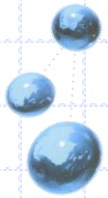


CC68J APLICACIONES EMPRESARIALES CON JEE

JSF

Implementación del patrón MVC

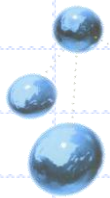
Profesores:

- Andrés Farías

v1.0

Objetivos: aprender a...

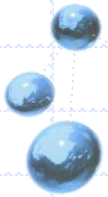
1. Revisitar el patrón arquitectural MVC.
2. Introducción a JSF.
3. Conceptos centrales de JSF.
4. Ejemplo de aplicación JSF.



Revisitando el patrón MVC

Entendiendo MVC y JSF

Introducción a JSF



Java Server Faces
Introducción

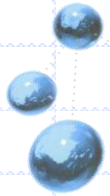
- La tecnología *Java Server Faces (JSF)* es un marco de trabajo (framework) de interfaces de usuario del lado de servidor para aplicaciones Web basadas en tecnología Java.
- Los dos componentes principales son:
  - ✓ Una librería de etiquetas para JSP.
  - ✓ Una API para manejo de eventos, validadores, etc.

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Java Server Faces

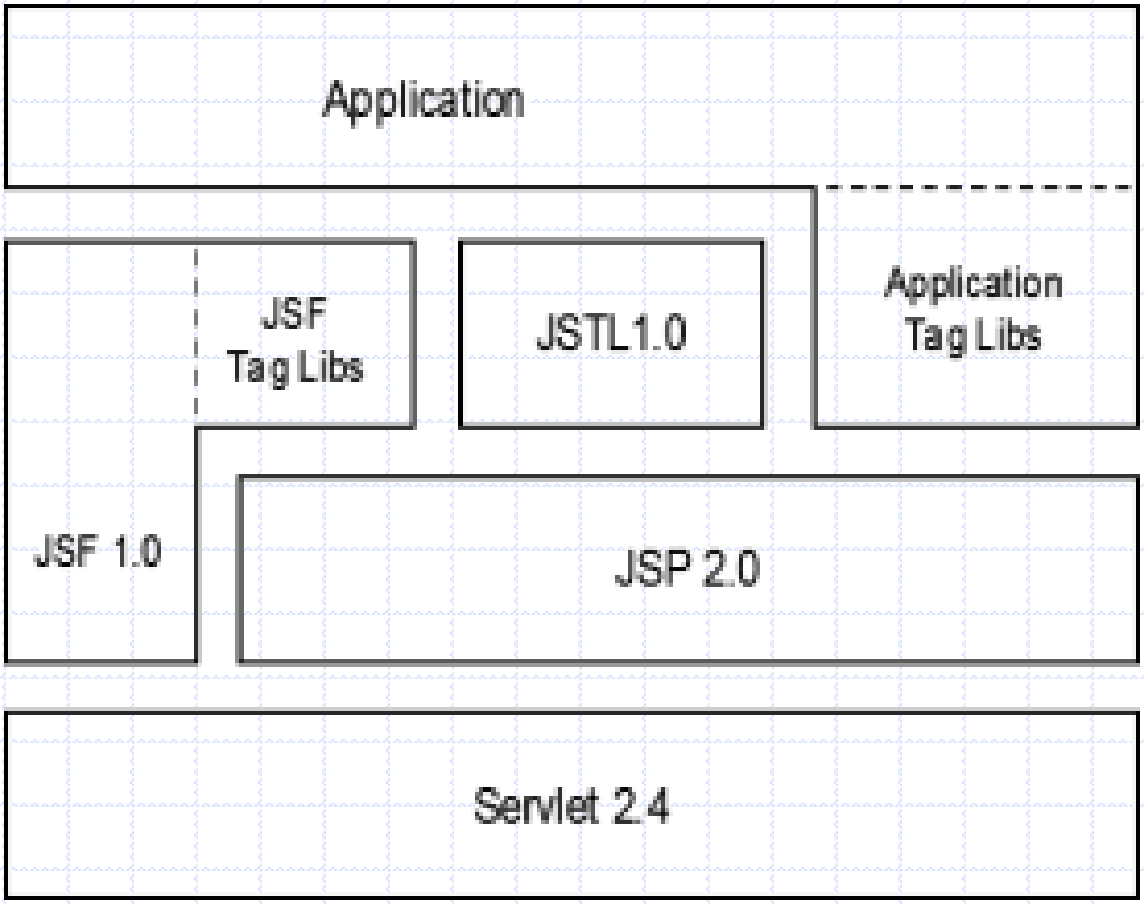
Introducción

- Permite a los desarrolladores pensar en términos de componentes, eventos, backing beans y otras interacciones, en vez de hablar de peticiones, respuestas y marcas.
- JSF promete reutilización, separación de roles, facilidad de uso de las herramientas.
- JSF tiene una meta específica: hacer el desarrollo web más rápido y fácil.



¿En qué tecnologías se basa?

Introducción

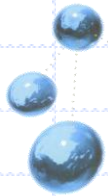


Conceptos centrales de JSF

Los conceptos clave

Resumen

1. Componentes de interfaz de usuario
2. Eventos
3. Beans manejados
4. Validadores
5. Internacionalización y localización
6. Conversores
7. Navegación

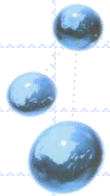


1. Componentes de UI

General

- Los componentes de la interfaz de usuario
  - ✓ Son JavaBeans
  - ✓ Se ejecutan en el lado del servidor
  - ✓ Tienen estado
  - ✓ Se organizan en árboles de vistas
  - ✓ Representación específica: *renderer*
  - ✓ Familia de representaciones: *kits de renderer*

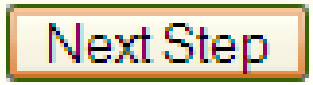
```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```



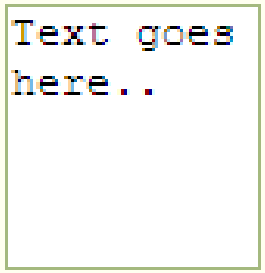
1. Componentes de UI

Ejemplos

