



CIFP César Manrique

UT06 Java Server Faces 2 y ORM Hibernate

UT06_01 Introducción a Java Server Faces (JSF)



CONTENIDOS

- Introducción a JSF
- CDI Weld
- Ciclo de vida request-reponse JSF
- Configuración de proyecto Maven, JSF
- Ejemplo básico
- Expression Language (EL)

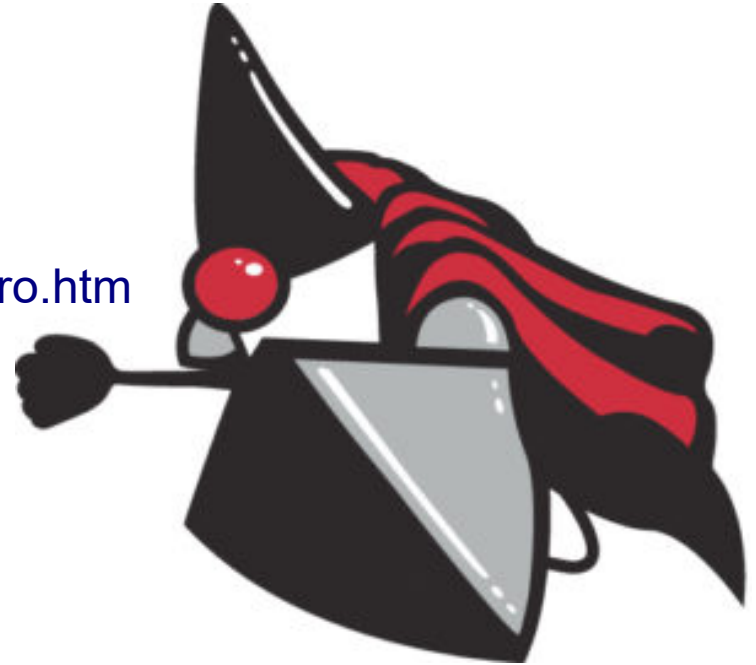




Java Server Faces

JSF 2.2

<https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm>





Tecnologías vistas

- Servlets
 - Java Server Pages (JSP)
 - Java Database Connectivity (JDBC)
 - Java Standard Template Library (JSTL)
 - Expression Language (EL)
 - Java Naming and Directory Interface (JNDI)
 - Java Native Interface (JNI) -no vista, pero no confundir con JNDI-
 - ...
 - Y ahora:
 - **Java Server Faces (JSF) // se merienda a JSP**
- Los veremos mejor ahora



Introducción a Java Server Faces (JSF)

- Java Server Faces (en adelante **JSF**) es un framework de componentes, de lado servidor, para la construcción de aplicaciones Web basadas en Java EE. Es una tecnología propia de la capa de presentación
- **Facelets** es el lenguaje de declaración de vistas y reemplaza a Java Server Pages (JSP)
- **JSF** incluye un nuevo conjunto de *tag libraries* que permiten generar componentes HTML
- Incorpora **managed beans**, java beans gestionados por JSF, que tanto pueden almacenar datos de usuario de la petición, como el modelo de respuesta, como actuar como mini-controladores de la capa de presentación
- El **Expression Language (EL)** es el mecanismo que permite comunicar a la capa de presentación (páginas web) con los managed beans y la lógica de negocio



Facelets

- Permite *templating*; esto es podemos tener:
 - Facelets templates. Páginas XHTML que definen el layout de una página maestra

Facelet template clients

Redefinen zonas del

template

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

```
<body>
```

```
<ui:composition template="./newTemplate.xhtml">
```

```
<ui:define name="content">
```

```
    Defino el contenido de la sección central
```

```
</ui:define>
```

```
</ui:composition>
```

```
</body>
```

```
</html>
```

```
<h:head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <h:outputStylesheet name="./css/default.css"/>
  <h:outputStylesheet name="./css/cssLayout.css"/>
  <title>Facelets Template</title>
</h:head>
```

```
<h:body>
```

```
<div id="top">
```

```
  <ui:insert name="top">Top</ui:insert>
```

```
</div>
```

```
<div id="content" class="center_content">
```

```
  <ui:insert name="content">Content</ui:insert>
```

```
</div>
```

```
<div id="bottom">
```

```
  <ui:insert name="bottom">Bottom</ui:insert>
```

```
</div>
```

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



JSF HTML component tag library

- La tag library html incorpora los componentes más usuales para generar elementos HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/x
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

  <body>

    <ui:composition template="./newTemplate.xhtml">

      <ui:define name="content">


        <h:form>
          <h:outputLabel for="txtMyProperty" value="Propiedad"/>
          <h:inputText id="txtMyProperty" value="#{myBean.myProperty}"/>

