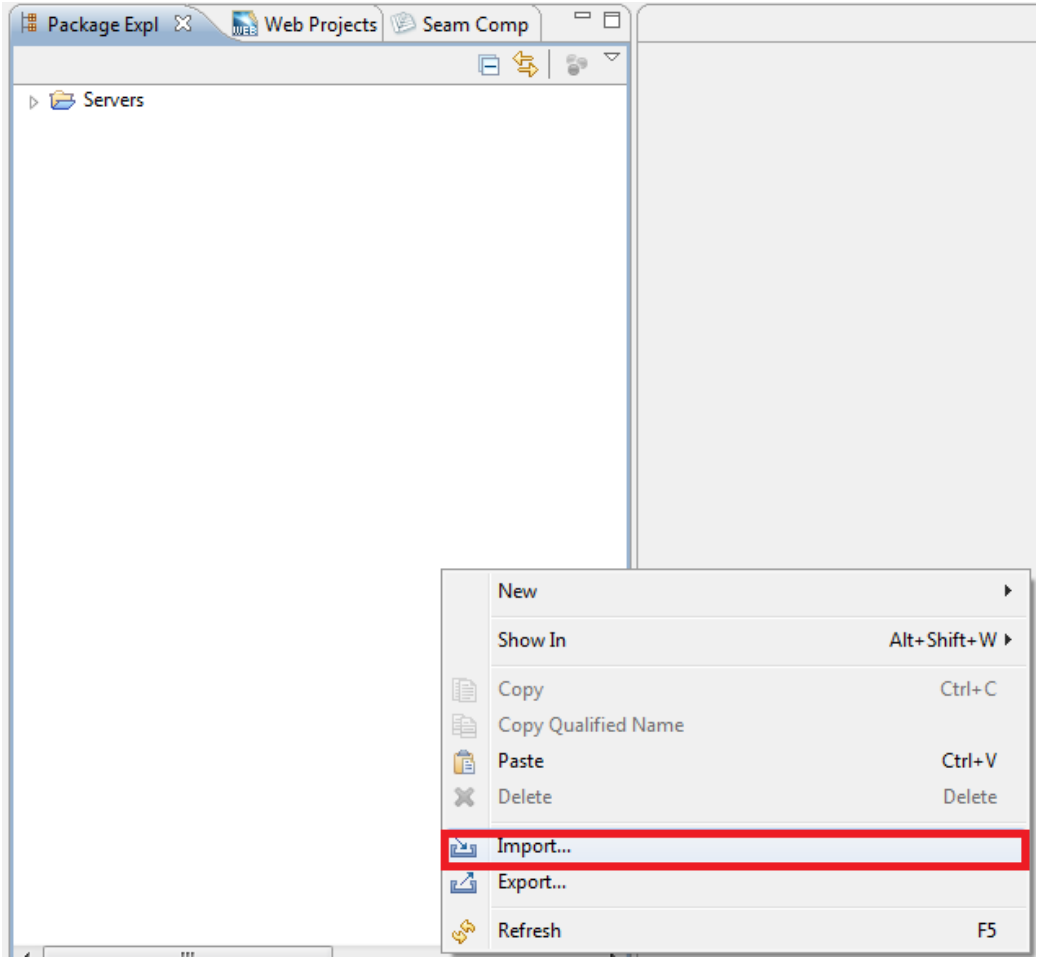
Prática 01 – Atividade 1



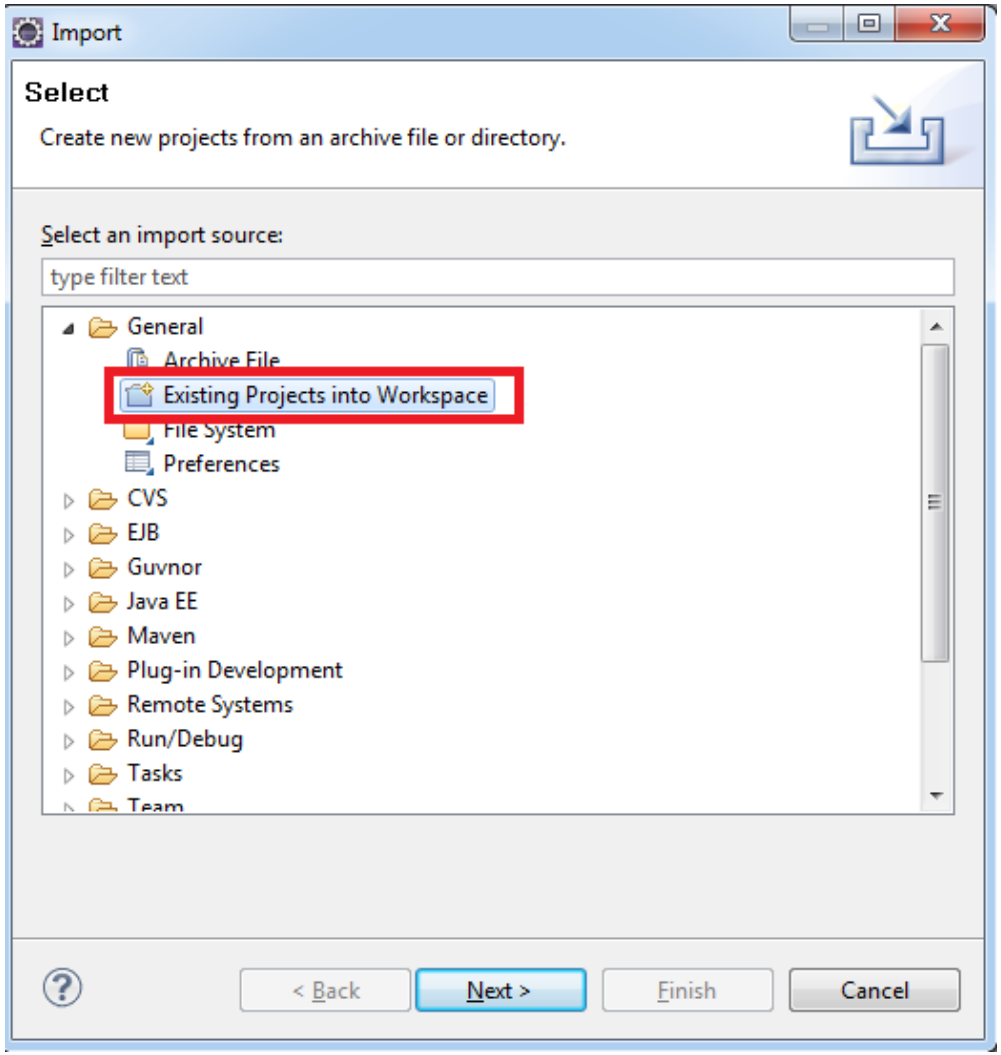
Prática 01 – Atividade 2

- Copiar o arquivo .m2.zip para <<CursoDesenWeb>>\<<user>>\.
- Descompactar o arquivo .m2.zip.