```
<h:commandButton id="siguiente" value="#{msg.buttonHeader}"
    action="sigPagina"/>
```



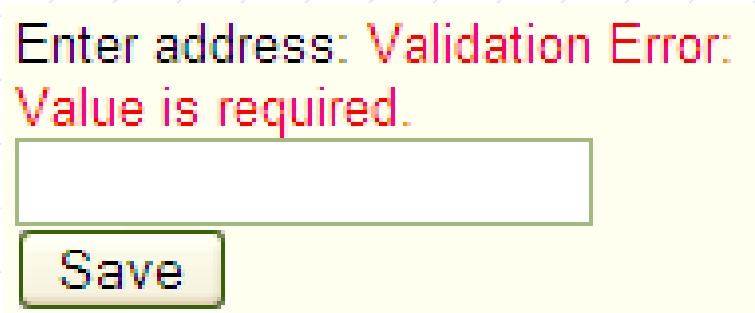

```
<h:inputTextarea id="textArea" rows="4" cols="7" value="texto"/>
```



1. Componentes de UI

De JSF a HTML (1/2)

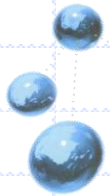
```
Enter address: <h:message style="color: red" for="useraddress" />
<h:inputText id="useraddress" value="#{jsfexample.address}"
        required="true"/>
<h:commandButton action="save" value="Save"/>
```



1. Componentes de UI

De JSF a HTML (2/2)

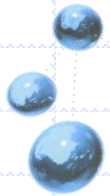
```
Enter address: <span style="color: red">
Validation Error: Value is required. </span>
<input id="jsftags:useraddress" type="text"
name="jsftags:useraddress" value="" />
<input type="submit" name="jsftags:_id1" value="Save" />
```



2. Eventos

General

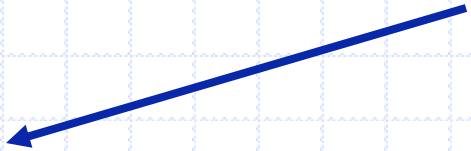
- Los componentes UI generan eventos.
- Los listeners se implementan en backing beans o clases aparte.
- 4 tipos de eventos:
  - ✓Value-change events
  - ✓Action events
  - ✓Data model events
  - ✓Phase events



2. Eventos

Ejemplo: value-change event

```
<h:inputText
   valueChangeListener="#{myForm.processValueChanged}" />
```

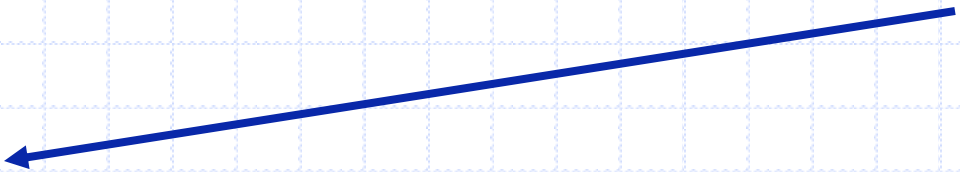


```
public void processValueChanged(ValueChangeEvent event){
   HtmlInputText sender =
        (HtmlInputText)event.getComponent();
   sender.setReadonly(true);
   changePanel.setRendered(true);
}
```

2. Eventos

Ejemplo: action event

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



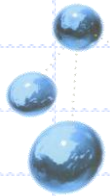
```
public void doIt(ActionEvent event){
   HtmlCommandButton button =
     (HtmlCommandButton)event.getComponent();
   button.setValue("It's done!");
}
```

3. Beans Manejados

Managed Beans – General

- *Beans de respaldo (Backing beans)*
  - ✓ JavaBeans especializados
  - ✓ Contienen datos de componentes UI, implementan métodos de oyentes de eventos
  - ✓ Controlador en el *Modelo Vista Controlador (MVC)*
  - ✓ Backing bean por página, formulario, …
  - ✓ Componente UI y backing bean están **sincronizados**

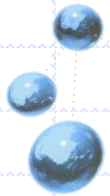
- Son backing beans que usan la facilidad *Manager Bean Creation Facility*



3. Beans Manejados

Ejemplo de declaración (faces-config.xml)

```
<managed-bean>

    <managed-bean-name>helloBean</managed-bean-name>

    <managed-bean-class>
        com.virtua.jsf.sample.hello.HelloBean
    </managed-bean-class>

    <managed-bean-scope>
        session
    </managed-bean-scope>

</managed-bean>
```

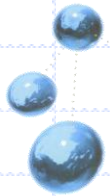


3. Beans Manejados

Ejemplo de uso

```
<h:outputText id="outPut" value="#{helloBean.numControls}" />
```

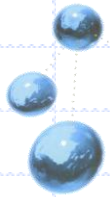
El uso de EL es similar a JSP.



4. Validadores

General

- Aseguran la correcta introducción de valores
- Evitan escribir código Java y/o Javascript
- JSF provee validadores estándar
- Se pueden crear validadores propios
- Generan mensajes de error.
- 3 tipos de validadores:
  - ✓ A nivel de componente UI
  - ✓ Métodos validadores en los backing beans (`validator`)
  - ✓ Clases validadoras (etiqueta propia anidada)

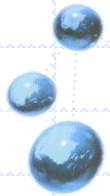


4. Validadores

General

- Estándar de JSF:
  - ✓ campo con valor requerido, validadores de la longitud de una cadena, y validadores de rango para enteros y decimales.
- Ejemplos:

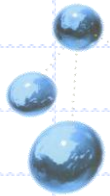
```
<h:inputText id="userNumber"
   value="#{NumberBean.userNumber}" required="true" />