          <h:commandButton action="#{myController.action}"/>
        </h:form>

      </ui:define>

    </ui:composition>


  </body>
</html>
```



Managed bean



Managed bean

- Un managed bean es un bean Java cuyo ciclo de vida es gestionado por JSF
- Y... ¿qué es un Java Bean...?

 - Un Java Bean es un estándar, es una clase que sigue las siguientes reglas
 - Todas las propiedades son privadas
 - El acceso a las propiedades debe realizarse con getters/setters
 - Define un constructor público sin argumentos (por defecto)
 - Implementa... **serializable**



¿Qué papel juegan los managed bean?

- En unos textos dicen que son el modelo, pero tienen métodos, ¿son el controlador?
- Leer los siguientes artículos
 - <https://dzone.com/articles/making-distinctions-between>
 - <http://stackoverflow.com/questions/746047/jsf-backing-bean-structure-best-practices>



Papeles que pueden jugar los Managed Bean en la aplicación

Tipo	Apodo	Ámbito (scope)
Model Managed-Bean	model-bean	session

Suelen ser clases POJO que usualmente almacenan entidades de base de datos y forman parte del modelo de datos de la aplicación

Backing Managed-Bean	backing-bean	request
----------------------	--------------	---------

Suelen estar asociados a una vista y almacenan datos relacionados con ella. Por ejemplo pueden respaldar la información suministrada en un formulario

Controller Managed-Bean	controller-bean	request
-------------------------	-----------------	---------

