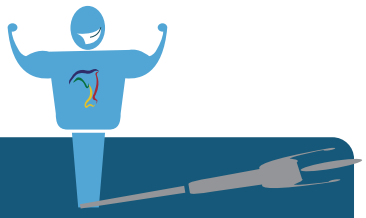
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

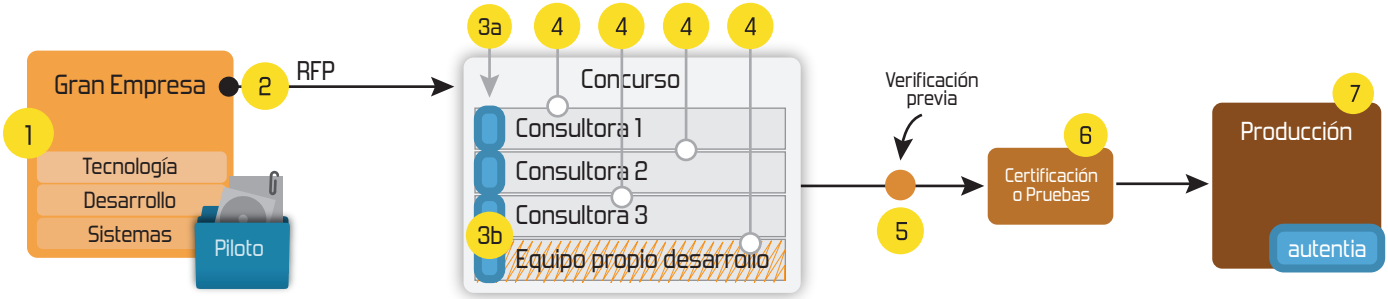
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Lanzado TNTConcept versión 0.6 (12/07/2007)

Desde Autentia ponemos a vuestra disposición el software que hemos construido (100% gratuito y sin restricciones funcionales) para nuestra gestión interna, llamado TNTConcept (auTeNTia).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Acegi, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño ..... Saber más en: http://tntconcept.sourceforge.net/

Contacte con Cristóbal González
criskerberos-tutoriales@yahoo.com

Autor: Cristóbal González Almirón es consultor de desarrollo de proyectos informáticos.

Su experiencia profesional se ha desarrollado en empresas como Compaq, HP, Mapfre, Endesa, Repsol, Universidad Autónoma de Madrid, en las áreas de Desarrollo de Software (Orientado a Objetos), tecnologías de Internet, Técnica de Sistemas de alta disponibilidad y formación a usuarios.

Descargar este documento en formato PDF EclipseJSF.pdf

Fecha de creación del tutorial: 2006-11-24

Java Server Faces con Eclipse

Introducción a la tecnología Java Server Faces

En nuestro trabajo diario en Autentia realizamos la construcción de portales para clientes utilizando las tecnologías

web más modernas. Una de estas tecnologías es el estándar Java Server Faces, que nos permite desarrollar aplicaciones empresariales robustas, en tiempos de desarrollo cortos, con una interfaz web amigable.

Cuando queremos hacer aplicaciones web empresariales, normalmente utilizamos algún framework o marco de trabajo que nos facilite la tarea. En el mundo Java disponemos de varios frameworks para este tipo de aplicaciones. En este tutorial le vamos a echar un vistazo al framework de Java Server Faces, que es un framework para desarrollo de aplicaciones web que utiliza el patrón de arquitectura MVC de tipo 2 (Arquitectura Modelo-Vista-Controlador con controlador monolítico).
No vamos a extendernos en este punto, pues no es el objetivo del tutorial, sino que vamos a describir un framework Java de este tipo: Java Server Faces.

El framework JSF instala en nuestra aplicación web un controlador que se encarga de la gestión de los formularios y de la navegación entre páginas. Toda la definición de las páginas y de la navegación entre las mismas. Esta es la ventaja principal de un controlador monolítico y con un fichero de configuración central: toda la lógica de la aplicación se define en un único punto.

El modelo de funcionamiento de JSF es sencillo, muy semejante al usado en Struts. Básicamente, cada página de nuestra aplicación está compuesta de un formulario (se pueden usar varios pero no es el modo adecuado).Al igual que en Struts, JSF permite asociar funciones lógicas, llamadas acciones, para cada formulario, de manera que al enviar el formulario con el botón “submit” se procese la acción correspondiente. En este momento el controlador de la aplicación decidirá, en función del valor retornado por la función, cuál es la página que se debe mostrar para el resultado de la función. Este mecanismo es el que produce la navegación por las páginas de la aplicación.

Pero además JSF introduce el procesamiento de los formularios por etapas. Cuando el formulario es enviado, el controlador realiza el procesamiento de las acciones asociadas al formulario en diferentes etapas. Durante estas etapas:

1. Se obtienen los valores de la página
2. Se procesan los eventos
3. Se realizan las validaciones de los valores
4. Se procesan las acciones
5. Se realiza el render (dibujado) de la nueva vista de la página
6. Se realiza la navegación, si es necesario, a otras páginas.

Lo más interesante de este modelo, como veremos al comentar las acciones inmediatas, es que no se deben completar todas las fases, y el proceso se puede interrumpir antes de llegar a la etapa de navegación, con lo que obtendríamos la misma página, pero con una vista actualizada. Este mecanismo lo utilizaremos para completar datos en la página (rellenar combos en el formulario), para realizar validaciones de parámetros, para reaccionar frente a eventos en los elementos del formulario, etc.

Ahora que ya hemos dado un vistazo por encima a JSF, pongámosnos manos a la obra.