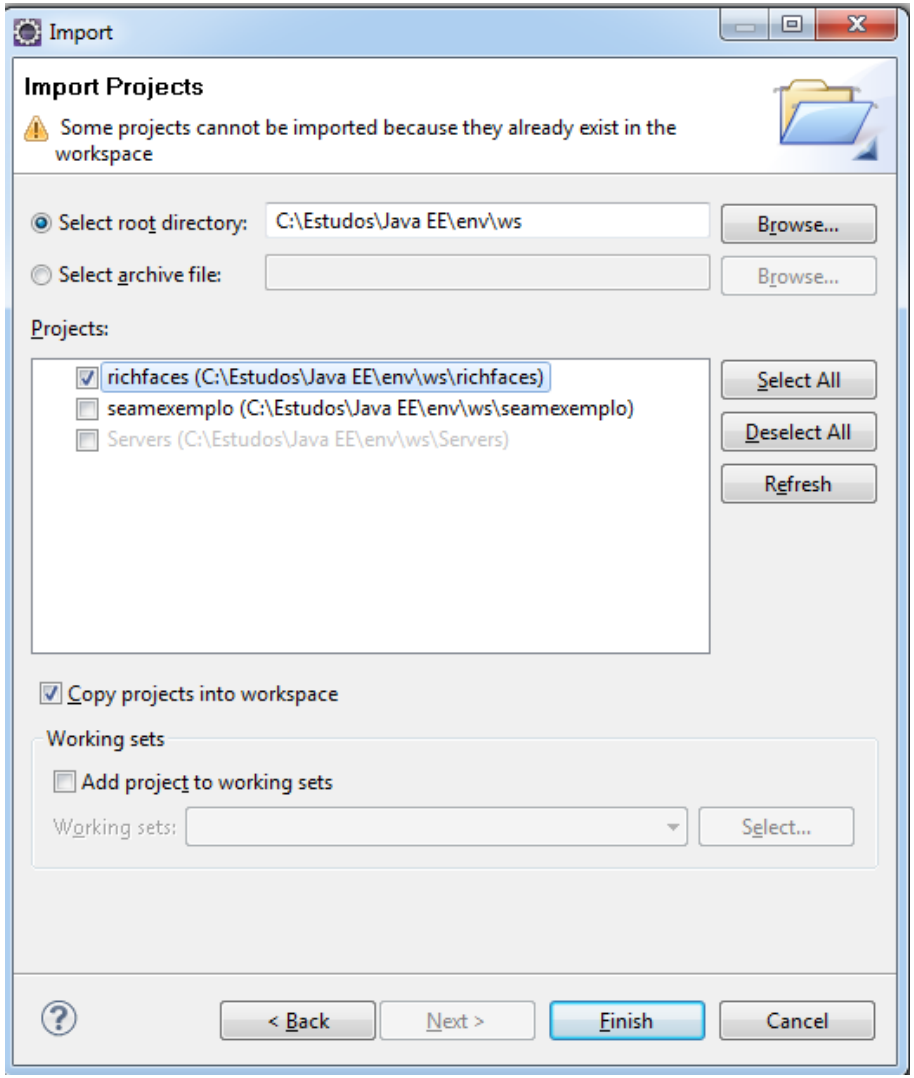
Prática 01 – Atividade 3



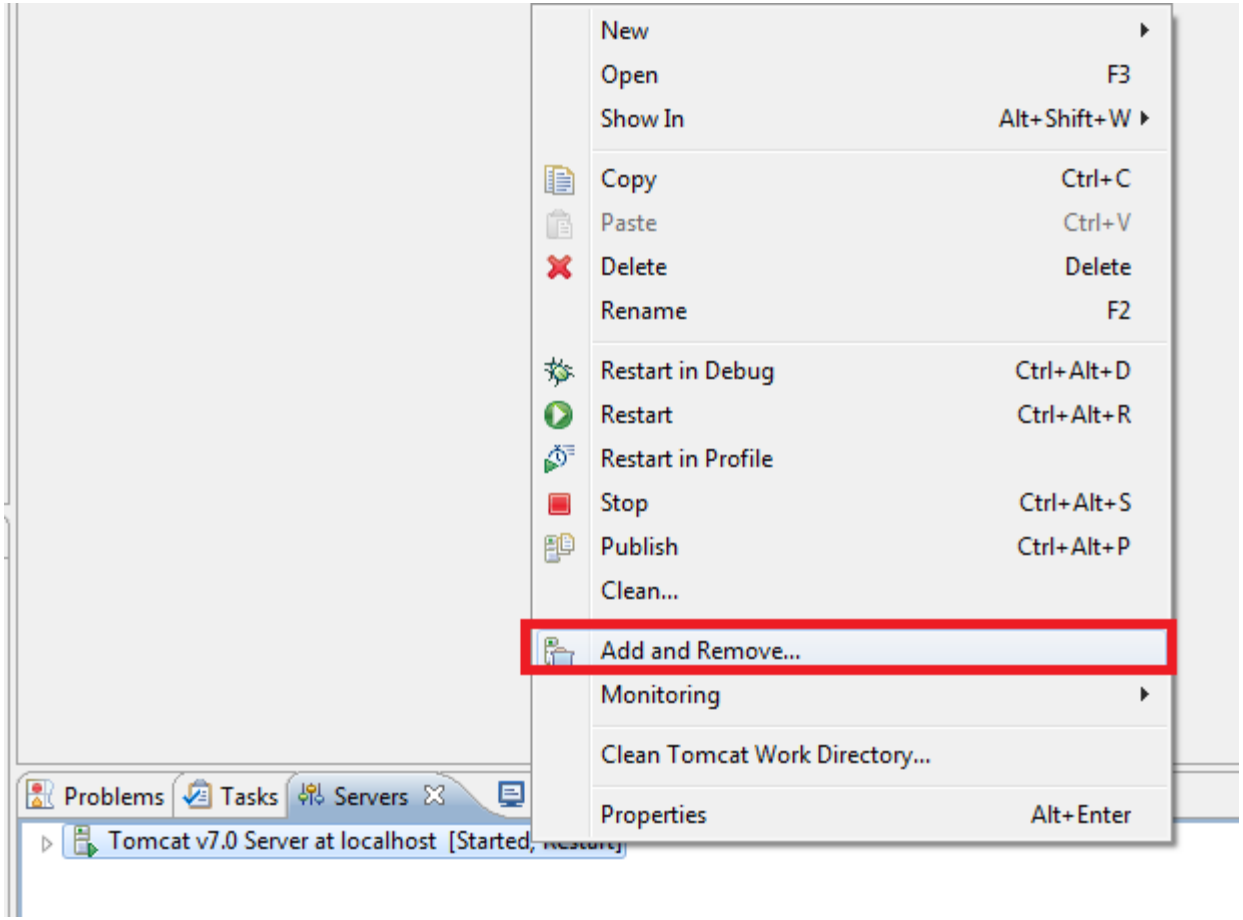
Prática 01 – Atividade 3



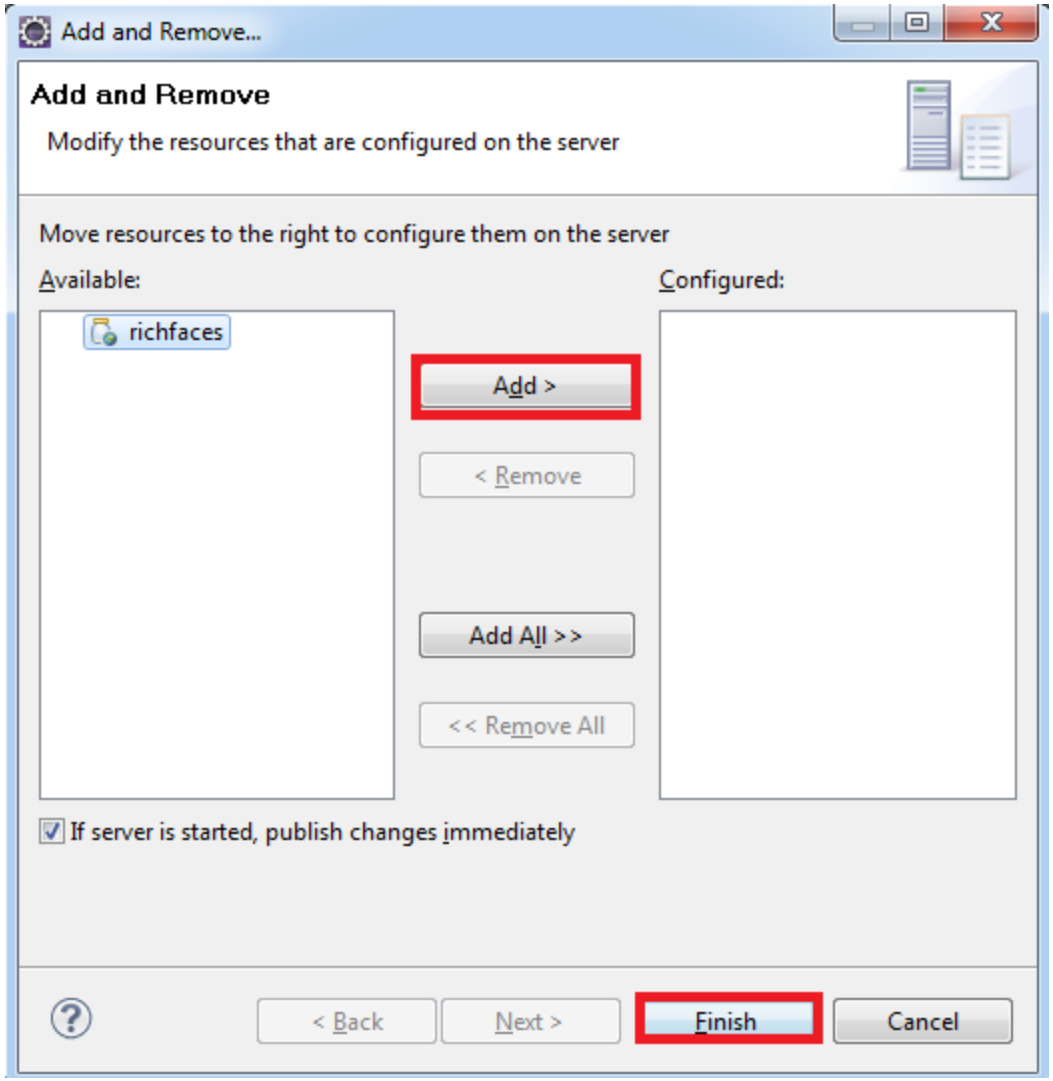
Prática 01 – Atividade 3



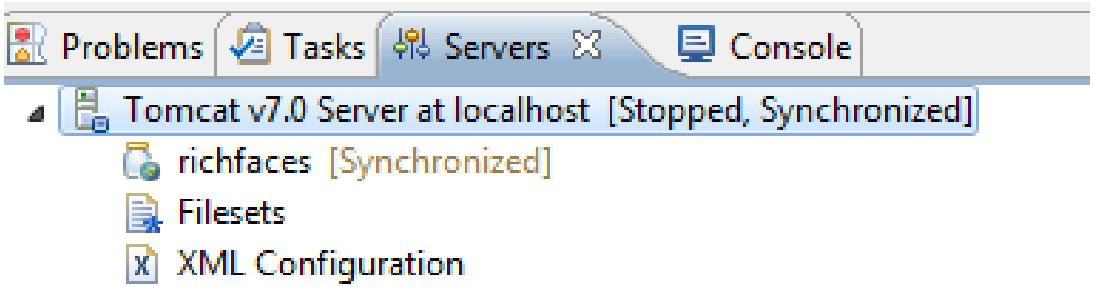
Prática 01 – Atividade 4



Prática 01 – Atividade 4



Prática 01 – Atividade 4



Prática 01 – Atividade 5

- Crie os arquivos:
  - helloWorld.xhtml;
  - HelloWorldBean.java;
- Registre o backing bean no faces-config.xml
- Altere o contéudo do xhtml e da classe java
- Inicie o servidor