Suelen ser beans con métodos que disparan lógica de negocio y en función de su valor de retorno (el nombre de una vista) se muestra una u otra vista de la aplicación

Support Managed-Bean	support-bean	session / application
----------------------	--------------	-----------------------

Beans que suelen almacenar colecciones de datos que precisan los controles de interfaz de usuario (por ejemplo una lista de ciudades, para un drop-down)

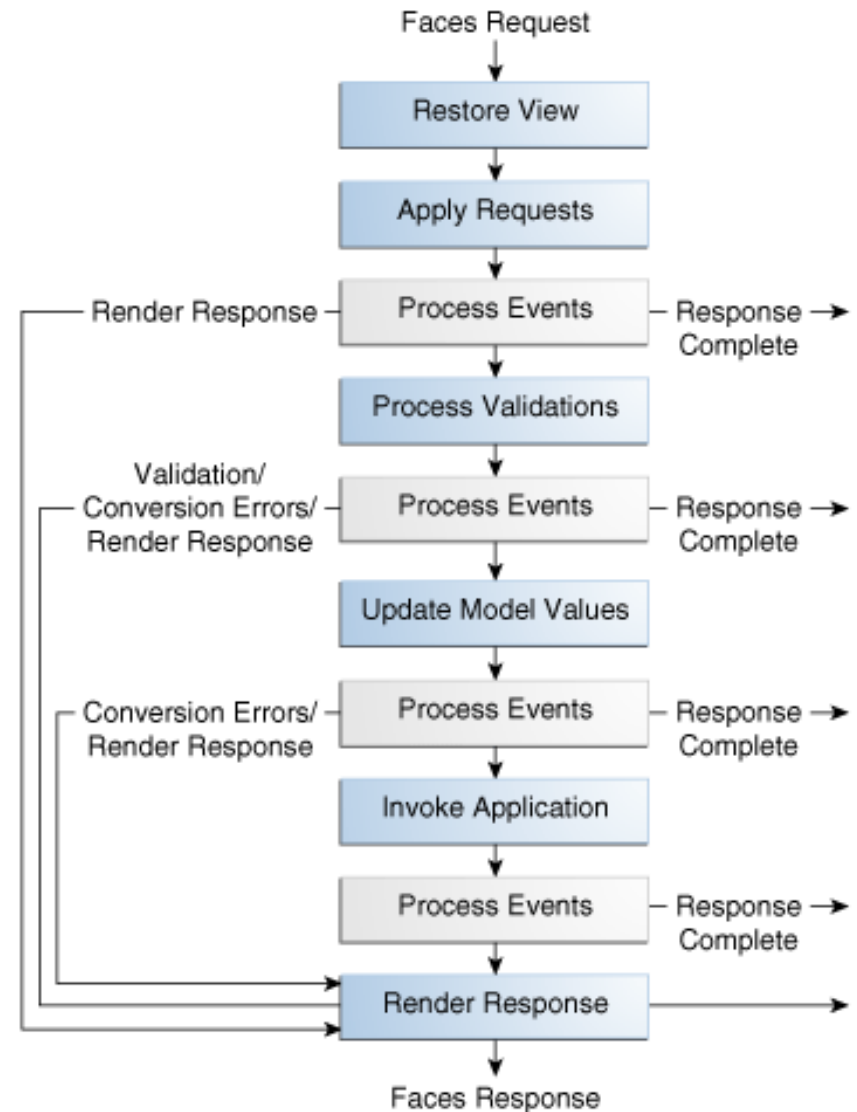
Utility Managed-Bean	utility-bean	application
----------------------	--------------	-------------

Funciones de utilidad para la aplicación. Por ejemplo un FileUploadBean



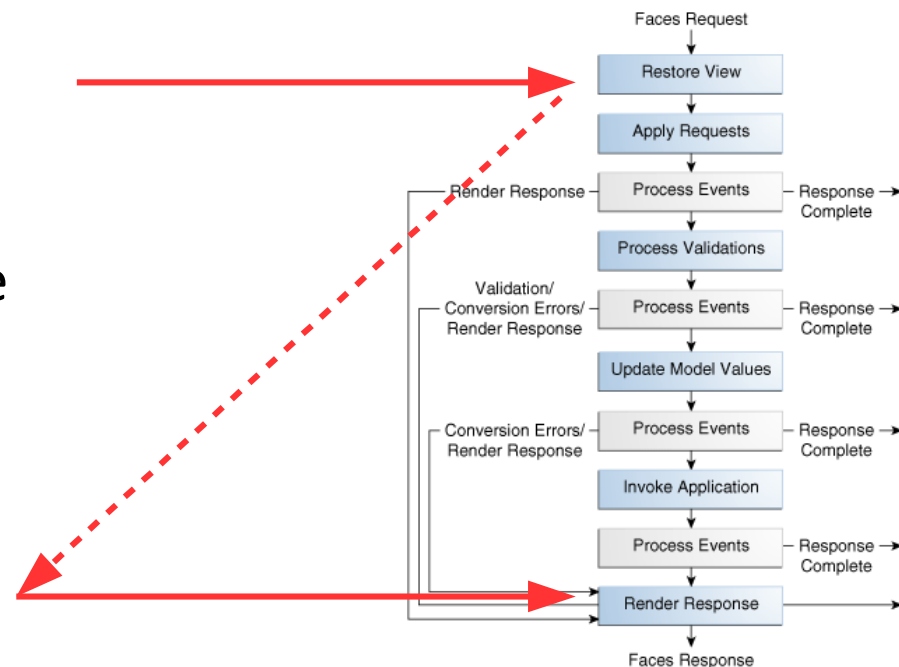
Ciclo de vida de petición JSF

- El ciclo de vida se refiere a las distintas fases de procesamiento por el que pasa la petición, desde que se recibe desde el cliente, hasta que se devuelve el resultado, en nuestro caso renderizado como HTML
- El ciclo de vida de petición de JSF se puede dividir en dos fases principales: Execute y Render
- Execute se divide en subfases



Ciclo de vida request-response JSF

- El ciclo de petición-respuesta de JSF maneja dos tipos de peticiones: iniciales y postback
- La primera vez que se pide una página es la petición inicial
- Postback se produce cuando el usuario envía un formulario contenido en una página que ya había sido cargada a partir de una initial request