<h:inputText>
   <f:validateLength minimum="2" maximum="10"/>
</h:inputText>
```



5. Internacionalización y localización

General

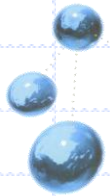
- *Internacionalización:* habilidad de una aplicación de soportar diferentes lenguajes dependiendo de la región del planeta en que nos encontremos.
- *Localización:* El proceso de modificar una aplicación para que soporte la lengua de una región.
- JSF ofrece el soporte, no las traducciones.
- El usuario indica su lengua mediante el navegador.



5. Internacionalización y localización

Ejemplo – declaración en faces-config.xml

```
<application>
   <locale-config>
        <default-locale>en</default-locale>
        <supported-locale>en</supported-locale>
        <supported-locale>en_US</supported-locale>
        <supported-locale>es_ES</supported-locale>
   </locale-config>
   <message-bundle>CustomMessages</message-bundle>
</application>
```



5. Internacionalización y localización

Ejemplo – Resource Boundles y uso

```
<f:loadBundle basename="LocalizationResources" var="bundle"/>
```

LocalizationResources_en.properties

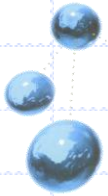
```
halloween=Every day is like Halloween.
numberOfVisits=You have visited us {0} time(s), {1}. Rock on!
toggleLocale=Translate to Spanish
helloImage=../images/hello.gif
```

```
<h:outputText value="#{bundle.halloween}"/>
```

6. Conversores

General

- Convierten el valor de un componente desde y a una cadena.
- Cada componente se asocia a un sólo conversor.
- El renderer lo usa para saber mostrar los datos.
- JSF tiene definidos para fechas, números, etc.
- Podemos crear los nuestros propios.
- Tienen en cuenta la localización y formato.



6. Conversores

General

- Muestran un error si la entrada no es correcta
- Por defecto JSF asigna uno adecuado

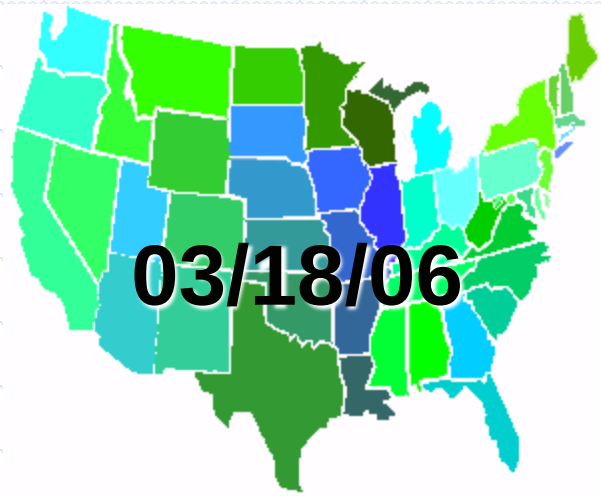
- Se pueden definir de 4 formas:
  - ✓ Etiqueta propia anidada en la del componente
  - ✓ En la etiqueta del componente con **converter**
  - ✓ Etiqueta **<f:converter>** anidada
  - ✓ Etiquetas predefinidas (otras) anidadas

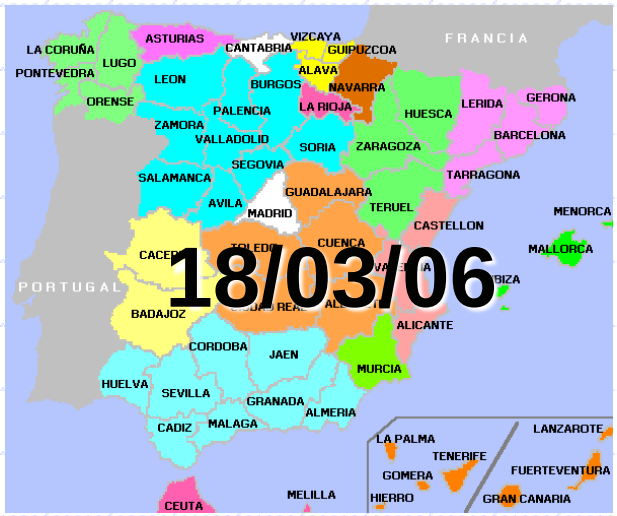
Conversores no predefinidos

6. Conversores

Ejemplo: Conversor predefinido

```
<h:outputText value="#{user.dateOfBirth}">
    <f:convertDateTime type="date" dateStyle="short"/>
</h:outputText>
```

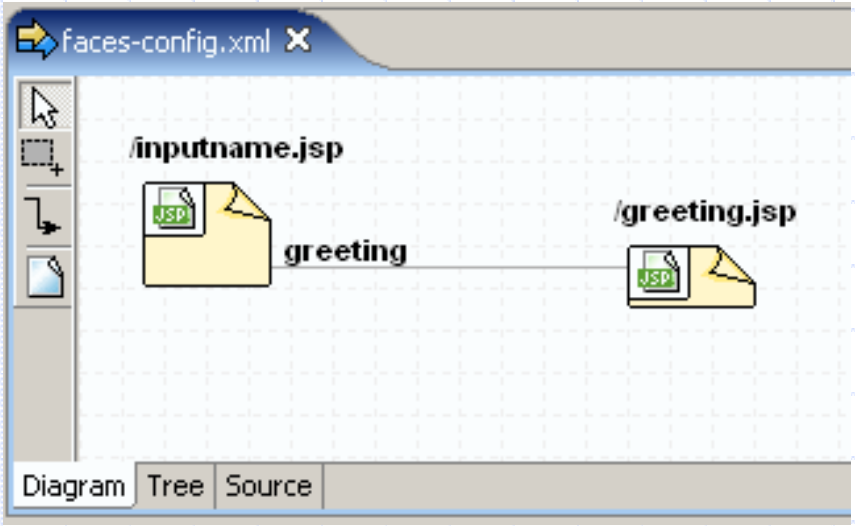




7. Navegación

General

- Habilidad de pasar de una página a la otra
- Lo controla el *manejador de navegación*
- Correspondencia salida/página: *caso de navegación*
- Hay que definir las *reglas de navegación*.



7. Navegación

Ejemplo de declaración (faces-config.xml)

```
<navigation-rule>
   <from-view-id>/login.jsp</from-view-id>
   <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/mainmenu.jsp</to-view-id>
   </navigation-case>
   <navigation-case>
        <from-outcome>failure</from-outcome>
        <to-view-id>/login.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

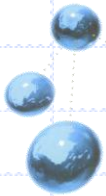
Página origen

acción

Página destino

Desarrollado en BEA Workshop

Un pequeño Ejemplo



Que ofrece workshop?