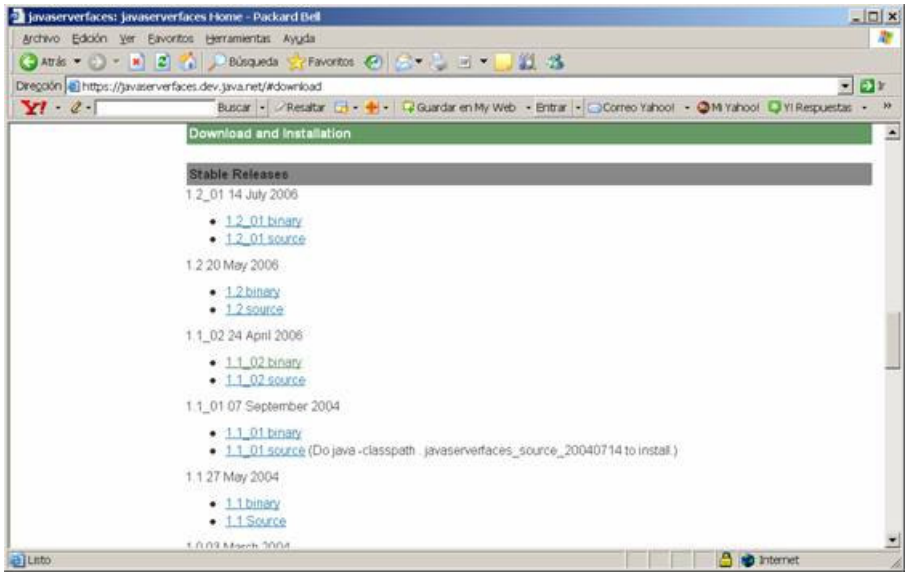
Instalación del plugin de JSF para Eclipse

Para realizar los ejemplos vamos a utilizar el entorno de desarrollo Eclipse, al que le vamos a añadir los elementos necesarios para desarrollar una página JSF.

Descargamos el plugin de JSF de Eclipse y lo instalamos

Descargando una implementación de JSF
En la página de http://javaserverfaces.dev.java.net podemos descargar una implementación de la biblioteca JSF de Sun Microsystems.

Nos vamos a la página de descargas.

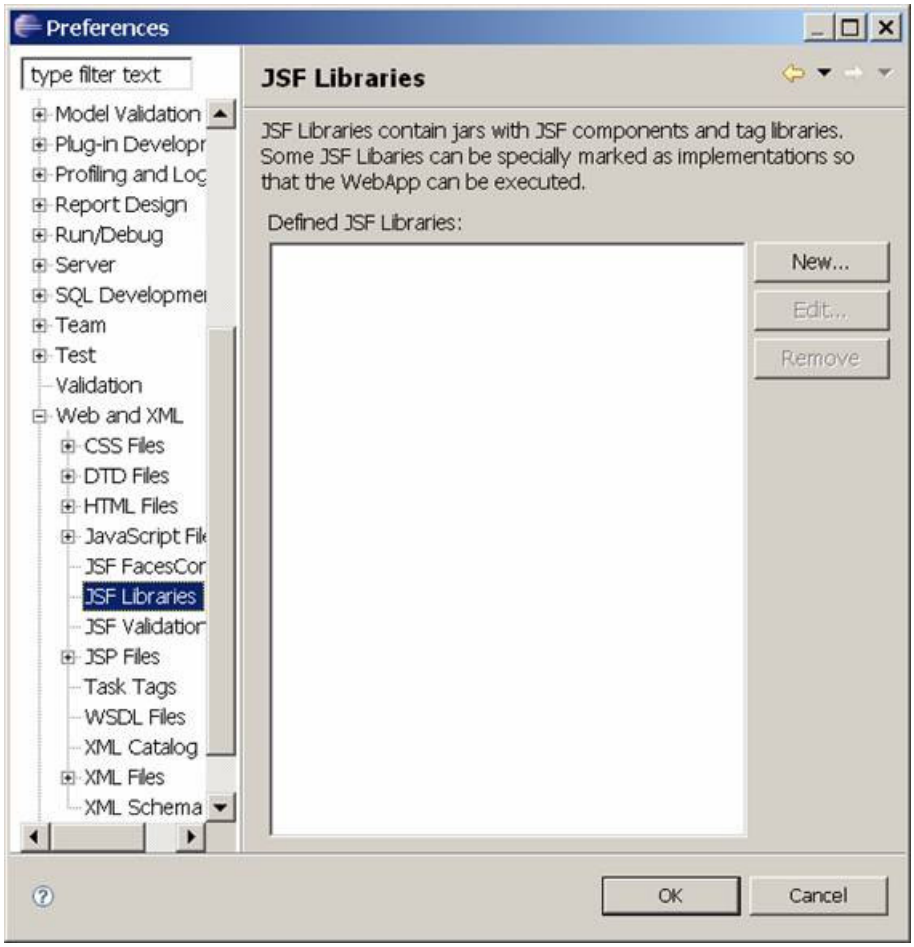


Vamos a probar la versión jsf-1_1_02.zip, por lo que nos descargamos el fichero y lo descomprimimos a una carpeta.

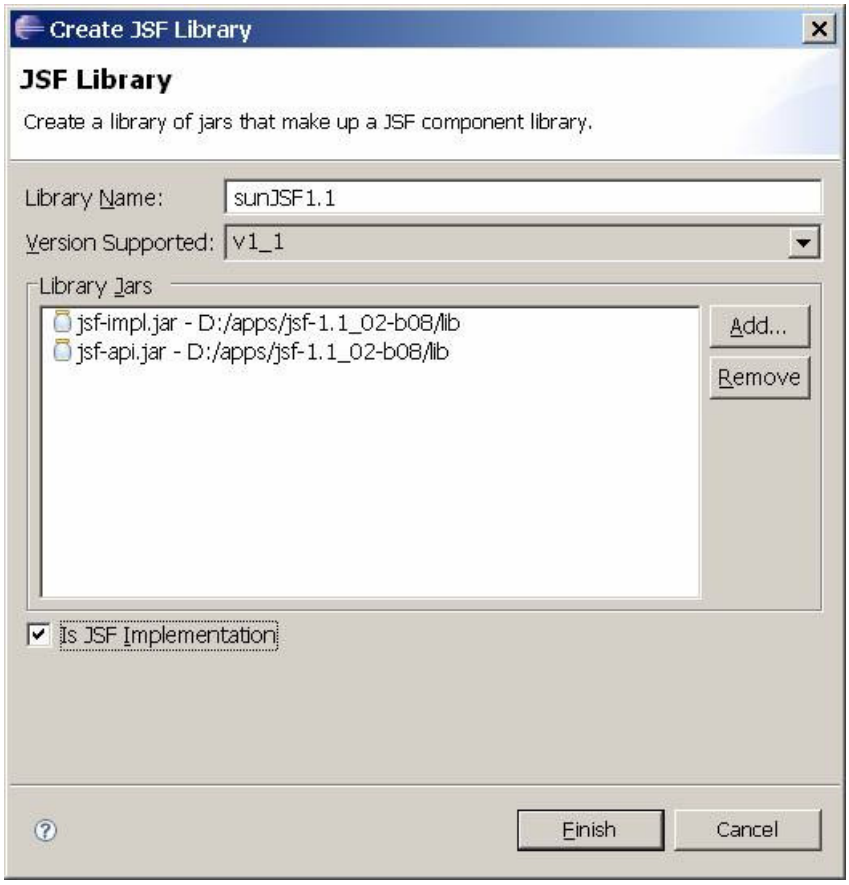
En la página vemos que ya está disponible la versión 1.2 de JSF. Usaremos la versión 1.1 pues es la que soporta nuestro plugin de Eclipse, y además porque está más revisada.