- Cuando el ciclo de vida gestiona una petición inicial, ejecuta únicamente las fases **Restore View** y **Render Response**





Ciclo de vida request-response JSF

- En el caso de un postback, donde hay que dar validar y convertir los datos de entrada, se lancen eventos y los datos se propaguen a los beans, ahí si se ejecutan todas las fases
- Las subfases de Execute son las siguientes:
 - ◆ Restore View Phase
 - ◆ Apply Request Values Phase
 - ◆ Process Validations Phase
 - ◆ Update Model Values Phase
 - ◆ Invoke Application Phase
 - ◆ Render Response Phase

<http://docs.oracle.com/javaee/7/tutorial/jsf-intro006.htm>

<http://docs.oracle.com/javaee/7/tutorial/jsf-intro006.htm>



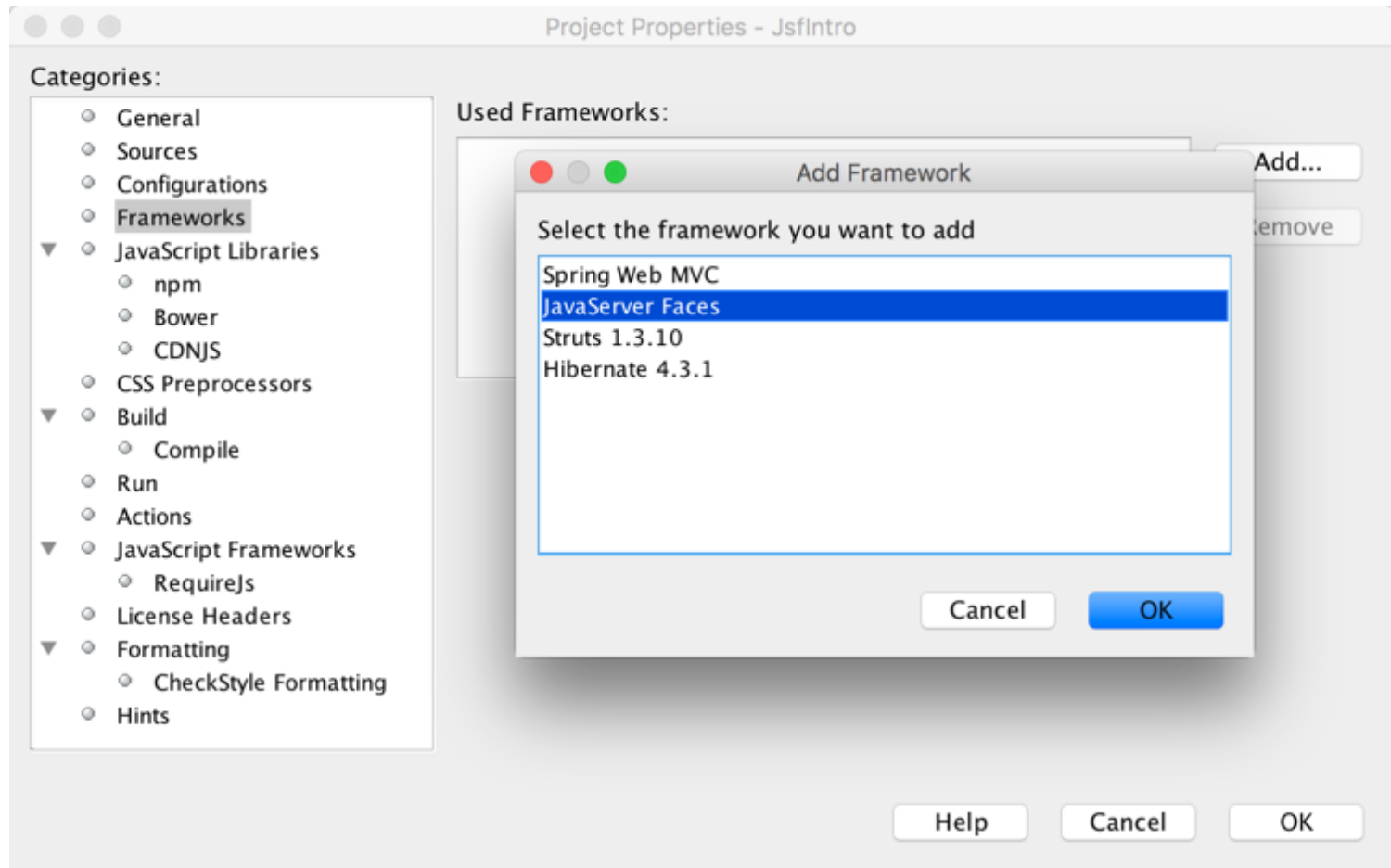
Proyecto básico Java Server Faces





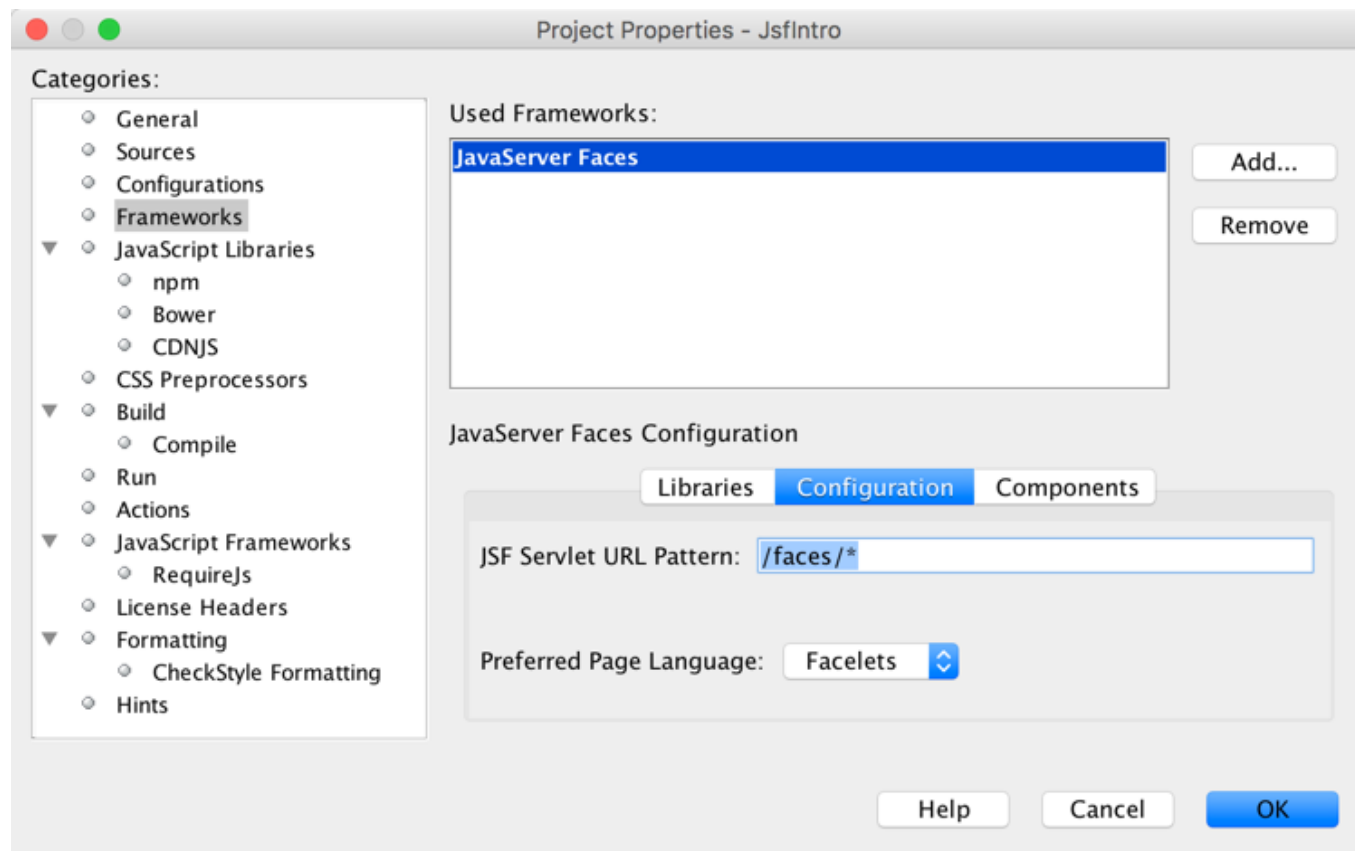
Crear proyecto JSF

- Creamos un proyecto Maven, Java EE 7
- En propiedades de proyecto añadimos Java Server Faces



Configuración de JSF

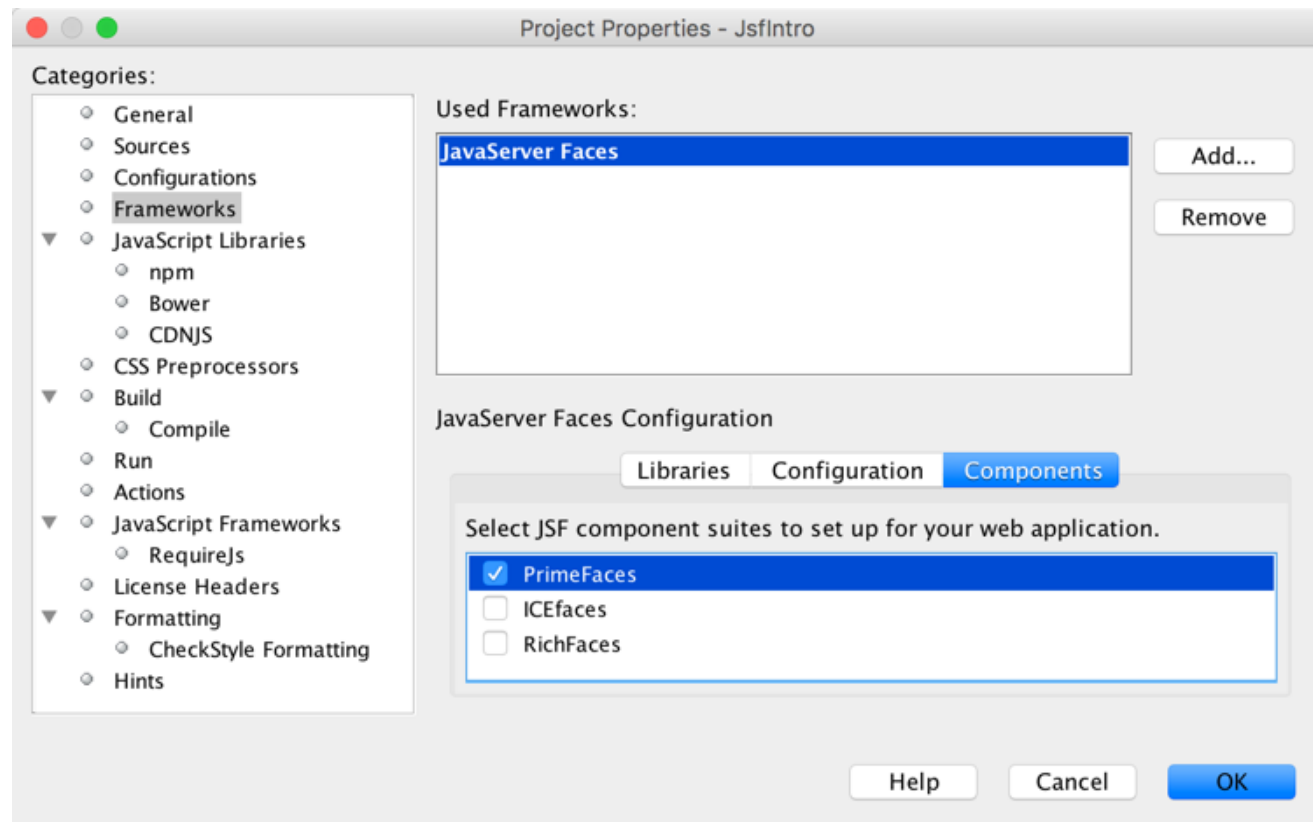
- El Servlet JSF estará mapeado sobre cualquier recurso bajo /faces/*
- Luego nuestras URL deberán incluir el componente faces
- El lenguaje de declaración de vistas Facelets (no usamos JSP)





Librería de componentes

- Vamos a utilizar la librería de componentes PrimeFaces
- Esta librería incluye nuevas *tag libraries* que generan componentes para la vista, mucho más potentes que los estándar JSF





¿Qué ha añadido el asistente?

- En web.xml

- Parámetros de contexto -->
- El servlet de JSF, mapeado contra /faces/