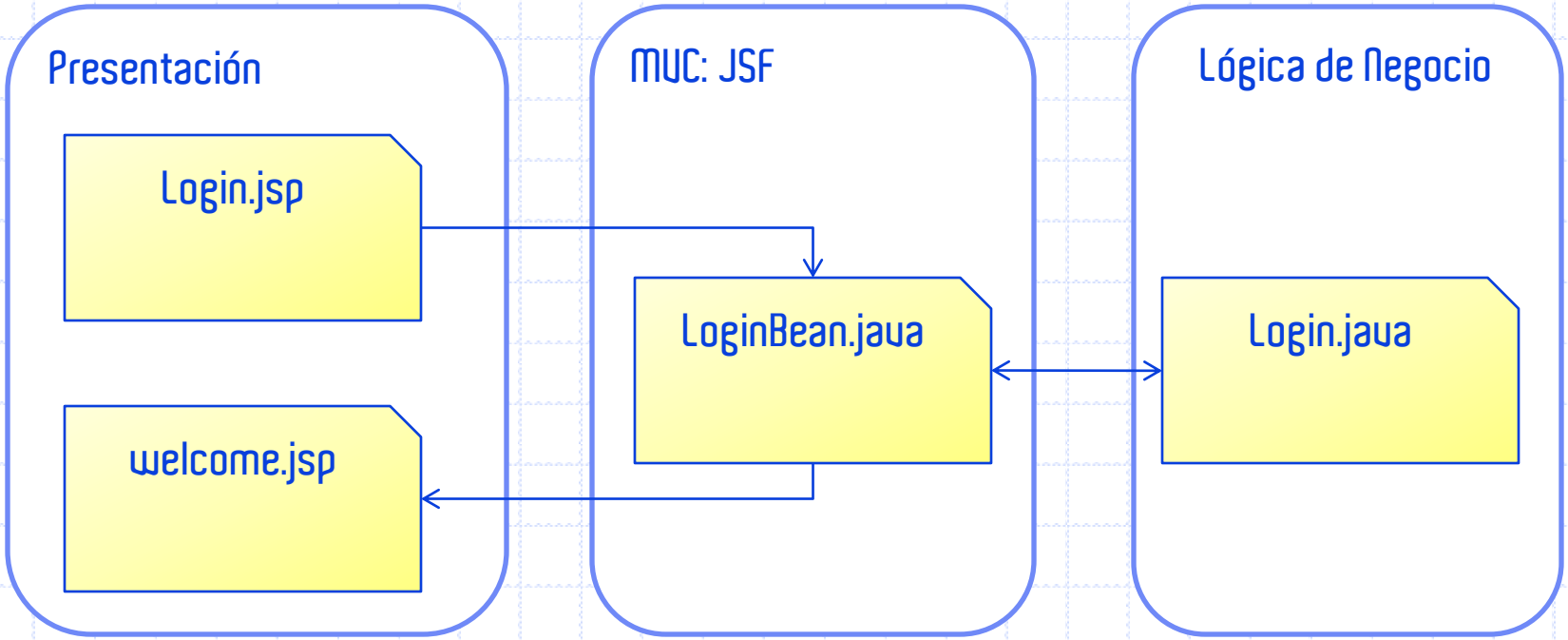
Un ejemplo sencillo

- Provee un IDE para desarrollar aplicaciones web basadas en JSF:
  - ✓ Editor visual (WYSIWYG) con panel de código y diseño.
  - ✓ Sofisticado asistente de código.
  - ✓ Chequeo estático de errores en múltiples niveles.
- AppXRay
  - ✓ Escanea todos los elementos importados por el proyecto web y genera un mapa inteligente de las relaciones entre ellos.
  - ✓ Estos elementos incluyen clases Java, JSP, artefactos de Struts, JSF, archivos de configuración XML y web, resource boundles, variables de la aplicación, propiedades, etc.
  - ✓ Esta aplicación mejora la productivadad para crear, editar o hacer debug de aplicaciones.

Login con WebLogic

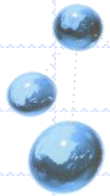
Un ejemplo sencillo

- Se desea construir un módulo de login para una aplicación.
  - ✓ Debe verificar nombres de usuario de largo mayor o igual a 6 caracteres y menor o igual a 10.
  - ✓ La misma validación con el password.