Registrando las bibliotecas de Sun para JSF 1.1

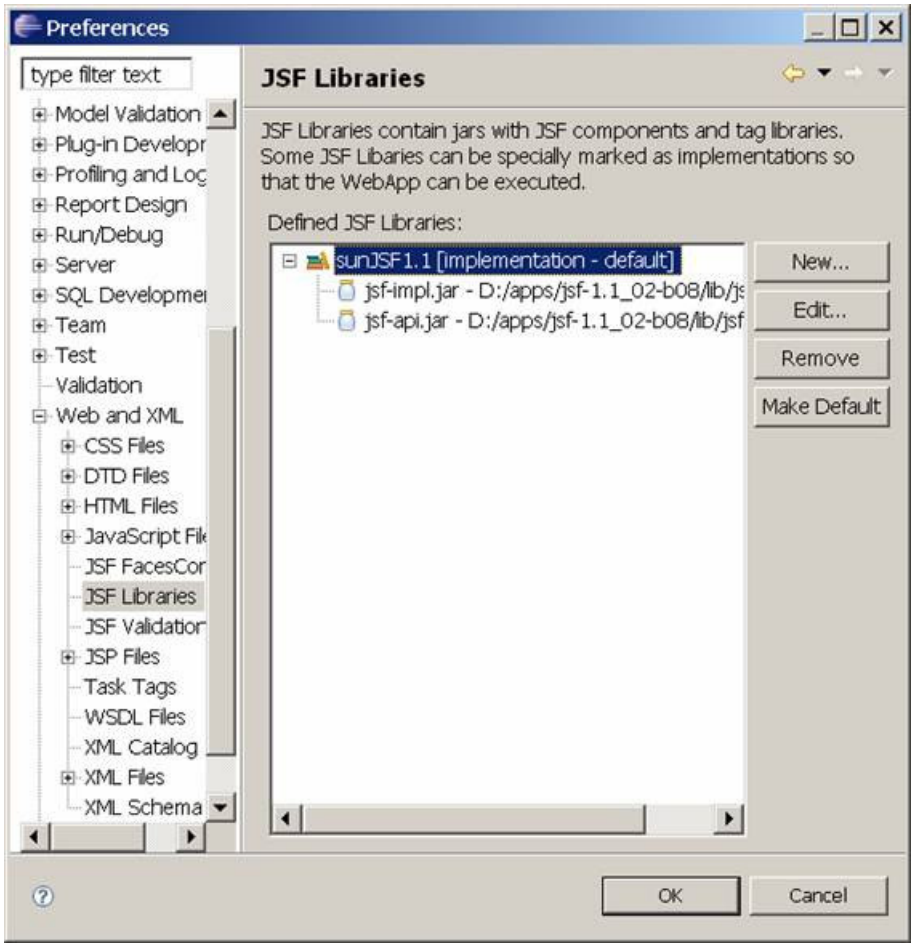
Antes de comenzar a utilizar las bibliotecas en el entorno Eclipse. Para ello nos vamos al menú del Eclipse "Ventanas\preferencias" y buscamos en el apartado "Web and XML\JSF libraries"



Pulsamos en el botón “New…”



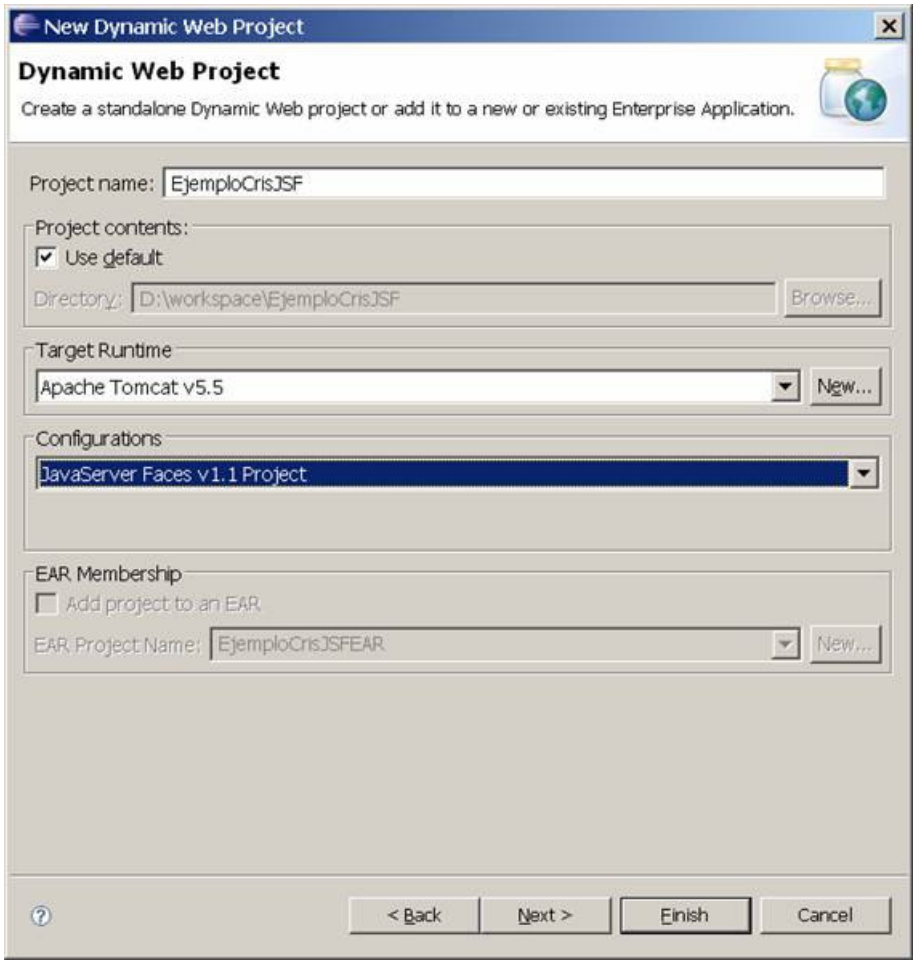
Hemos elegido un nombre para la biblioteca “sunJSF1.1”, hemos indicado que cumple la especificación JSF 1.1, añadimos las dos bibliotecas y marcamos la casilla “is JSF implementation”, luego pulsamos “Finish”.