```
<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-name>
  <param-value>Development</param-value>
</context-param>
```

```
-->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
```

- En pom.xml

- Dependencias a primefaces -->

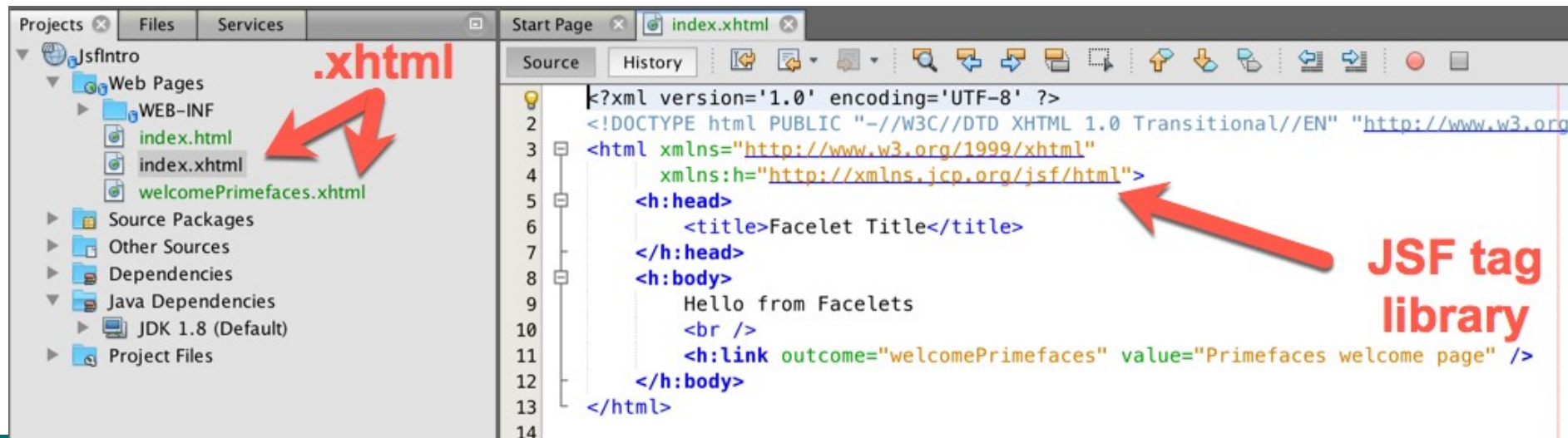
- Páginas .xhtml de ejemplo