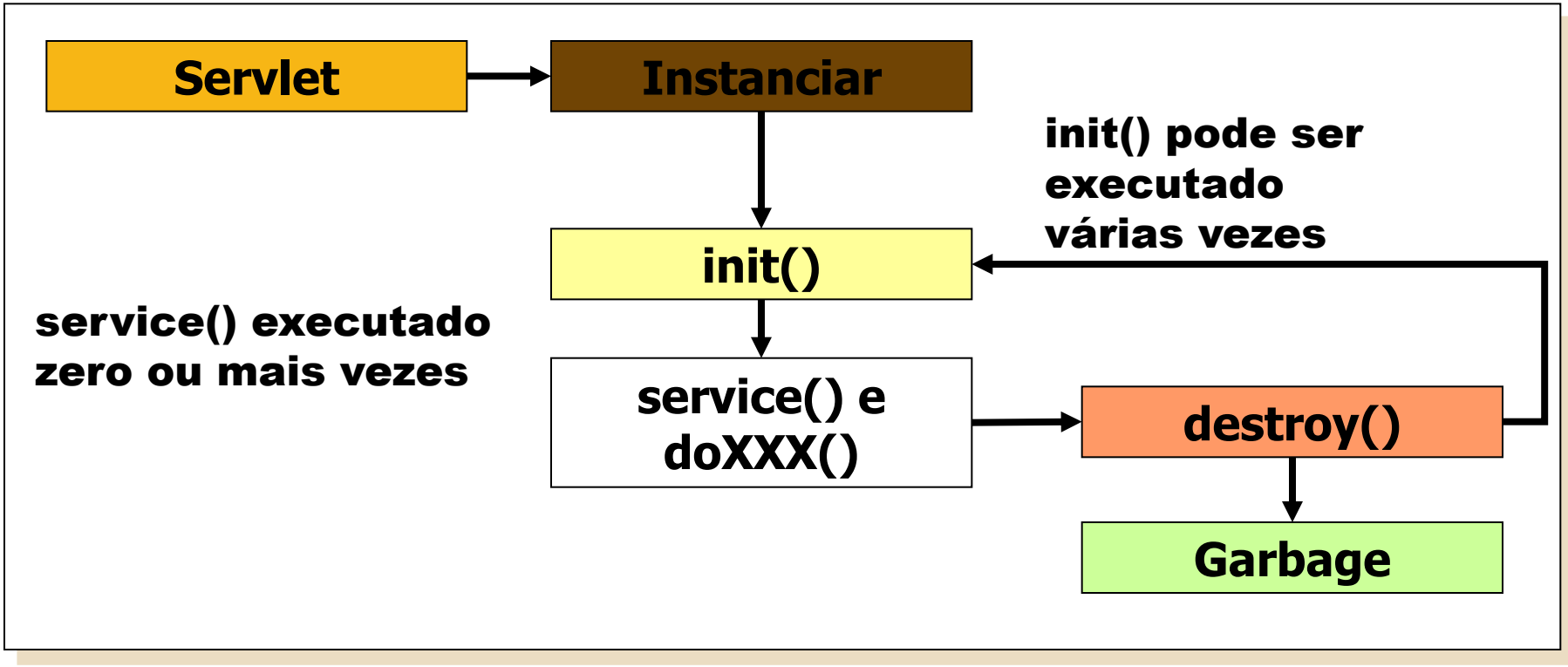
JSF

- Relembrando…
- O JSF provê:ComponentesUI orientados a eventos.
- Suporte **nativo** a manutençãode estadoentre múltiplas requisições.

JSF

- É possível manter o sincronismo entre os componentes UI e **objetos Java**, responsáveis por coletar dados de entrada de usuário e por processar os eventos, os chamados ***backing beans***.
- Como isto é feito pelo *framework* JSF?

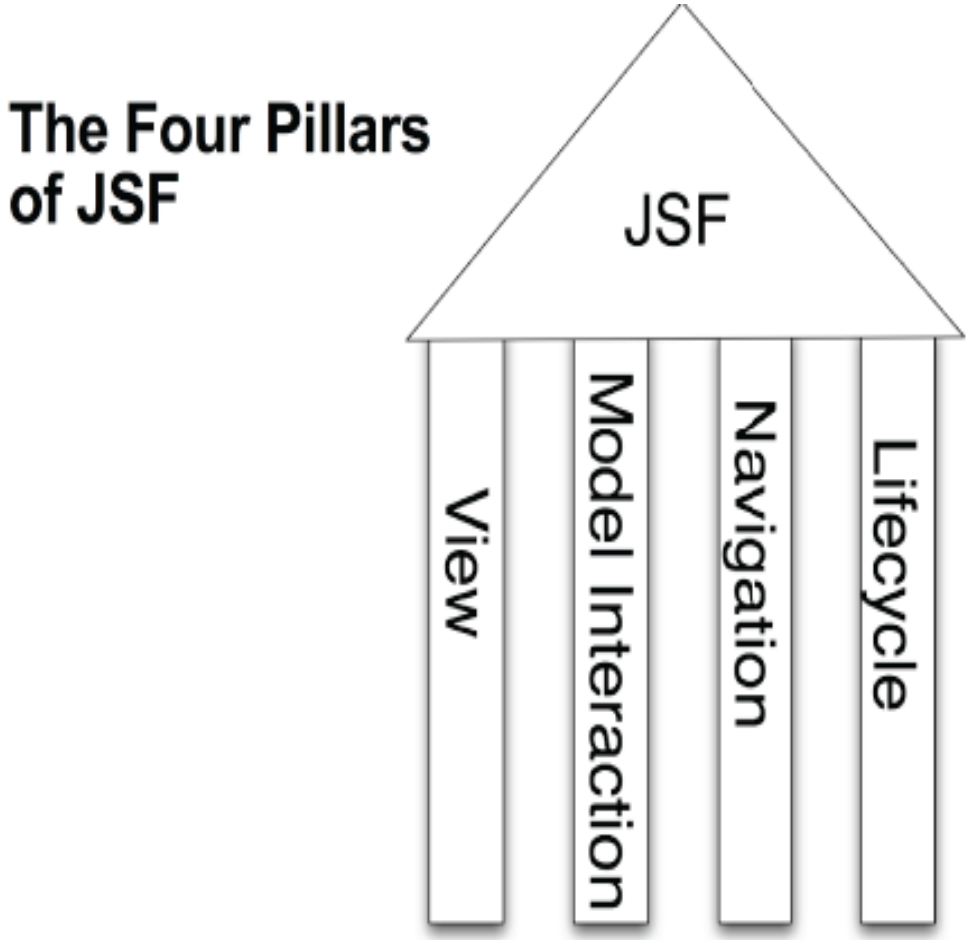
JSF


Servlet
Instanciar
init() pode ser executado várias vezes
init()
service() executado zero ou mais vezes
service() e doXXX()
destroy()
Garbage

JSF

- O ciclo de vida de Servlets não contém nenhum mecanismo adicional ao processamento do serviço requerido.
- Ou seja:
  - Não existem mecanismos para componentes UI orientados a eventos.
  - Não existem mecanismos para manutenção de estado entre múltiplas requisições.
- E JSF?

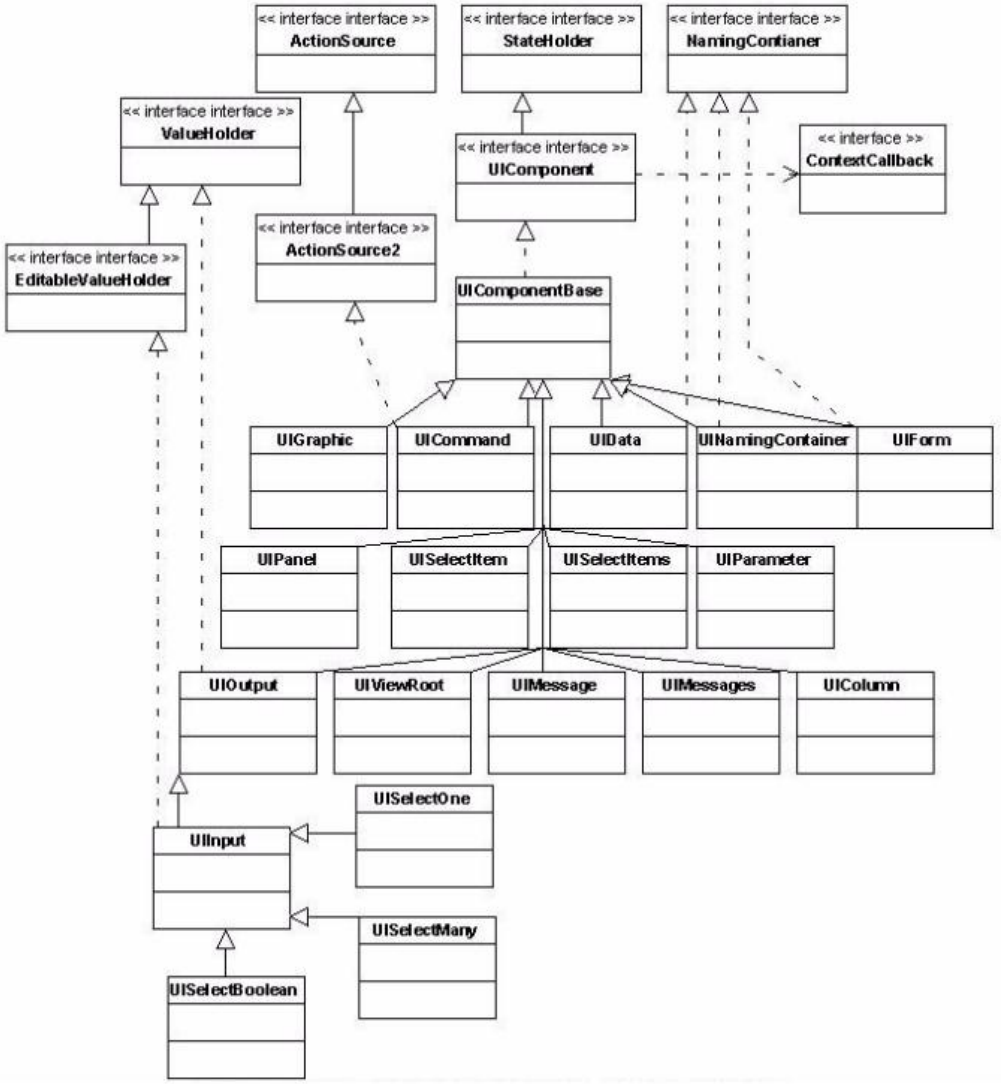
JSF


The Four Pillars of JSF
JSF
View
Model Interaction
Navigation
Lifecycle

JSF

- O foco do JSF é fornecer componentes de interação com o usuário(UI).
- Os componentes são responsáveis por gerar o código HTML através de seus renderizadores.
- JSF já provê 26 UI componentes prontos para o uso.
  - Estes componentes provêem widgets básicos para interação com o usuário como:
    - Input, output, comandos (botões e links), labels, layout, tabelas simples e etc