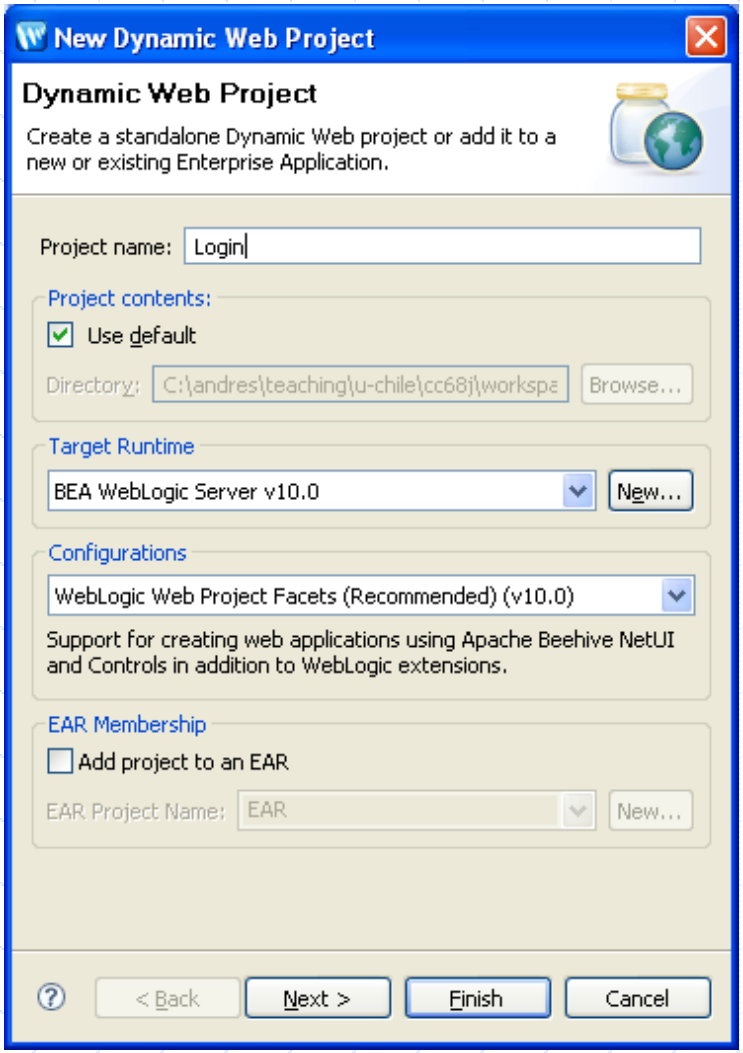
La receta...

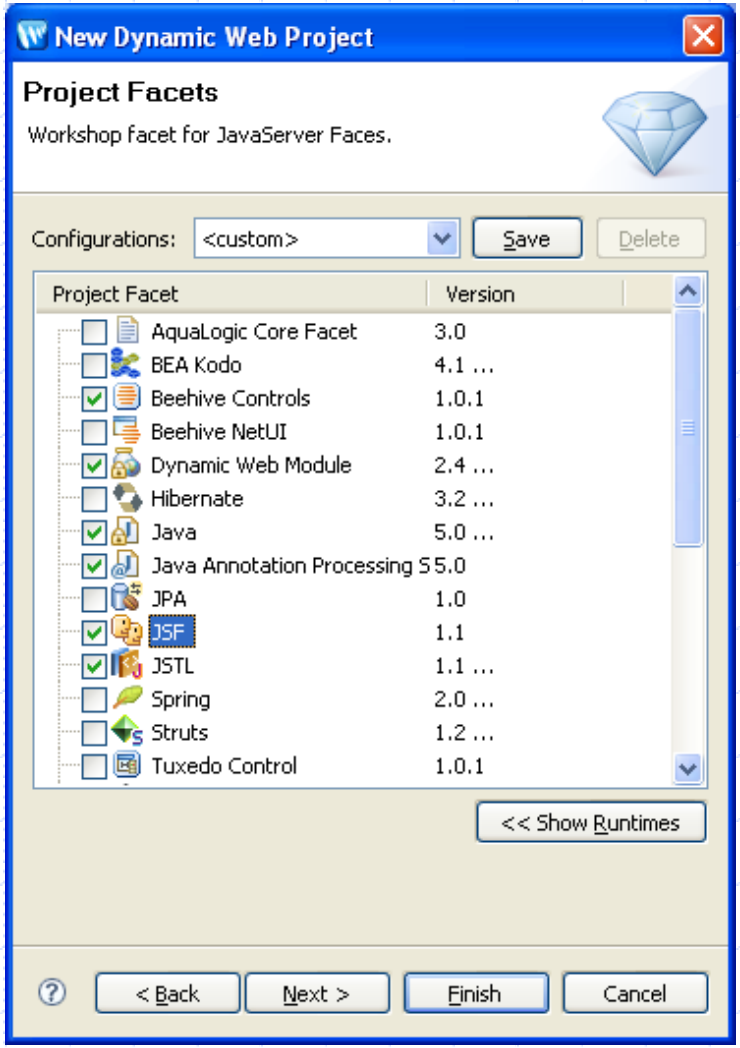
- Crear los beans de negocio:
  - ✓ LoginBean
    - Este bean contiene información del caso de uso: username y password.
    - Además agrega la lógica de navegación en términos abstractos: **success** o **failure**, por ejemplo.
- Crear la vista para el Login:
  - ✓ Login.jsp
    - Provee una vista (view) con los campos de input, error y submit.
  - ✓ Welcome.jsp
    - Contiene un mensaje de bienvenida en caso que el login sea exitoso.
- Definir las reglas de navegación:
  - ✓ Caso de navegación: Para el caso de uso.
  - ✓ Regla de navegación: Para cada posible camino del caso de uso.



Creación del proyecto

Eligiendo las librerías





Creando el bean

LoginBean

```
package cl.uchile.cc68j.login;
public class LoginBean {

    private String username;
    private String password;

    public String login(String username, String password) {
           if (username.equals("afarias") && password.equals("password")) return "sucess";
           else return null;
    }

    public String getPassword() { return password;     }

    public void setPassword(String password) { this.password = password; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }
}
```

Beans administrados

Modificando el faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config>
  <application>
    <message-bundle>resources.application</message-bundle>
    <locale-config><default-locale>en</default-locale></locale-
   config>
  </application>

  <managed-bean>
   <managed-bean-name>login</managed-bean-name>
   <managed-bean-class>cl.uchile.cc68j.login.LoginBean</managed-
   bean-class>
   <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

</faces-config>
```

Creando la vista

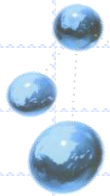
Login.jsp

```
<%@ page language="java" contentType="text/html;
     charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
     Transitional//EN"
     "http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://java.sun.com/jsf/core"
     prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html"
     prefix="h" %>
<f:loadBundle basename="resources.application"
     var="bundle"/>
<f:view>

<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
     charset=ISO-8859-1">
<title>Insert title here</title>
</head>