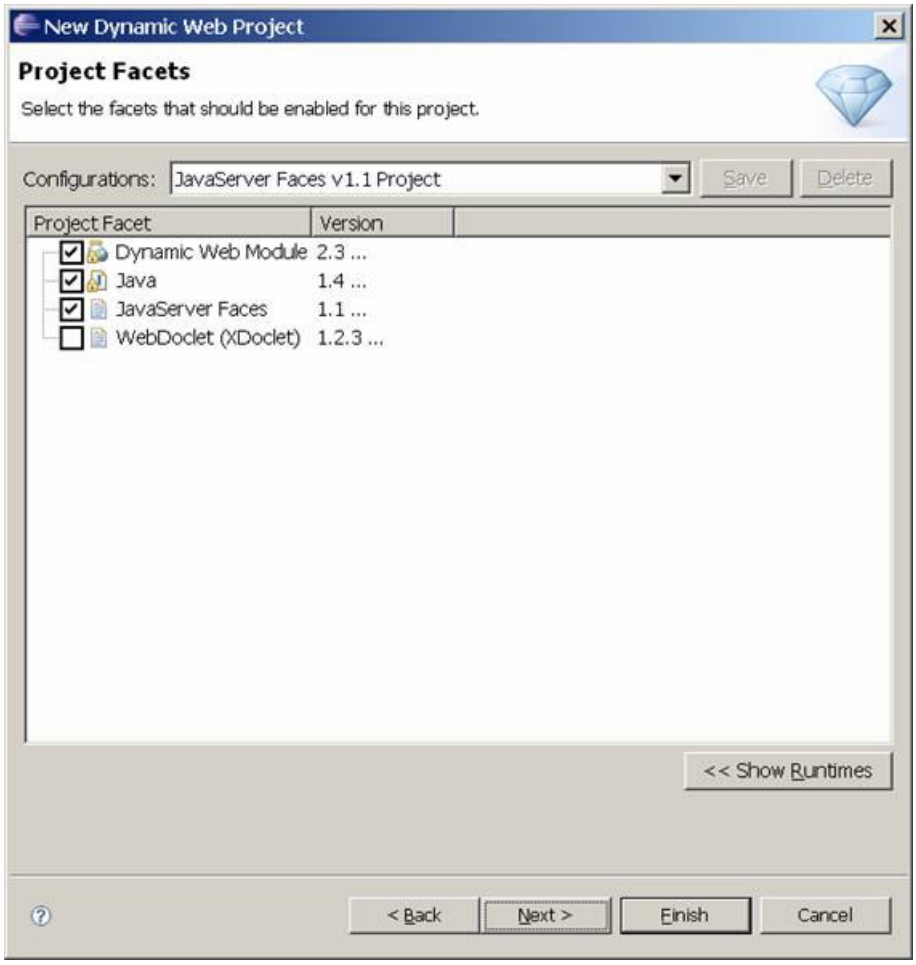
Con ello ya hemos registrado la biblioteca JSF de Sun.
También vamos a añadir otros archivos jar organizados en dos bibliotecas adicionales: commons y jslt.