JSF


<< interface interface >>
ActionSource
<< interface interface >>
StateHolder
<< interface interface >>
NamingContianer
<< interface interface >>
ValueHolder
<< interface interface >>
UIComponent
<< interface >>
ContextCallback
<< interface interface >>
ActionSource2
<< interface interface >>
EditableValueHolder
UIComponentBase
UIGraphic
UICommand
UIData
UINamingContainer
UIForm
UIPanel
UISelectItem
UISelectItems
UIParameter
UIOutput
UIViewRoot
UIMessage
UIMessages
UIColumn
UISelectOne
UIInput
UISelectMany
UISelectBoolean

JSF

- Todos os componentes UI JSF são subclasses de *UIComponentBase*.
  - *UIComponentBase*: responsável por definir os mecanismos padrões de comportamento e estado de um componente UI JSF.

JSF

- *UIColumn*: representa uma única coluna de dados de um componente *UIData*.
- *UICommand*: representa um comando responsável por disparar ações (*actions, actionListeners*).
- *UIData*: representa uma coleção (*collection*) de dados encapsulados por um *DataModel*.

JSF

- *UIForm:* encapsula um conjunto de mecanismos para submissão de dados para a aplicação.
- *UIGraphic*: representa uma imagem.
- *UIInput*: representa um dado de entrada de usuário.

JSF

- *UIMessage*: representa uma mensagem a ser exibida ao usuário.
- *UIMessages*: representa um conjunto de mensagens a serem exibidas ao usuário.
- *UIOutput*: representa um dado de saída em uma página.

JSF

- *UIPanel*: representa um painel agrupador de componentes UI.
- *UIParameter*: representa parâmetros de substituição.
- *UISelectBoolean*: subclasse de *UIInput*, representa a seleção booleana de um campo através de seleção única.

JSF

- *UISelectItem*: representa um único elemento dentre um conjunto de elementos selecionáveis.
- *UISelectItems*: representa um conjunto de elementos selecionáveis.
- *UISelectMany*: subclasse de *UIInput*, permite ao usuário a seleção de múltiplos elementos de um grupo de elementos.

JSF

- *UISelectOne*: subclasse de *UIInput*, permite ao usuário a seleção de um único elemento de um grupo de elementos.
- *UIViewRoot*: representa a raiz da árvore de componentes UI de uma página em particular.

JSF

- E ainda, componentes UI JSF implementam interfaces comportamentais responsáveis por definir certos aspectos de comportamento.

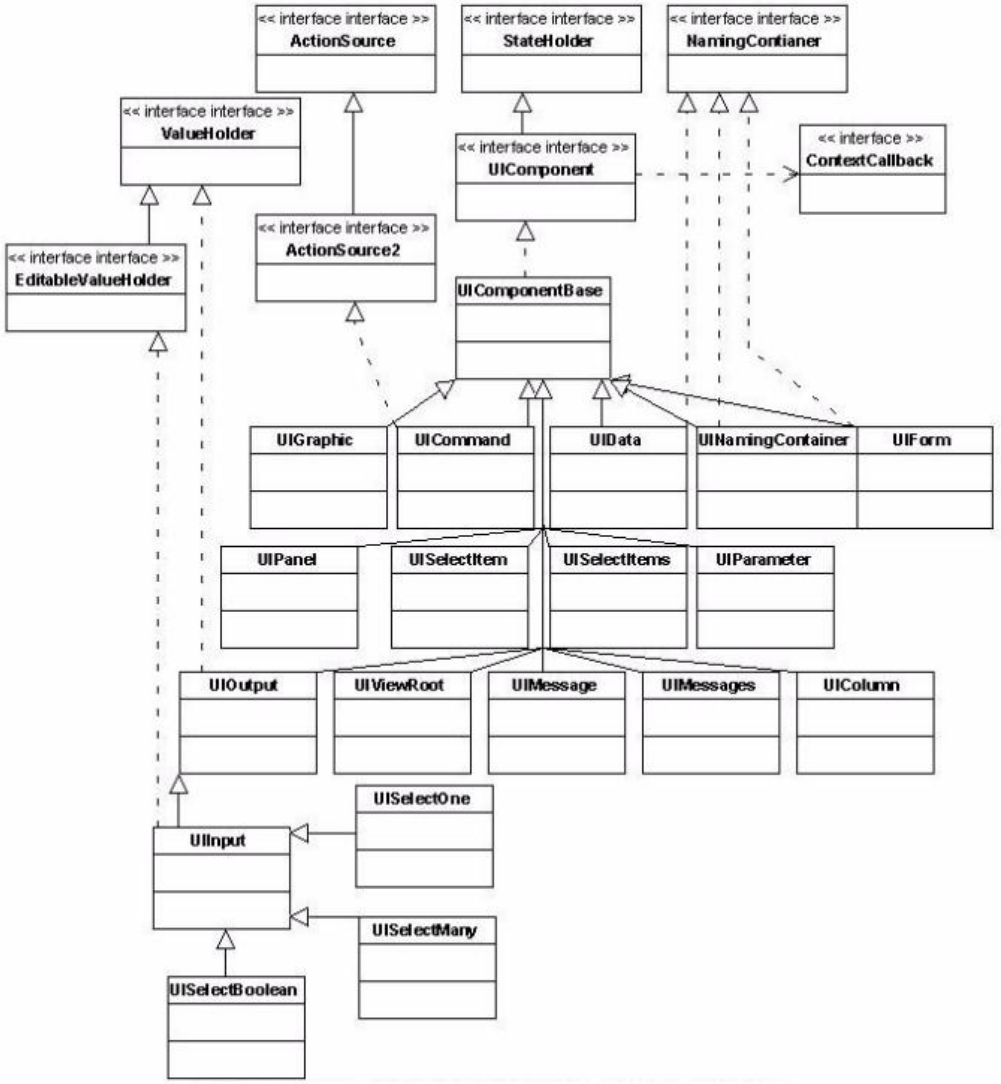
JSF

- *ActionSource*: indica que um componente pode disparar ações (*actions, actionListeners*).
- *ValueHolder*: indica que um componente contém um valor associado.
- *StateHolder*: indica que um componente contém um estado que deve ser mantido entre múltiplas requisições.

JSF

- *NamingContainer*: estabelece que cada componente deve possuir um identificador único (*id*).
- *EditableValueHolder*: subclasse de *ValueHolder*, especifica mecanismos adicionais para componentes editáveis, como validação e orientação a eventos.

JSF


<< interface interface >>
ActionSource
<< interface interface >>
StateHolder
<< interface interface >>
NamingContianer
<< interface interface >>
ValueHolder
<< interface interface >>
UIComponent
<< interface >>
ContextCallback
<< interface interface >>
ActionSource2
<< interface interface >>
EditableValueHolder
UIComponentBase
UIGraphic
UICommand
UIData
UINamingContainer
UIForm
UIPanel
UISelectItem
UISelectItems
UIParameter
UIOutput
UIViewRoot
UIMessage
UIMessages
UIColumn
UISelectOne
UIInput
UISelectMany
UISelectBoolean

JSF

- **Todas** aplicações JSF são construídas através de componentes que podem ser desde simples campos de entrada de dados até um sofisticado painel com uma árvore ordenada.
- Um exemplo de componente de entrada de dados:

```
<h:inputText id="name"
        value="#{helloWorldBean.name}">
```