<body>
<table border="1" cellpadding="0" width="800">
<tr>
<td></td>
</tr>
<tr>
<td>
<h:messages globalOnly="true" layout="table"/>
<h:form id="loginForm">
<h:panelGrid columns="3">
<f:facet name="header">
<h:outputText value="#{bundle['login.title']}"/>
</f:facet>
<h:outputText value="#{bundle['login.user']}"/>
<h:inputText id="username" value="#{login.username}"
     required="true">
<f:validateLength minimum="6" maximum="10"/>
</h:inputText>
<h:message for="username"/>
<h:outputText value="Password:"/>
<h:inputSecret id="password"
     value="#{login.password}" required="true">
<f:validateLength minimum="6" maximum="10"/>
</h:inputSecret>
<h:message for="password"/>
</h:panelGrid>
<h:commandButton action="#{login.login}"
     value="Login"/>
</h:form></td></tr>
</table></body></html>
</f:view>
```

Navegación

Modificando el faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config>
  <application>
    <message-bundle>resources.application</message-bundle>
    <locale-config><default-locale>en</default-locale></locale-
   config>
  </application>
   ...
  <navigation-rule>
   <from-view-id>/pages/Login.jsp</from-view-id>
   <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/pages/welcome.jsp</to-view-id>
   </navigation-case>
  </navigation-rule>
  </faces-config>
```

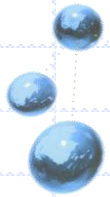


Teoría vs. Práctice

Teoría vs Práctica

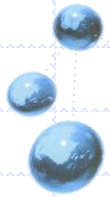
En teoría

- Son aplicaciones web que trabajan más como aplicaciones de escritorio:
  - ✓ Fáciles de programar.
  - ✓ Menos preocupación sobre el ciclo de vida del request.
- Mejor alcance y separación
  - ✓ Application/Session/Request
- Árbol de componentes abstracto, que es independiente del objetivo (HTML, etc.).
- Buena separación de la lógica de negocio y la capa de presentación.



En la práctica

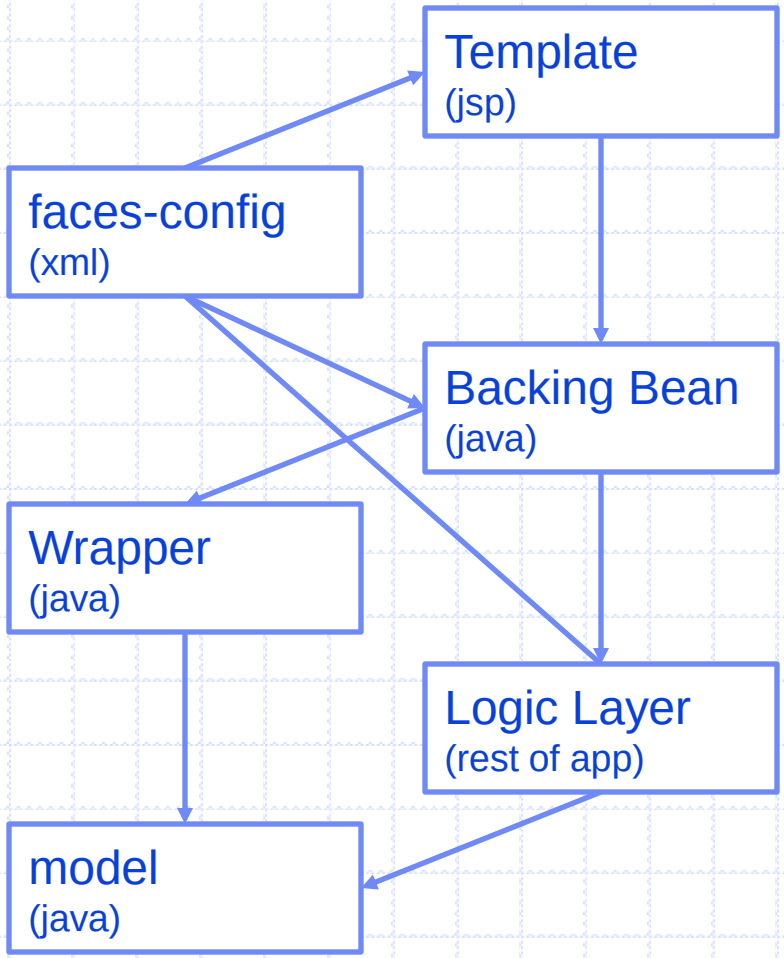
- Más difícil de trabajar con JSF que con otros frameworks
  - ✓ Mayor curva de aprendizaje
  - ✓ Se debe entender bien el ciclu de vida del request
    - Especialmente las maneras en que JSF podría acortarlo.
- Impide acceder a la mitad de las capacidades de HTML.
  - ✓ Abstracción hecha vía custom taglibs que hacen los templates en un "JSFXML".
  - ✓ Lógica en el template y tags propietarios se interponen en el objetivo de separar la lógica de negocio con la UI
- El modelo de eventos de JSF presenta la necesidad de wrappers para interactuar con la capa del modelo:
  - ✓ Se debe mantener el árbol de componentes y el modelo de eventos en sesión.



JSF: Estructura y código

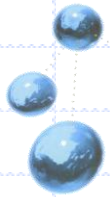
JSF Estructura

- El template (más comunmente conocido como jsp) define la interfaz gráfica.
- El **faces-config** define la navegación y los backing beans
- **Backing beans** manejan el procesamiento de acciones, navegación y conexiones a la capa de negocio.
- **Wrapper** bean encapsula los datos en POJOs para el manejo de JSF.
- **Capa lógica** puede ser inyectada como se define en el faces-config
- **Model** es data basica (POJO)



JSF templates

- Archivos JSP casi siempre.
- Se apoya fuertemente en tag libraries (taglibs)
  - ✓ Core (f) – tags básico de definición de páginas.
  - ✓ Html (h) – define los tags html estándar.
- Se debe aprender una cantidad importante de de taglibs con restricciones
  - ✓ Dificiles de trabajar ellos, yaque no luce como html normal
  - ✓ Es aun más difícil escribir nuevos tags.
- Masivo uso de EL (Expression Language)
- Puede motivar o apoyar la mezcla de código con HTML.