Creación de un nuevo proyecto JSF

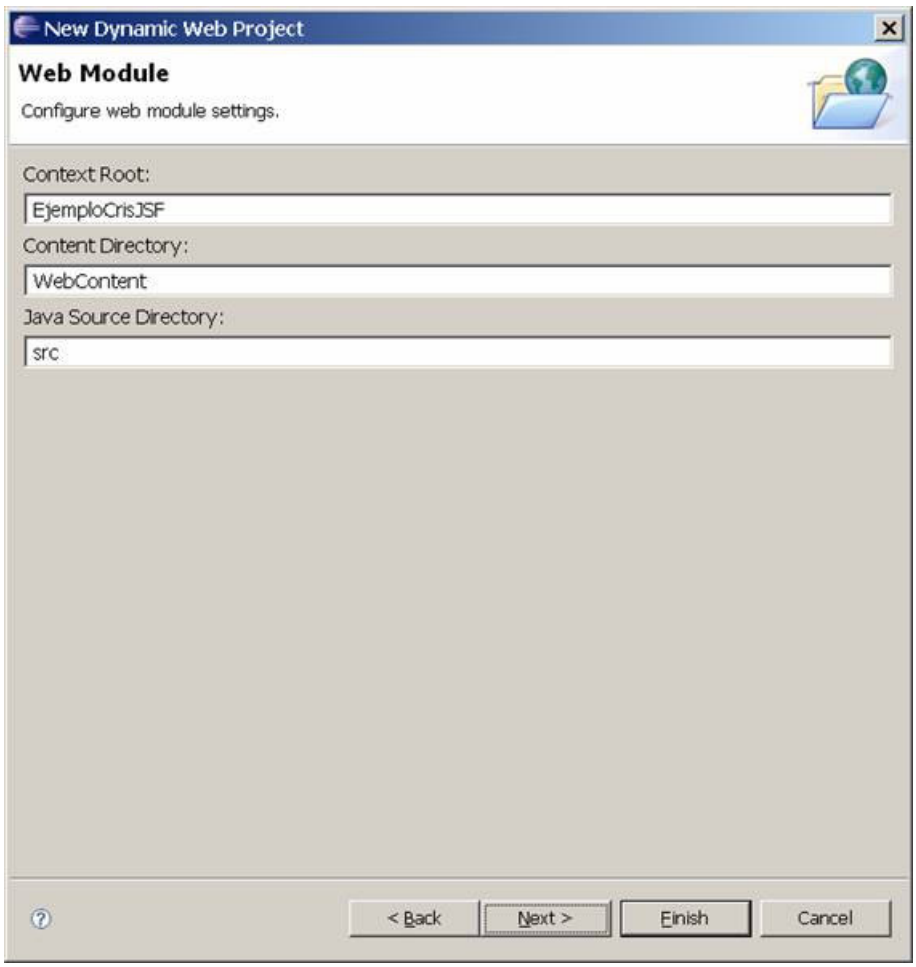
Pulsamos en nuevo proyecto y elegimos “Dynamic Web”



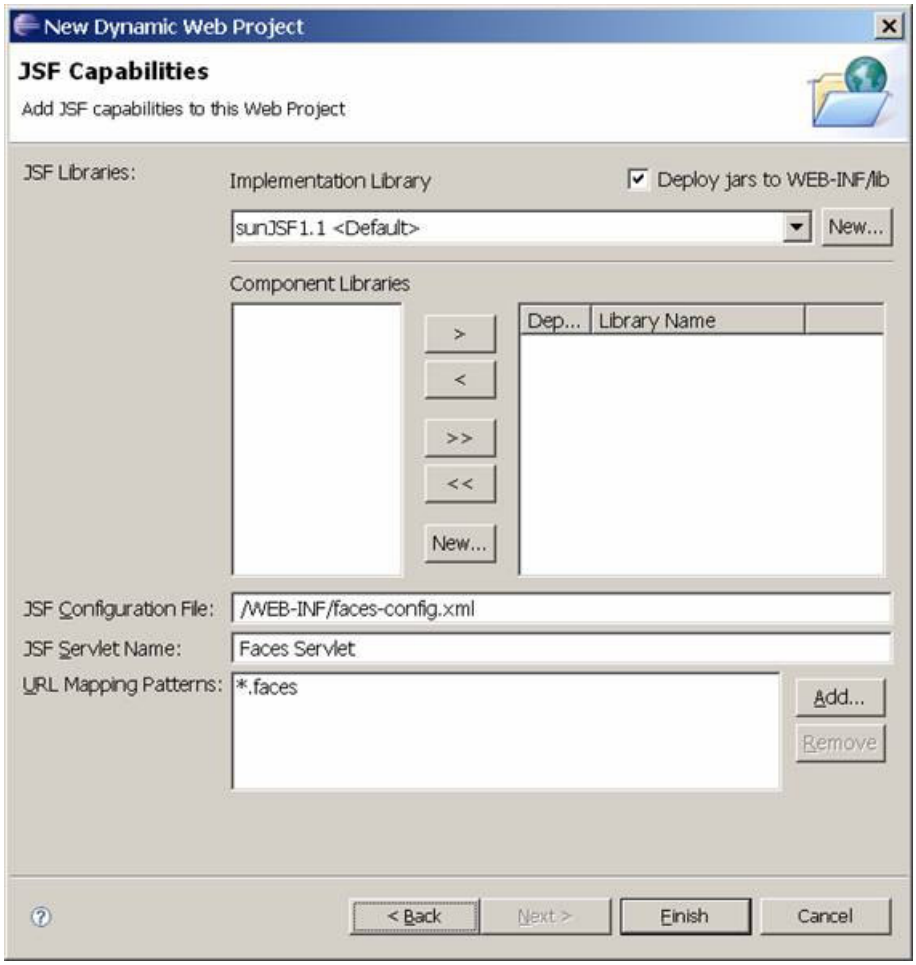
Le he puesto de nombre al proyecto EjemplCrisJSF. Como Runtime tengo el Tomcat 5.5, para que el Eclipse haga automáticamente el despliegue en el Tomcat. Además en
"Configuration" he elegido "Java Server Faces 1.1" y pulsamos en "Next"