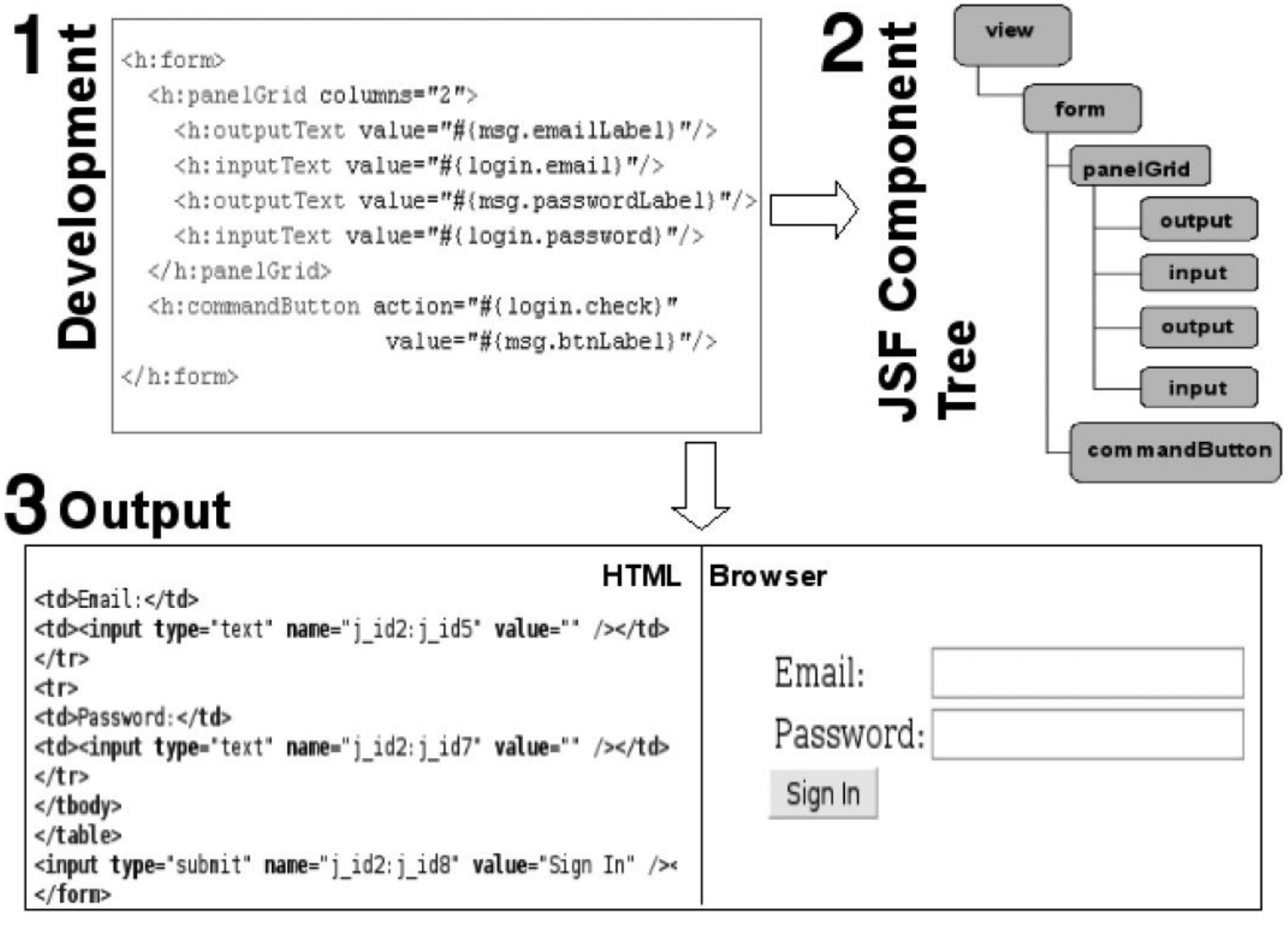
Renderização de componentes

- JSF permite a separação de um componente de como ele é apresentado(encoding) e de como ele a entrada é processada (decoding).
- A aparência pode variar, de acordo com o dispositivo em que ele é acessado.
- A renderização pode ser realizada pelo próprio componente ou delegada a algum renderizador.
- Um componente pode ter vários renderizadores.

Renderização de componentes

- Por exemplo:
  - Um componente pode ter rendezadores que produzem a saída dos dados em diferentes formatos:
    - HTML;
    - XML;
    - WML;
    - Etc.
- Componentes padrão do JSF vêm com renderizadores HTML.

Renderização de componentes



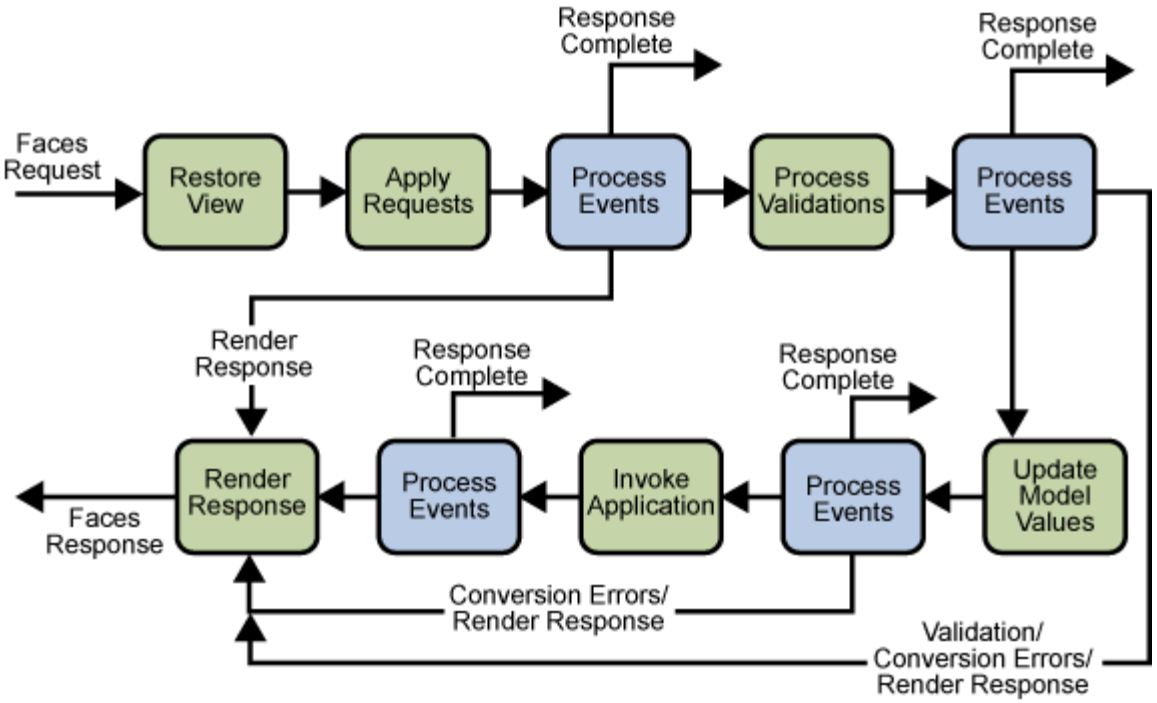
Renderização de componentes

1. Uma página com código de tags JSF. Parecido com uma página JSP. Mas as semelhanças ficam somente aí!Quando a página é processada essas tags exibem a segunda parte da figura.
2. Está é a árvore de componentes do JSF. Esta árvore acompanha o ciclo de vida do JSF. Ao final do ciclo de vida o JSF irá solicitar que cada componente seja renderizado.
3. Este é o código gerado pelas tags JSF exibidos pelo navegador.

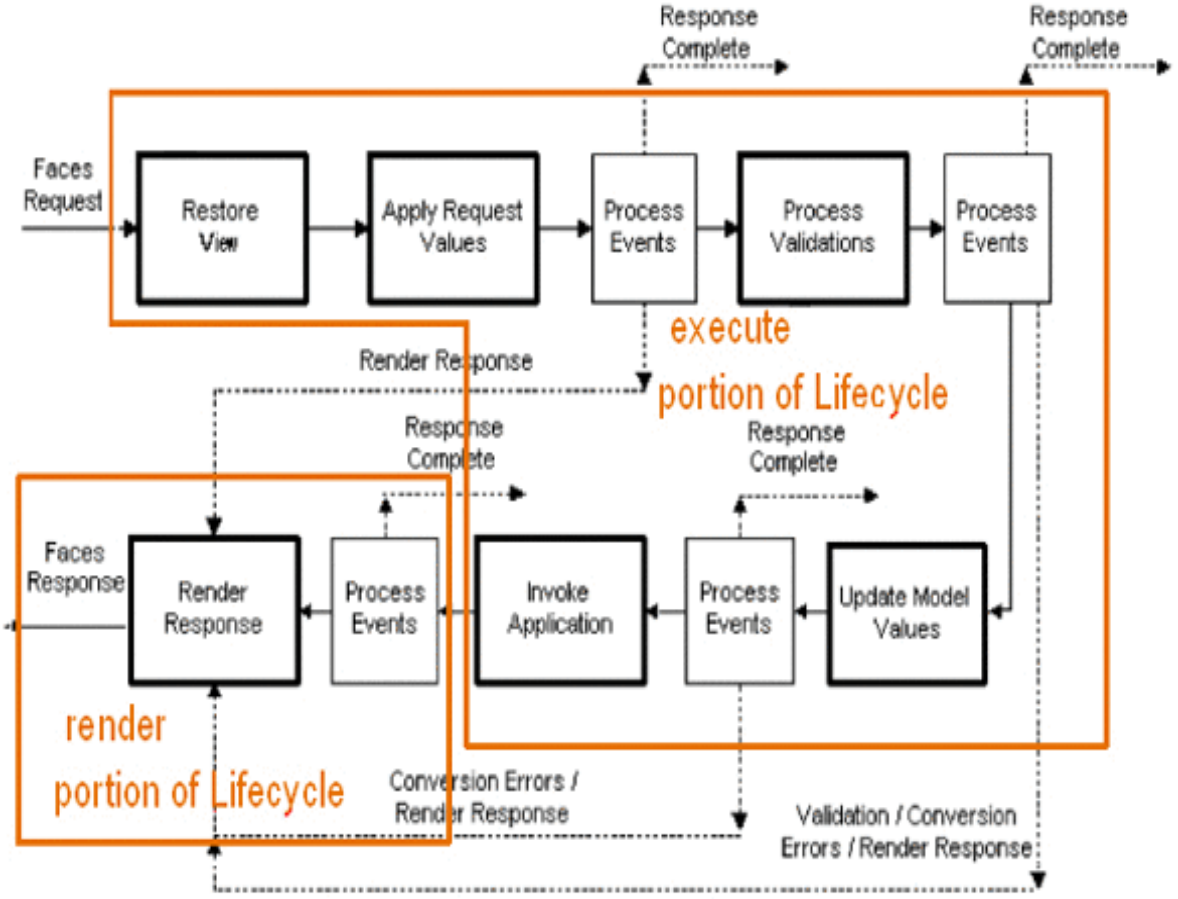
Eventos

- JSF vai além do paradigma de requisição/resposta e provê um poderoso mecanismo baseado em eventos.
- Os componentes de UI enviam eventos quando ativados que são capturados por ouvintes.
- Os ouvintes por sua vez são responsáveis por processar esses eventos.
- Por exemplo: Ao clicar em um botão um evento é disparado e capturado por algum ouvinte.