```
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>6.0</version>
</dependency>
```



Examinando Facelets

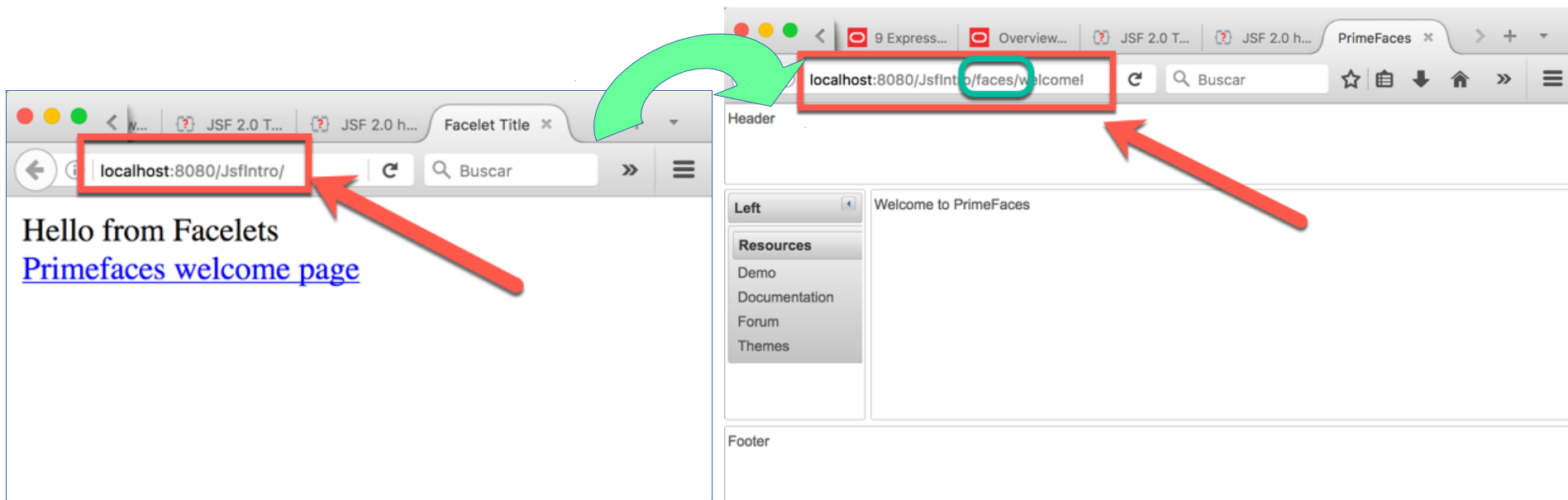
- Vemos que se han añadido dos páginas facelets (.xhtml)
- Lo primero que vemos es que son documentos XML
- Lo segundo una declaración de tag library de JSF con prefijo **h**
 - Los elementos `<h:head>`, `<h:body>` son realmente elementos JSF (que luego renderizarán a `<head>`, `<body>` como HTML)
- Un elemento `<h:link>` con outcome `welcomePrimefaces`. La navegación JSF resolverá a la vista `welcomePrimefaces.xhtml`





Ejecutando el proyecto

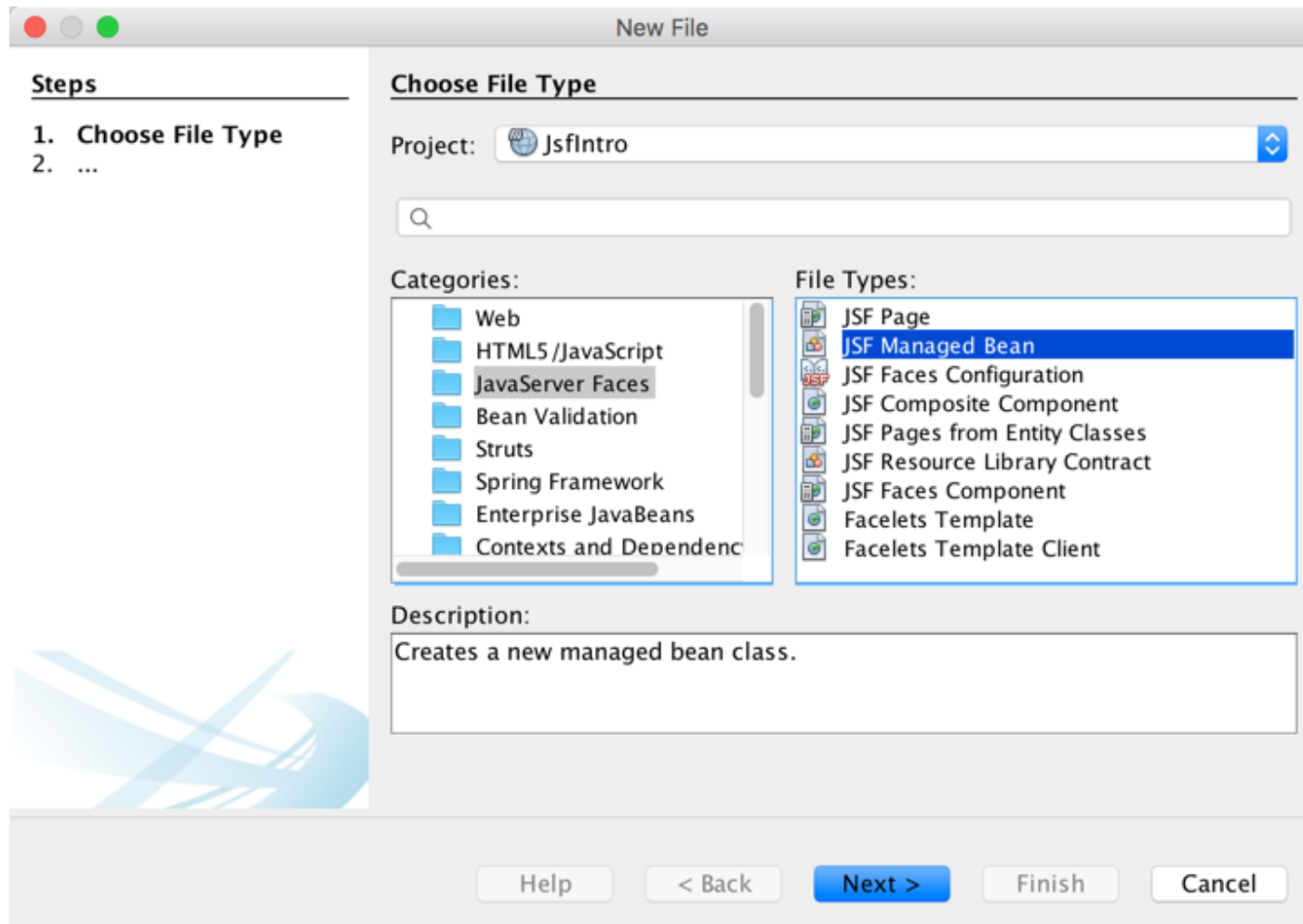
- De mano se carga la página **index.xhtml**, ya que así está definido en web.xml
- Si pulsamos el link veremos que las direcciones generadas por los tag JSF siempre llevan el componente **/faces/**
- Esto es necesario para que “pasen” por el servlet JSF





Añadiendo el primer managed bean

- Fácil: **New > Java Server Faces > JSF Managed Bean**





ActorBean

- Creamos un bean para almacenar las propiedades del actor

New JSF Managed Bean

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

☐ Add data to configuration file

Configuration File:

Name:

Scope:

Nombre para facelets →

→ **Ámbito del bean**

⚠ The file ActorBean.java already exists.

< Back Next > Finish Cancel Help



Página JSF del formulario

- Definimos un form con los tag JSF
- Los input mapean directamente en las propiedades del bean
- El action llama directamente al método save del bean
- La navegación es implícita y viene dado por el valor de retorno de save

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>Nuevo Actor</title>
  </h:head>
  <h:body>

    <h:form>
      <h:panelGrid id="pnlActor" columns="2" border="1">
        <f:facet name="header">
          <h:outputText value="Nuevo actor"/>
        </f:facet>
        <h:outputLabel for="txtFirstName" value="Nombre"/>
        <h:inputText id="txtFirstName" value="${actorBean.firstName}"/>
        <h:outputLabel for="txtLastName" value="Apellidos"/>
        <h:inputText id="txtLastName" value="${actorBean.lastName}"/>
        <h:commandButton value="Insertar" action="#{actorBean.save()}" />
      </h:panelGrid>
    </h:form>
  </h:body>
</html>
```