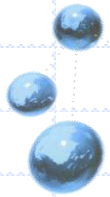


Ejemplo de template

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view>
<html><head><title>Items</title></head><body>
<h:form id="items">
   <h:dataTable id="itemlist" value="#{JsfBean.allItems}"
var="entry">
      <h:column>
         <f:facet name="header">
            <h:outputText value=""/>
         </f:facet>
         <h:selectBooleanCheckbox id="itemSelect"
value="#{entry.selected}" rendered="#{entry.canDelete}"/>
         <h:outputText value="" rendered="#{not entry.canDelete}"/>
      </h:column>
</h:form>
</body></html>
</f:view>
```

faces-config.xml

- Define los backing beans
  - ✓ Syntaxis no como Spring (desafortunadamente)
  - ✓ Nombre usado en EL en el template
  - ✓ Control en Scope (request, session, etc.)
- Define las reglas de navegación
  - ✓ Basado en vistas
    - De donde viene (view)
    - Que resultado (id)
    - Donde ir (view)
  - ✓ Puede hacer match de outcomes (salidas de las páginas) con expresiones regulares.

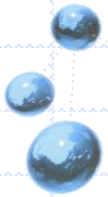


Ejemplo de faces-config

```
<faces-config>
   <navigation-rule>
          <from-view-id>/jsf/JsfItems.jsp</from-view-id>
          <navigation-case>
                    <from-outcome>newItem</from-outcome>
                    <to-view-id>/jsf/JsfAddItem.jsp</to-view-id>
          </navigation-case>
          <navigation-case>
                    <from-outcome>*</from-outcome>
                    <to-view-id>/jsf/JsfItems.jsp</to-view-id>
          </navigation-case>
   </navigation-rule>

   <managed-bean>
        <managed-bean-name>JsfBean</managed-bean-name>
        <managed-bean-class>org.example.jsf.JsfBean</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
          <managed-property>
                    <property-name>logic</property-name>
                    <value>#{someLogicBean}</value>
          </managed-property>
   </managed-bean>
</faces-config>
```

JSF backing beans

- Beans típicos con getters y setters y métodos adicionales para manejar acciones.
  - ✓ Almacenan datos requeridos para procesar las acciones del usuario utilizando setters.
  - ✓ Recupera datos utilizando getters y métodos.
- Frecuentemente incluye código para encapsular (to wrap) objetos de datos.
- Conectan al resto de la aplicación
  - ✓ Típicamente vía inyección
    - No-recursiva (i.e. no como Spring), no instanciará dependencias de dependencias.

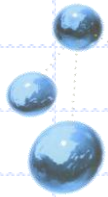


Ejemplo de backing bean

```
public class JsfBean {
            private DataModel itemsModel;
            private JsfItemWrapper currentItem = null;
            ...
            private JsfLogic logic;
            public void setLogic(JsfLogic logic) {
               this.logic = logic;
            }
            ...
            public DataModel getAllItems() {
               List wrappedItems = new ArrayList();
               List items = logic.getAllVisibleItems(logic.getCurrentSiteId());
               for (Iterator iter = items.iterator(); iter.hasNext(); ) {
                       JsfItemWrapper wrapper =
                          new JsfItemWrapper((Item) iter.next());
                          wrappedItems.add(wrapper);
               }
               itemsModel = new ListDataModel(wrappedItems);
               return itemsModel;
            }
            ...
            public String processActionList() {
               return "listItems";
            }
}
```

JSF wrapper bean

- Encapsulan POJO de información básica.
- Requerido para evitar poner información de la UI en los datos del POJO.
- Frecuentemente incluye los setters y getters para las propiedades relacionadas a la UI.
- También incluye un setter y getter para cada atributo del POJO.



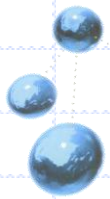
Sample wrapper bean

```
public class JsfItemWrapper {

        private Item item;

         // is this item selected by the user
        private boolean isSelected;

        public JsfItemWrapper(Item item) {
                this.item = item;
        }
        public Item getItem() { return item; }

        public void setItem(Item item) { this.item = item; }

        public boolean isSelected() { return isSelected; }

        public void setSelected(boolean isSelected) {
                this.isSelected = isSelected;
        }
}
```

Web app basics

4 cosas necesarias en un webapp

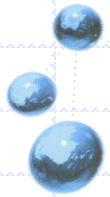
1. Imprimir texto dinámico
   - Dibuja datos en la página.
2. Estructuras iterativas
   - Dibuja colecciones o tablas
3. Render opcional de componentes
   - Dibuja algunos componentes basados en su estado.
4. Gatilla acciones
   - Acciones de usuarios o transferencia de datos.



Imprimir texto dinámico

```
<h:outputText value="#{JsfAppBean.currentItem.title}"/>
<h:outputText value="#{msgs.jsfapp_text}"/>
```

- Utiliza el tag **h:outputText**
  - ✓ También **h:outputLabel** y **h:outputFormat**
- Utiliza Expression Language (EL)
  - ✓ Requiere un bean, definido en el faces-config o el template.
- Puede establecer un estilo y habilitar/desabilitar el escape de caracteres especiales.



Estructuras de Loop

```
<h:dataTable id="itemlist" value="#{JsfAppBean.allItems}" var="entry">
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msgs.jsfapp_text}"/>
    </f:facet>
    <h:outputText value="#{entry.item.title}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msgs.jsfapp_hidden}"/>
    </f:facet>
    <h:selectBooleanCheckbox id="itemHidden" value="#{entry.item.hidden}" disabled="true" />
  </h:column>
</h:dataTable>
```

- **h:dataTable** es la estructura de ciclo principal.
  - ✓ También lo es **h:panelGrid** en cierto nivel.
- Toma una colección como un valor
  - ✓ Usa una variable (entry) para interactuar con la colección.
- Usa **h:column** para definir cada columna.

Rendering opcional

```
<h:outputText value="#{entry.item.title}" rendered="#{not entry.canDelete}"/>