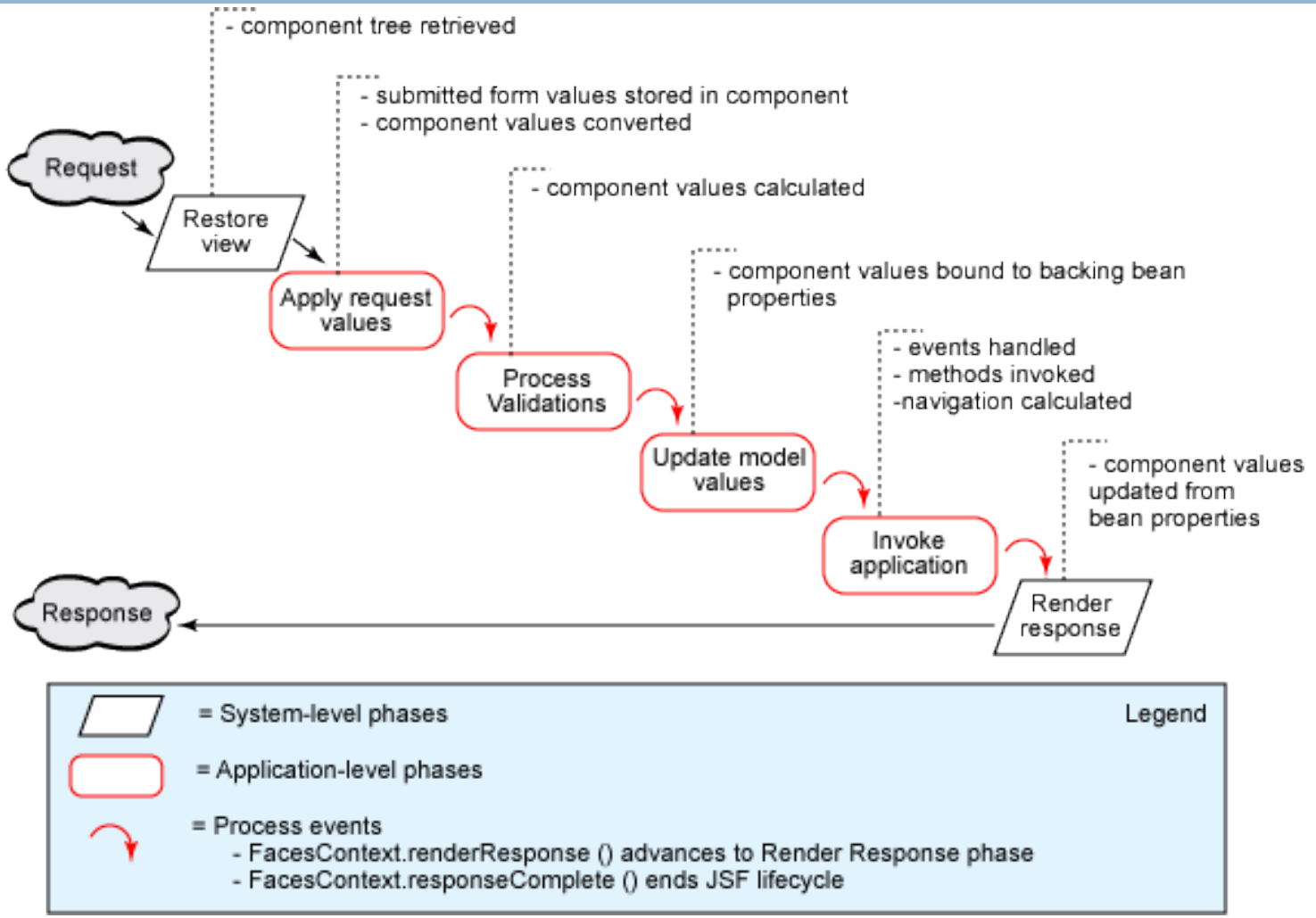
JSF Lifecycle



JSF Lifecycle



JSF Lifecycle



JSF Lifecycle

- *Restore view*:
  - *Decoding phase*.
  - Uma visão (*view tree*) representa o conjunto de **componentes UI JSF** que constituem uma página em particular.

JSF Lifecycle

- *Restore view*:
  - Estratégias de persistência de uma visão:
    - *Client-side*: através de campos ocultos de formulário.
    - *Server-side*: através da sessão HTTP.
  - Cada **componente UI JSF** de uma visão deve possuir um identificador único (*id*).

JSF Lifecycle

- *Apply Request Values*:
  - Cada componente *UIInput* possui um valor associado atribuído pelo usuário.
  - Para cada componente *UIInput* presente na *view tree* corrente o valor atribuído pelo usuário será associado ao respectivo componente *UIInput*.

- **Atenção:** nesta fase não há atribuição de valores aos ***backing beans***, ainda.

JSF Lifecycle

- *Process Validations*:
  - A validação de dados é realizada para cada componente *UIInput*, cada qual acionando suas respectivas rotinas de **conversão** e **validação** de dados.

```
<h:inputText id="helloInput" value="#{helloBean.numControls}"
  required="true">
  <f:validateLongRange minimum="1" maximum="500"/>
</h:inputText>
```

JSF Lifecycle

- *Update **Model** Values*:
  - Neste momento, após todas as atribuições, conversões e validações executadas para cada *UIInput*, ocorre a atribuição dos valores informados aos ***backing beans*** informados.

```
<h:inputText id="helloInput" value="#{helloBean.numControls}" required="true">
...
</h:inputText>
```

JSF Lifecycle

- *Invoke Application*:
  - A partir deste momento eventos associados aos ***backing beans*** são acionados.
  - Exemplos: eventos ***action*** e eventos ***actionListener***.

```
<h:commandButton id="redisplayCommand" type="submit“ value="Redisplay“
    actionListener="#{helloBean.addControls}"/>
```

JSF Lifecycle

- *Render Response*:
  - Envio da resposta ao usuário.
  - *Encoding phase*.

```
<a href id="redisplayCommand" type="submit" value="Redisplay" .../>
```

JSF

- Padrões de desenho em uso pelo JSF:
  - *Decorator: UI customization;*
  - *Singleton: FacesServlet, Lifecycle, ViewHandler, RenderKit;*
  - *Strategy: Flexibel Rendering model;*
  - *Template Method: PhaseListeners;*
  - *Observer: java.util.EventListener.*

JSF

- Por fim, há todo um arcabouço para componentes UI orientados a eventos e com manutenção de estado entre múltiplas requisições.

- Isto é suficiente para aplicações web?
  - **Não.** E a nevagação entre páginas?
    - JSF é simplório neste aspecto. Como resolver?
    - **JBoss Seam! Que iremos ver mais a frente.**

Richfaces

- É uma implementação do JSF.
- Open Source.
- Provê vários componentes JSF com requisições com e sem AJAX.
- Provê recursos para customização do visual da aplicação (look and feel) de maneira simples.

Componentes do Richfaces

- Divides em dois grupos
  - Core AJAX: Componentes que utilizam requisições AJAX de maneira bem simples para o desenvolvedor.
    - Permite que apenas algumas áreas da página sejam atualizadas a cada requisição;
  - UI: Componentes para facilitar a interação do usuário com a aplicação.
    - Suportam os componentes da biblioteca Core AJAX e podem ser customizados facilmente.

Richfaces

- Possui uma biblioteca para o desenvolvimento de novos componentes. A CDK(Component Development Kit) fornece um conjunto de ferramentas para a criação de componentes JSF com AJAX.
- Outras funcionalidades:
  - Suporte ao Facelets ;
  - Possibilidade de criar componentes através de código Java;
  - API de componentes JavaScript para interação no lado do cliente
  - Suporte da comunidade;

Histórico do Richfaces

- Criado a partir do Ajax4Jsf framework.
- Foi criado por Alexander Smirnov que se juntou a Exadel em 2005.
- A idéia era utilizar em conunto os conceitos “bacanas” de AJAX com as técnicas do então novo JSF.
- A primeira versão comercial foi lançada em março de 2006 com o nome de Exadel VCP (Visual Component Plataform)