Código del managed bean

```
@Named(value = "actorBean")
@RequestScoped
public class ActorBean {

    private String firstName;
    private String lastName;

    public ActorBean() {
    }

    public String save() {
        return "ok";
    }

    public String getFirstName() {
        return firstName;
    }

    // ...
}
```

La navegación implícita JSF
buscará la vista ok.xhtml y
la mostrará



Página ok.xhtml

- Accede a las propiedades del bean de request

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
    <h:outputStylesheet library="css" name="main.css" />
  </h:head>
  <h:body>
    <h1>Resultado</h1>
    <p>Insertado actor con los siguientes datos:</p>
    <dl>
      <dt>Nombre</dt>
      <dl>#{actorBean.firstName}</dl>
      <dt>Apellidos</dt>
      <dl>#{actorBean.lastName}</dl>
    </dl>
  </h:body>
</html>
```



Accediendo a Managed Beans

CDI Context and Dependency Injection





CDI

- Context and Dependency Injection
 - Weld es la implementación de referencia
 - Dependency Injection. Permite que un componente simplemente declare una variable de un tipo interfaz que vaya a necesitar. Será la infraestructura CDI quien cree la clase de implementación correspondiente a esa interfaz y la provea al componente
- ~~@ManagedBean(name = "helloWorldBean", eager = true)~~
 - Anotación que define un bean gestionado JSF
 - En CDI será:
- **@Named (name = "helloWorldBean")**





Ejemplo

- En el proyecto ForVago, definimos dos ManagedBean; uno de ámbito aplicación y el otro de ámbito request
- El bean de aplicación **MasterDataBean** tiene un ciclo de vida igual al de la aplicación y se va a utilizar para “cachear” ciertas colecciones (por ejemplo la lista de provincias)
- El bean de request **HotelSearchBean**, sólo pervive el ciclo de una request. Hemos definido que va a proveer la lista de provincias y municipios a los select. El problema que al ser de request, tendría que estar pidiendo continuamente los datos a MySQL. Vamos a hacer que los pida al MasterDataBean, para ahorrarnos llamadas a BD



Ejemplo: MasterDataBean

- En el constructor usa los DAOs para guardarse la lista de provincias y municipios

```
1 /
2 @Named(value = "masterDataBean")
3 @ApplicationScoped
4 public class MasterDataBean {
5
6     private final List<Provincia> provincias = new ArrayList<>();
7     private final List<Municipio> municipios = new ArrayList<>();
8
9     /**
10      * Creates a new instance of MasterDataBean
11      */
12     public MasterDataBean() {
13         ProvinciasDao provDao = DaoFactory.getInstance().getProvinciasDao();
14         this.provincias.addAll(provDao.selectAll(null));
15
16         MunicipiosDao munDao = DaoFactory.getInstance().getMunicipiosDao();
17         this.municipios.addAll(munDao.selectAll(null));
18     }
19
20     public List<Provincia> getProvincias() {
21         return this.provincias;
22     }
23 }
```



Ejemplo: HotelSearchBean

- Necesita a MasterDataBean, pide a CDI que lo inyecte con la anotación @Inject
- El bean CDI no está disponible en el constructor. De ahí el método init anotado con @PostConstruct. En el método @PostConstruct ya es seguro utilizar CDI

```
@Named(value = "hotelSearchBean")
```

```
@RequestScoped
```

```
public class HotelSearchBean implements Serializable {
```

```
    private Short idProvincia;
```

```
    private Short idMunicipio;
```

```
    private List<Municipio> municipios;
```

```
    @Inject
```

```
    MasterDataBean masterDataBean;
```

Indico a CDI que necesito un MasterDataBean, CDI inyectará la referencia al objeto creado