<h:commandLink id="updatelink"
        action="#{JsfAppBean.processActionUpdate}"
        rendered="#{entry.canDelete}">
    <h:outputText value="#{entry.item.title}"/>
</h:commandLink>
```

- Manejado por el tag **h:** con el atributo **rendered** (que toma EL).
  - ✓ Puede ser precedido por **not** para invertir el efecto.
- Permite llevar lógica al template.

Gatillar acciones

```
<h:commandButton value="#{msgs.jsfapp_new}"
    action="#{JsfAppBean.processActionNew}" />
```

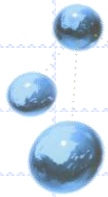
```
public String processActionNew() {
   currentItem = null;
   itemText = TEXT_DEFAULT;
   itemHidden = HIDDEN_DEFAULT;
   return "newItem";
}
```

- Une efectivamente a un *bean action* a un botón submit (**h:commandButton**).
  - ✓ También lo hace **h:commandLink**
- Gatilla una acción en el bean, que retorna un string para indicar la vista (view) a la cual navegar.

JSF EN LA PRÁCTICA

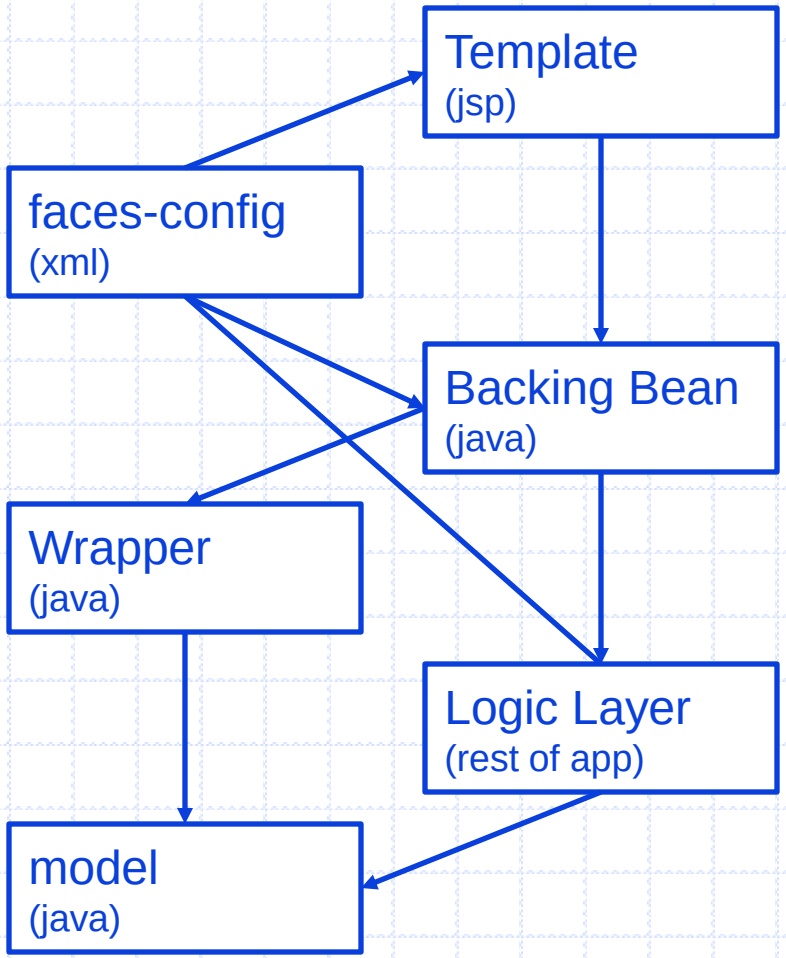
La experiencia con JSF

- JSF tiene una curva de aprendizaje empinada.
  - ✓ Tiende a hacer las cosas de una manera no intuitiva.
- Componentes de UI están restringidos
  - ✓ No hay suficientes, y no son suficientemente flexibles.
- Es difícil combinar con AJAX
  - ✓ Esto podría cambiar al corto plazo.
- Usa mucho Javascript
  - ✓ URL location tiene bugs (función back).
- Difícil para que los diseñadores gráficos trabajen con JSF.



Revisitando la estructura JSF

- El **template** mezcla HTML y lógica de presentación.
- Malo para diseñadores gráficos.
- El **faces-config** es difícil de mantener (la navegación también en los backing files).
  - ✓ Es fácil quedar d esincronizados.
  - ✓ La sintaxis no es como Spring.
- **Backing beans** funcionan bien
  - ✓ Una de las pocas gracias
- **Wrapper** es un trabajo extra con el que hay que lidiar.
  - ✓ Algunas veces hay que encapsular múltiples objetos.
  - ✓ Requiere código extra en el backing bean.



Apache MyFaces

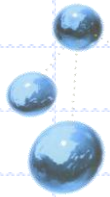
- Es la implementación Apache de JSF
  - ✓ http://myfaces.apache.org/
  - ✓ Open source
  - ✓ Mejor que la implementación de Sun (dicen)
- Más funcionalidades
  - ✓ Taglibs adicionales y más flexibles.
    - Tomahawk (h) – Componentes nuevos y actualizados.
      - http://myfaces.apache.org/tomahawk/

Oracle ADF faces

- Oracle ADF Faces es un conjunto robusto y rico de componentes de UI que trabajan con JSF
  - ✓ Tablas avanzadas
  - ✓ Selector de Fechas y colores.
  - ✓ Subir archivos.
  - ✓ Validadores por el lado cliente.
- Diseñado para trabajar con Oracle Jdeveloper.
- Soporte para AJAX.

Otros faces

- Facelets
  - ✓ https://facelets.dev.java.net/
  - ✓ JSF sin JSP (similar a tapestry)
- Struts Shale
  - ✓ http://shale.apache.org/
  - ✓ Framework basado en JSF, similar a struts



Links

- Java Server Faces Home
  - ✓ http://java.sun.com/javaee/javaserverfaces/
- MyFaces
  - ✓ http://myfaces.apache.org/