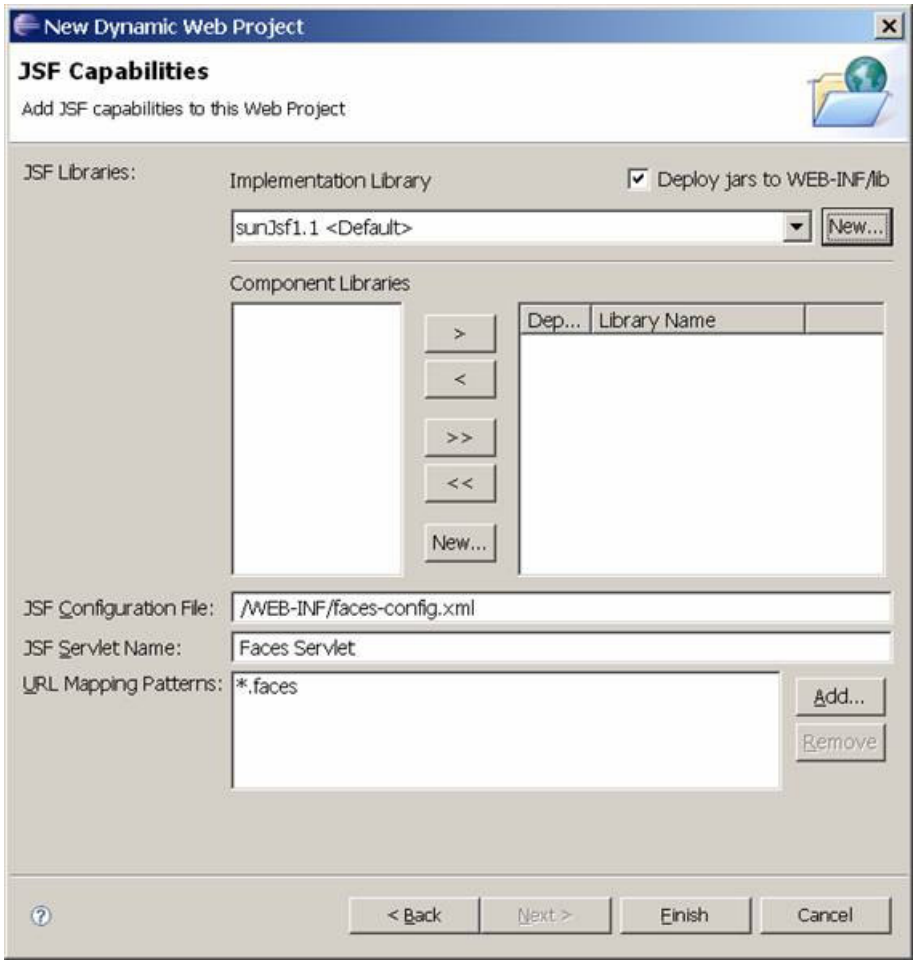
Como vemos, se ha añadido la faceta de Eclipse para Java Server Faces 1. Pulsamos en “Next”.



Las propiedades del Web Module las dejamos por defecto. Pulsamos en “Next”.



En el cuadro de diálogo elegimos todas las bibliotecas .jar que están en la carpeta lib de la implementación. Además le tenemos que poner un nombre, y pulsamos “Finish”.



Pulsamos “Next” y finalizará el proceso de creación del proyecto.

También vamos a copiar a la carpeta de biblioteca, que está en WebContent\WEB-INF\lib un par de bibliotecas del Tomcat, en concreto las bibliotecas jstl.jar y estandar.jar que encontraremos en la carpeta del Tomcat webpps\jsp-samples\WEB-INF\lib. Lo podríamos haber hecho también siguiendo el método usado para las bibliotecas de JSF, pero en este caso hemos optado por el método simple.

Un vistazo al plugin de Eclipse para JSF

Para ver en acción el plugin de JSF abrimos desde Eclipse el fichero feces-config.xml. Veremos que Eclipse abre una ventana de edición especial para este fichero.

Como vemos el plugin añade el fichero faces-config.xml al proyecto web, en la carpeta WEB-INF. Si pulsamos sobre el fichero se abre un editor visual para el fichero. También vemos que el plugin ha modificado el fichero web.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp_ID">
	<display-name>EjemploCrisJSF</display-name>
	<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>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
		<welcome-file>index.htm</welcome-file>
		<welcome-file>index.jsp</welcome-file>
		<welcome-file>default.html</welcome-file>
		<welcome-file>default.htm</welcome-file>
		<welcome-file>default.jsp</welcome-file>
	</welcome-file-list>
</web-app>
```

Creación de la página de inicio JSF

Nuestra primera página será una típica página principal, con un texto de presentación y un formulario de inicio de sesión. Para ello necesitamos crear lo siguiente:

1. Un bean que guarde la información que va a introducir el usuario en el formulario.
2. Una página de inicio JSP que pida usuario y contraseña y realice una acción en el formulario
3. Dar de alta el bean dentro del módulo JSF

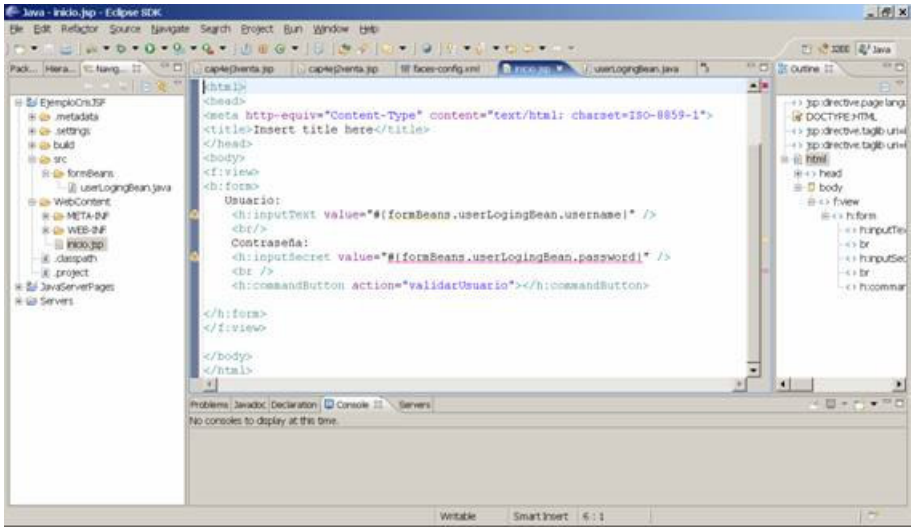
Para crear el bean vamos a crear una clase que luego llevaremos al WEB-INF\classes. Dentro de Eclipse lo hacemos creando la clase userLoggingBean en la carpeta src\formBeans, que hemos creado al efecto.

userLogingBean.java

```
package formBeans;

public class userLogingBean {
	private String username;
	private String password;

	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;
	}

	public String validarUsuario() {
		if (username.equals("usuario") && password.equals
("sesamo"))
			return "ok";
		else
			return "badUsername";
	}