```
    @PostConstruct
```

```
    public void init() {
```

```
        this.municipios = masterDataBean.getMunicipios();
```

```
    }
```

hotelSearchBean no puede utilizar a masterDataBean en el constructor. Esto es debido a que al ser masterDataBean un bean CDI puede que no esté disponible en el constructor de hotelSearchBean

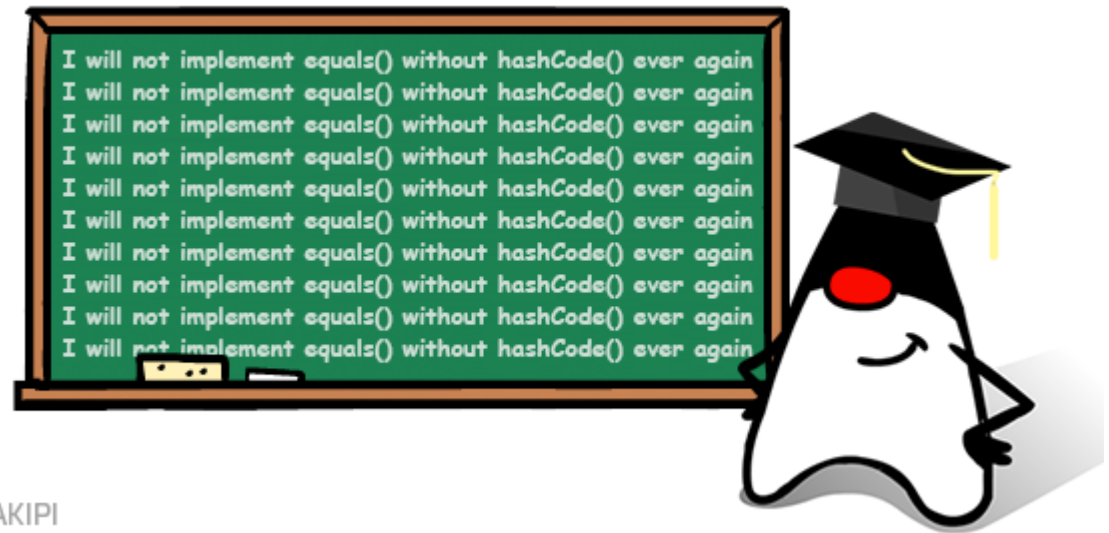
La anotación @PostConstruct define un método de inicialización, donde ya estarán todos los objetos CDI





Expression language (EL)

<https://docs.oracle.com/javaee/7/tutorial/jsf-el.htm>



TAKIPI



Expression Language (EL)

- Permite usar expresiones simples para acceder dinámicamente a componentes Java Bean. La siguiente expresión comprueba si hay ítems en un bean carrito de compra

```
<c:if test="${sessionScope.cart.numberOfItems > 0}">  
    ...  
</c:if>
```

- EL permite realizar las siguientes tareas
 - Lectura de propiedades de JavaBean y objetos implícitos
 - Escritura de datos, como form input, en componentes JavaBean
 - Invocar métodos public y static
 - Ejecución de operaciones aritméticas, booleanas y de tipo String
 - Construcción dinámica de colecciones y ejecución de operaciones sobre colecciones



Evaluación inmediata o diferida

- Evaluación inmediata indica que la expresión es evaluada y obtenido el resultado tan pronto como la página es renderizada por primera vez

\${persona.edad} ←se evalúa a int inmediatamente

- Evaluación diferida implica que la expresión es evaluada en un tiempo posterior, en alguna de las fases del ciclo de vida JSF

#{pesona.edad} ←se evalua en alguna fase posterior

- JSF usa fundamentalmente evaluación diferida. Estas expresiones permiten:
 - *Value expressions* que se pueden usar para leer o escribir datos
 - *Method expressions*



Ejemplos

- Referencia a métodos estáticos: *classname.field*
 - **Boolean.FALSE**
- Acceso a arrays o List: *var[index]*
 - `${customer.orders[1]}`
- Acceso a Map: *var["key"]* ó *var.key* (siendo key de tipo String)
 - `${provincias["tenerife"]}` `${provincias.tenerife}`
- Invocación de métodos: *expra[exprb](prms)* *expra.identb(prms)*
 - `#{userNumberBean.userNumber('5')}`
- *Lambda expressions*



Operaciones en objetos Collection

- Construcción de un set:
 - {1, 2, 3}
- Construcción de un list
 - [1, 2, 3]
 - [1, “dos”, [tres, cuatro]]
- Construcción de un map
 - {“one”:1, “two”:2, “three”:3}
- Ver más: <https://docs.oracle.com/javaee/7/tutorial/jsf-el004.htm>



Operadores

- **Arithmetic:** +, - (binary), *, / and div, % and mod, - (unary).
- **String concatenation:** +=.
- **Logical:** and, &&, or, ||, not, !.
- **Relational:** ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le. Comparisons can be made against other values or against Boolean, string, integer, or floating-point literals.
- **Empty:** The empty operator is a prefix operation that can be used to determine whether a value is null or empty.
- **Conditional:** A ? B : C. Evaluate B or C, depending on the result of the evaluation of A.
- **Lambda expression:** ->, the arrow token.
- **Assignment:** =.
- **Semicolon:** ;.



Links

- Gestionar excepciones View Expired
 - <http://stackoverflow.com/questions/4992526/how-to-handle-session-expiration-and-viewexpiredexception-in-jsf-2>
- Validaciones JSF 2.2
 - <http://incepttechnologies.blogspot.com.es/p/validation-in-jsf.html>
- Validación múltiples componentes
 - <https://www.mkyong.com/jsf2/multi-components-validator-in-jsf-2-0/>
- Listeners
 - <http://www.journaldev.com/7028/jsf-event-listener-action-phase-value-change>
- Choose the right bean scope
 - <http://stackoverflow.com/questions/7031885/how-to-choose-the-right-bean-scope/>



PREGUNTAS