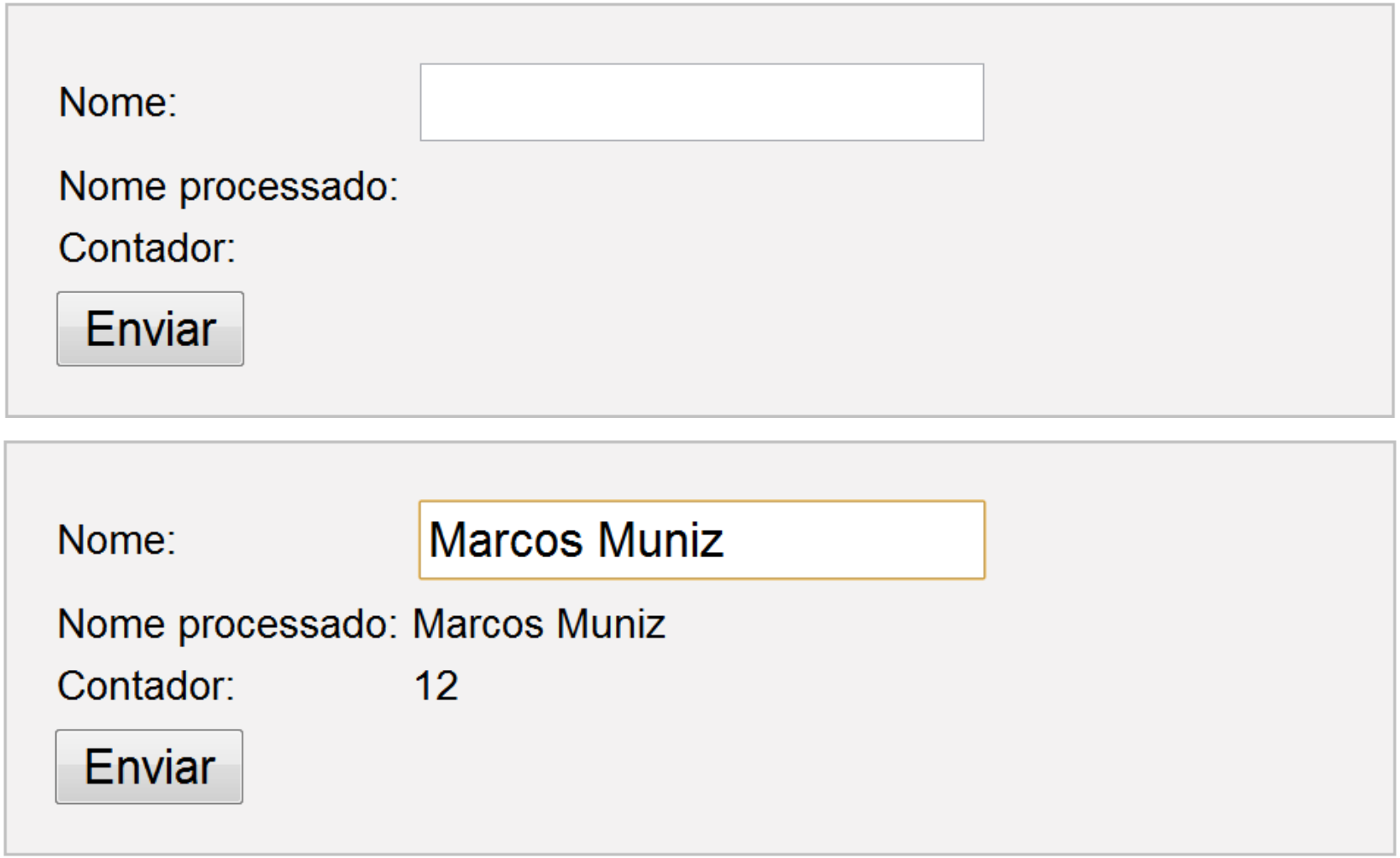
Histórico do Richfaces

- Ainda em 2006 o projeto foi dividido em dois sub-projetos:
  - Ajax4Jsf (open source) ;
  - RichFaces (comercial) .
- Em março de 2007 Exadel e JBoss criaram uma parceria para tornar o código do Richfaces open source e os dois sub-projetos foram unidos e chamado apenas de Richfaces.
- Hoje em dia é o framework mais utilizado para JSF.

Richfaces em prática

- Vamos criar uma página que irá receber o nome do usuário processar a entrada e exibir o nome em outro campo na mesma página. Além disso criaremos um campo para contar o número de caracteres digitados.
- As próximas figuras ilustram o funcionamento da aplicação.

Richfaces em prática



Richfaces em prática

- Passos para executar a prática:
  1. Crie uma página xhtml;
  2. Adicione as bibliotecas JSF na página(caso já não estejam;
  3. Adicione os componentes de entrada e sa'ida de de dados e o botão para Enviar os dados.
  4. Crie um bean para processar os eventos.
  5. Adicione as propriedades(com get e set) no bean.
  6. Crie um método para processar o evento de envio dos dados.
  7. Registre o bean no arquivo faces-config.xml

Dicas para a prática

- Os seguintes componentes podem ser utilizados :
- <rich:panel>, <h:panelGrid>,<h:outputText>, <h:inputText>, <h:outputText >
- O método do seu bean deve receber um ActionEvent como parâmetro.

Adicionando AJAX

- Vamos atualizar os campos Nome processado e contador sem atualizar toda a página.
- Para isto basta:
  - Utilizar componentes da taglib a4j;
  - Definir quais componentes devem ser atualizados após o processamento do evento;

Entendendo o funcionamento

- Vamos criar um simples ouvinte de fase para identificar cada fase.
- Para criar um ouvinte devemos:
  - Criar a classe do ouvinte que implemente a interface javax.faces.event.PhaseListener;
  - Registrar o ouvinte na configuração do JSF(faces-config.xml).
  - Implementar os três métodos da interface (afterPhase, beforePhase e getPhaseId)

Entendendo o funcionamento

```
public class PhaseListener implements javax.faces.event.PhaseListener {
        public void afterPhase(PhaseEvent event) {

        event.getFacesContext().getExternalContext().log("AFTER
        "+event.getPhaseId());
        }
        public void beforePhase(PhaseEvent event) {

        event.getFacesContext().getExternalContext().log("BEFORE
                "+event.getPhaseId());
        }
        public PhaseId getPhaseId() {
                return PhaseId.ANY_PHASE;
        }
}
```

Criando uma validação

- Utilize o exemplo anterior e adicione a tag <f:validateLength> para validar o tamanho da entrada.
- Execute a aplicação e verifique os campos contador e nome processado não serão atualizados até que o usuário preencha o valor mínimo informado na validação.

Phase Listener

Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: BEFORE RESTORE_VIEW 1
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER RESTORE_VIEW 1
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: BEFORE APPLY_REQUEST_VALUES 2
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER APPLY_REQUEST_VALUES 2
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: BEFORE PROCESS_VALIDATIONS 3
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER PROCESS_VALIDATIONS 3
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: BEFORE RENDER_RESPONSE 6
Jun 08, 2012 12:01:23 AM org.apache.catalina.core.ApplicationContext log
INFO: AFTER RENDER_RESPONSE 6

Validação e o ciclo de vida

- Repararam o que houve no ciclo de vida?
- Como a validação não foi atendida o ciclo de vida foi encurtado.
- Da fase de validação fomos direto para a fase de renderização.
- Como seria então se a validação não tivesse terminado o ciclo de vida?

Phase Listener

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE RESTORE_VIEW 1

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER RESTORE_VIEW 1

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE APPLY_REQUEST_VALUES 2

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER APPLY_REQUEST_VALUES 2

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE PROCESS_VALIDATIONS 3

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER PROCESS_VALIDATIONS 3

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE UPDATE_MODEL_VALUES 4

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER UPDATE_MODEL_VALUES 4

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE INVOKE_APPLICATION 5

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER INVOKE_APPLICATION 5

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: BEFORE RENDER_RESPONSE 6

Jun 08, 2012 12:04:21 AM
org.apache.catalina.core.ApplicationContext log

INFO: AFTER RENDER_RESPONSE 6

Componentes Richfaces: AJAX

- Quatro componentes básicos do Richfaces permitem enviar uma requisão AJAX:
  - <a4j:commandLink>
  - <a4j:commandButton>
  - <a4j:support>
  - <a4j:poll>

Componentes Richfaces: AJAX

<a4j:commandLink> and <a4j:commandButton>

- Utilizam o evento onclick padrão do DHTML(Dynamic HTML) para iniciar a requisição.
- Podem ser utilizados os atributos action e actionListener para especificar a ação a ser executada pelo componente.
- Caso a action seja utilizada o método deve retornar null. Como parte da página deve ser atualizada não devemos retornar um valor.

Componentes Richfaces: AJAX

<a4j:commandLink> and <a4j:commandButton>

- Os componentes que devem ser atualizados podem ser especificados através do atributo reRender.
- Vários componentes podem ser atualizados a partir de uma simples requisição. Para indicar quais componentes apenas escreva os ids separados por vírgula.

```
<a4j:commandButton value="Enviar"
reRender="nome, telefone, idade"/>
```

Componentes Richfaces: AJAX

<a4j:support>

- Adiciona funcionalidade AJAX a qualquer componente padrão do JSF.
- Apesar do <a4j:commandLink> e <a4j:commandButton> utilizarem o onclick para realizar a requisição AJAX, o <a4j:support> permite ao desenvolvedor especificar qual evento utilizar.
- <a4j:support> deve ser envolvido por um componente como filho direto de um componente JSF padrão.

Componentes Richfaces: AJAX

<a4j:support>

- Um dos seus principais atributos é o evento DHTML no qual a requisição AJAX será vinculada.

```
<h:inputText id=“nome" value="#{helloBean.nome}">
<a4j:support event="onkeyup" reRender=“nomeProc, contador, nome"
actionListener="#{helloBean.contadorListener}" />
</h:inputText>
```

Componentes Richfaces: AJAX

<a4j:support>

- No exemplo anterior a tag <h:input> deve suportar o evento onKeyUp.
- O número de eventos que pode ser definido é limitado ao número que de eventos que o componente pai suporta.
- Outra maneira de escrever o código anterior é o seguinte:

Componentes Richfaces: AJAX

<a4j:support>

<h:inputText id=“nome" value="#{helloBean.name}"
onkeyup=“alguma_funcao_js()">
...
</h:inputText>

Componentes Richfaces: AJAX

<a4j:support>

- O código HTML gerado é semelhante ao abaixo:

<input id="form:nomeInput" type="text" name="form:nomeInput"
onkeyup="A4J.Ajax.Submit('_viewRoot','form',event,{'parameters'
{'form:j_id4':'form:j_id4'} ,'actionUrl':'/rc
tabs/a4j/support.jsf;jsessionid=EC4E5A2309EA3008E80BCD4957A35EF5'})"
/></td>

Componentes Richfaces: AJAX

<a4j:support>

- O código a seguir não funciona.

```
<h:inputText value="#{userBean.name}"
onkeyup="alert('up')">
<a4j:support event="onkeyup" reRender="echo"/>
</h:inputText>
```

- O literal onKeyUp tem precedência na chamada.