}
```

Creamos un archivo inicio.jsp como sigue:



El texto de la página es

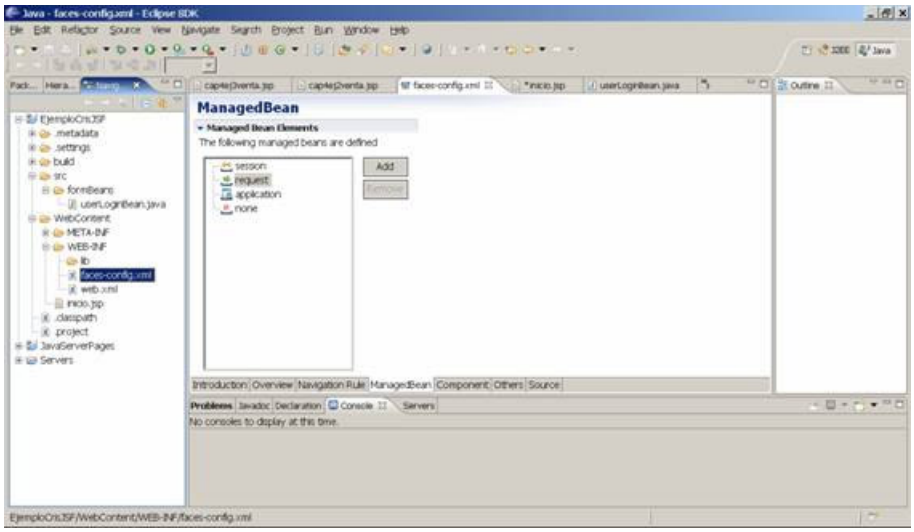
inicio.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>Insert title here</title>
</head>
<body>
<f:view>
<h:form>
   Usuario:
      <h:inputText value="#{userLoginBean.username}" />
      <br/>
      Contraseña:
      <h:inputSecret value="#{userLoginBean.password}" />
      <br />
      <h:commandButton action="#{userLoginBean.validarUsuario}"
value="Iniciar"></h:commandButton>

</h:form>
</f:view>

</body>
</html>
```

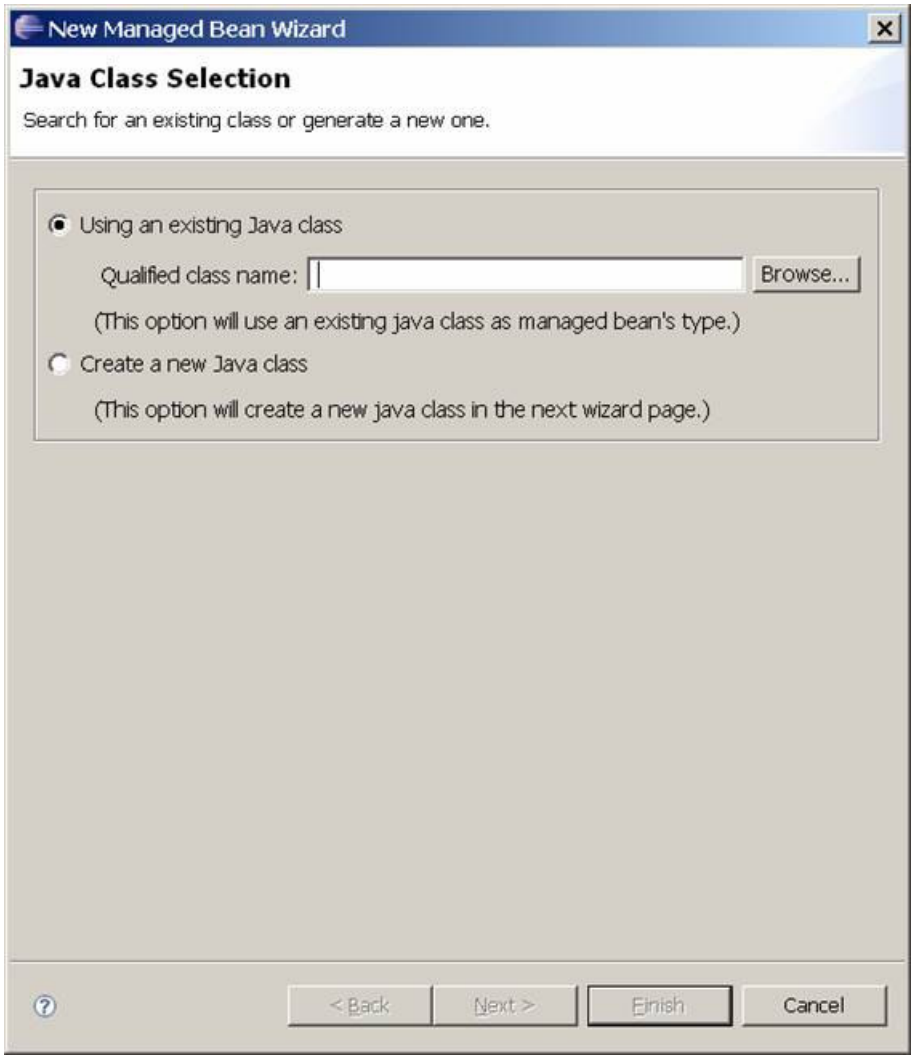
El Eclipse se está quejando de que el formBean.userLogingBean no lo puede resolver. Hay que darlo de alta en los manager beans del plugin de Eclipse.

Para ello pulsamos sobre el fichero faces-config.xml y nos vamos a la pestaña “Managed Beans” que nos mostrará el