Componentes Richfaces: AJAX

<a4j:support>

- Implemente o código a seguir:

```
<h:panelGrid>
        <h:selectOneRadio value="#{userBean.color}">
                <f:selectItem itemLabel="Red" itemValue="Red"/>
                <f:selectItem itemLabel="Blue" itemValue="Blue"/>
                <f:selectItem itemLabel="Green" itemValue="Green"/>
                <f:selectItem itemLabel="Yellow" itemValue="Yellow"/>
                <a4j:support event="onclick" reRender="col"/>
        </h:selectOneRadio>
        <h:outputText id="col" value="Color: #{userBean.color}"/>
</h:panelGrid>
```

Componentes Richfaces: AJAX

<a4j:support>

- Exercício: Utilize o código anterior como base e crie um bean para contar o número de vezes que o usuário selecionou uma cor.
- Em outro bean armazene a cor selecionada.
- Exiba a cor e o número de vezes que o usuário selecionou uma cor a cada requisição do usuário.

Componentes Richfaces: AJAX

<a4j:poll>

- Funciona de maneira semelhante aos demais componentes no envio da requisição AJAX.
- O seu diferencial está que para enviar um evento não precisa de uma interação com o usuário.
- Os eventos são enviados periodicamente.

Componentes Richfaces: AJAX

<a4j:poll>

```
<h:form>
        <h:panelGrid columns="2">
                <h:panelGrid columns="2">
                        <a4j:commandButton value="Start Clock"
                        action="#{clockBean.startClock}"
                        reRender="poll"/>
                        <a4j:commandButton value="Stop Clock"
                        action="#{clockBean.stopClock}"
                        reRender="poll"/>
                </h:panelGrid>
                <h:outputText id="clock" value="#{clockBean.now}" />
        </h:panelGrid>
</h:form>
```

Componentes Richfaces: AJAX

<a4j:poll>

- Implemente o clockBean e inicie o relógio quando o usuário acionar o comando de início e pare o relógio quando o usuário acionar o comando de parar.

Componentes Richfaces: AJAX

limitToList

- Este atributo existe em todos componentes de ação(componentes que iniciam ações AJAX).
- Mudar o valor deste atributo para **true** limita as atualizações somente para os elementos indicados no atributo reRender.
- Alguns componentes podem ser atualizados em requisições mesmo que não estejam no reRender, o limitToList evita este comportamento que muitas vezes é indesejado.

Componentes Richfaces: AJAX

limitToList

```
<h:form>
   <h:panelGrid columns="2" border="1">
      <a4j:commandButton value="Update #1" />
      <a4j:commandButton value="Update #2" limitToList="true"
         reRender="now2" />
      <a4j:outputPanel ajaxRendered="true">
         <h:outputText id="now1" value="#{dateBean.now1}" />
      </a4j:outputPanel>
      <h:outputText id="now2" value="#{dateBean.now2}" />
   </h:panelGrid>
</h:form>
```

Componentes Richfaces: AJAX

limitToList

- Implemente o bean dateBean com dois atributos now1 e now2 que devem retornar a data atual.
- Verifique o comportamento do limitToList.

Componentes Richfaces: AJAX

Entendendo qual dado processar

- Em uma aplicação sem AJAX o formulário é submetido e todos os campos dentro do formulário são processados durante a fase de aplicar os valores(Apply Request Values).
- Quando a requisição é via AJAX o formulário é submetido por completo porém é possível especificar quais controles devem ser processados.

Componentes Richfaces: AJAX

<a4j:region>

- Permite especificar quais componentes serão processados (decoding, conversão, validação, atualização do modelo) no servidor.

```
<h:form>
    <h:panelGrid>
    </h:panelGrid>
    <a4j:region>
        <h:panelGrid>
        . . .
        </h:panelGrid>
    <a4j:commandLink>
    </a4j:region>
    . . .
</h:form>
```

Componentes Richfaces: AJAX

<a4j:region>

- Sem o <a4j:region> todo o formulário será submetido e processado.
- Muitos desenvolvedores confundem o componente devido ao nome!
- O componente não define quais componentes serão atualizados, mas sim quais componentes serão processados.

Componentes Richfaces: AJAX

ajaxSingle

- Este atributo funciona semelhante ao <a4j:region>.

```
<h:inputText value="#{profile.age}">
    <a4j:support event="onblur" reRender="userInfo"
    ajaxSingle="true">
</h:inputText>
```

- É equivalente a:

```
<a4j:region>
    <h:inputText value="#{profile.age}">
        <a4j:support event="onblur" reRender="userInfo">
    </h:inputText>
</a4j:region>
```

Componentes Richfaces: AJAX

process

- Este atributo deve ser utilizado em situações onde o <a4j:region> ou o atributo process são utilizados mas outro elemento fora da região deva ser processado no envio da requisição AJAX.

Componentes Richfaces: AJAX

process

```
<h:inputText id="fsn" value="#{userBean.frequentShopperNumber}"
required="true">
    <a4j:support event="onkeyup" reRender="outtext"
    ajaxSingle="true"
    process="code"/>
</h:inputText>
<h:inputText id="state" value="#{userBean.state}" required="true">
    <a4j:support event="onkeyup" reRender="outtext"
    ajaxSingle="true"
    process="code"/>
</h:inputText>
<h:inputText id="code" ajaxSingle="true"
value="#{userBean.promotionalCode}"/>
```

Componentes Richfaces: AJAX

process

- No exemplo anterior os dois primeiros campos de entrada tem o atributo ajaxSingle="true" que significa que apenas aquele componente será processado.
- A não ser que todos os componentes sejam colocados em uma única região não é possível processar o campo code.
- Utilizando o atributo process é possível definir componentes fora da região/ajaxSingle que serão processados na requisição.

Componentes Richfaces: AJAX

process

- Este atributo pode ser vinculado aos objetos:
  - java.util.Set;
  - java.util.List;
  - Um array de strings;
  - Uma string (com os ids, separados por vírgula).
- Implemente o bean do exemplo anterior e verifique o funcionamento do atributo process.

Componentes Richfaces: AJAX

Controlando o número de requisições com filas

- Todos componentes AJAX podem degradar a performance de um sistema se forem utilizados de forma inadequada.
- Veja o exemplo a seguir:

```
<h:inputText value="#{exemplo.estado}" >
<a4j:support event="onkeyup" reRender="estado" />
</h:inputText>
```

Componentes Richfaces: AJAX

Controlando o número de requisições com filas

- Suponha que o usuário digite muito rápido. A cada tecla digitada uma requisição será enviada ao servidor.
- Utilizando filas é possível minimizar o tráfico da aplicação fazendo com que os dados das múltiplas requisições sejam processados somente após a execução da requisição em curso.
- Dois atributos auxiliam neste controle :
  - eventsQeueue;
  - requestDelay.

Componentes Richfaces: AJAX

eventsQueue

- Definir eventos em filas é simples:

```
<h:inputText value="#{exemplo.estado}" >
        <a4j:support event="onkeyup" reRender="estado"
        eventsQueue="filaExemplo"/>
</h:inputText>
```

- A próxima requisição adicionada na filaExemplo será processada somente depois que a última requisição AJAX em curso para a fila for processada.
- Por exemplo, se o usuário digitar a primeira letra, uma requisição será disparada. Se ele digitar outras letras enquanto a requisição não for atendida, essas requisições serão enfileiradas e depois que o processamento do primeiro caracter for terminado todas as demais requisições serão processadas como uma só. Ou seja, os caracteres digitados em sequência serão tratados como uma única requisição.

Componentes Richfaces: AJAX

eventsQueue

- Definir eventos em filas é simples:

```
<h:inputText value="#{exemplo.estado}" >
        <a4j:support event="onkeyup" reRender="estado"
        eventsQueue="filaExemplo"/>
</h:inputText>
```

- A próxima requisição adicionada na filaExemplo será processada somente depois que a última requisição AJAX em curso para a fila for processada.
- Por exemplo, se o usuário digitar a primeira letra, uma requisição será disparada. Se ele digitar outras letras enquanto a requisição não for atendida, essas requisições serão enfileiradas e depois que o processamento do primeiro caracter for terminado todas as demais requisições serão processadas como uma só. Ou seja, os caracteres digitados em sequência serão tratados como uma única requisição.

Componentes Richfaces: AJAX

eventsQueue

- Utilizar filas é uma maneira de evitar ataques simples de negação de serviço(DoS) para a sua aplicação ☺ .

Componentes Richfaces: AJAX

requestDelay

- Outro atributo para impedir um grande número de requisições em sequência é o requestDelay.

```
<h:inputText value="#{exemplo.estado}">
    <a4j:support event="onkeyup"
    eventsQueue="filaExemplo" requestDelay="3000"
    reRender="estado" />
</h:inputText>
```

- No exemplo anterior cada caracter digitado será armazenado em uma fila por 3 segundos (3000 milisegundos).

Componentes Richfaces: AJAX

requestDelay

- Em conjunto com o requestDelay está o atributo eventsQueue.
- No exemplo anterior uma requisição seria processada após 3 segundos e se a requisição anterior já tivesse sido executada. Senão ela ficaria aguarndando. Uma nova requisição teria de esperar novamente os 3 segundos.
- Continuando no mesmo exemplo. Suponha que a última requisição tenha sido executada em 5 segundos. Se uma nova requisição for realizada a mesma será executada imediatamente, a próxima poderá ser executada somente em 3 segundos após o término desta.

Componentes Richfaces: AJAX

ignoreDupResponses

- Se alterado para true, este atributo irá ignorar(não processar) a execução corrente de uma requisição AJAX e executar a nova requisição.
- A requisição no servidor da requisição anterior é executada, apenas a renderização da resposta que é descartada.
- Se ignoreDupResponses for utilizado e eventsQueue não for informado uma fila será criada por padrão com o id do componente.
- O funcionamento do ignoreDupResponses é semelhante ao eventsQueue só que ao invés de não processar a requisição a resposta que não é processada.