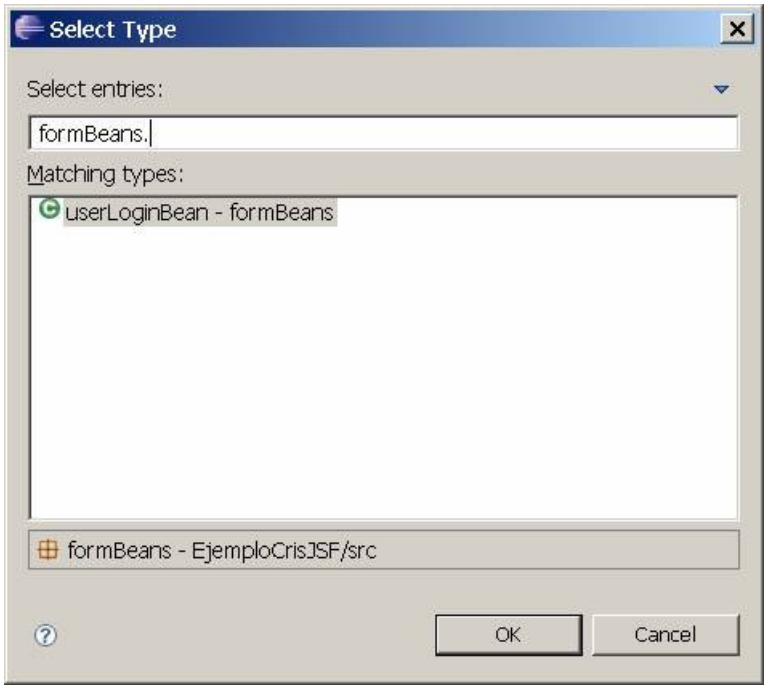
plugin JSF



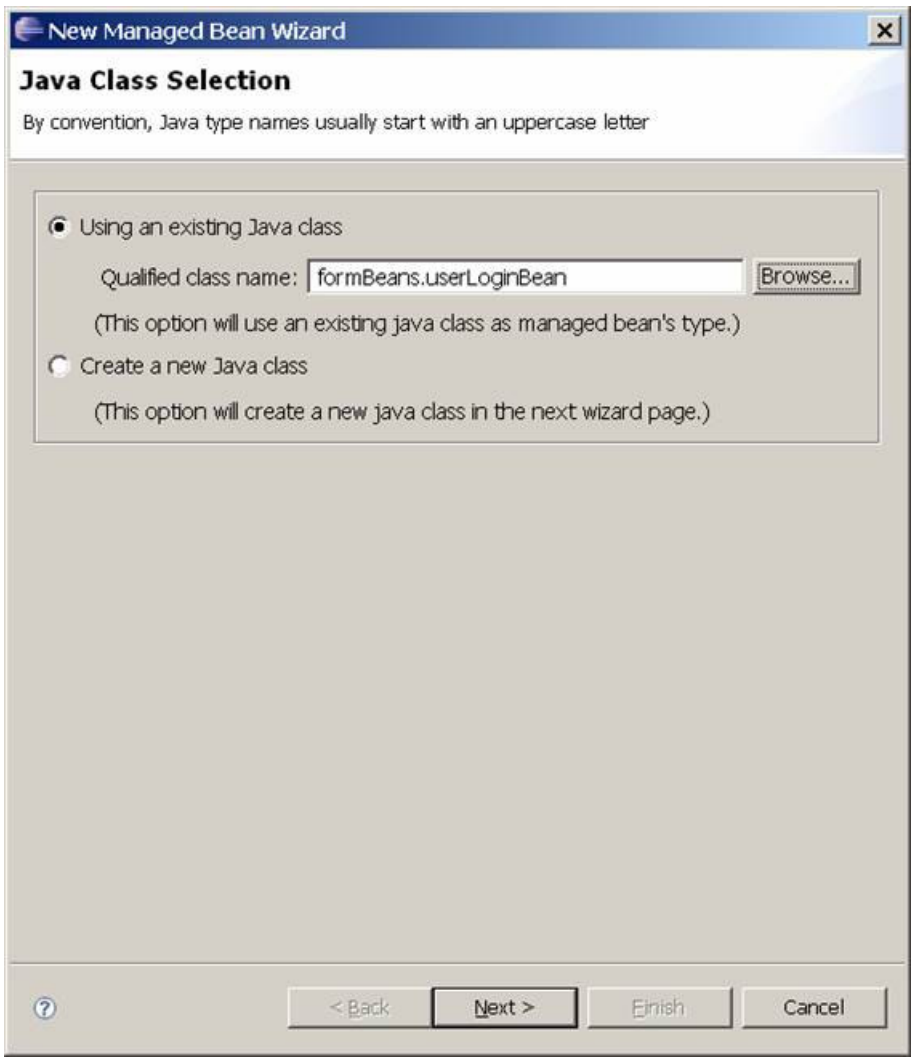
Ahora ya podemos darlo de alta. Pulsamos sobre request y luego el botón de Add…



Ahora pulsamos “Browse” para localizar la clase (somos vagos o precavidos…).

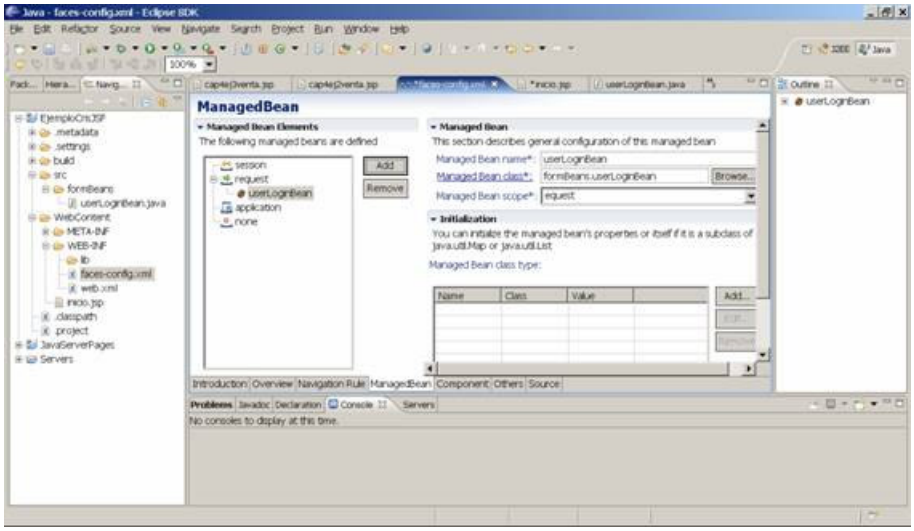


En cuanto que escribimos formBeans. (que es el paquete donde está nuestra clase) el plugin la encuentra. La seleccionamos y pulsamos OK.



Pulsamos “Next”

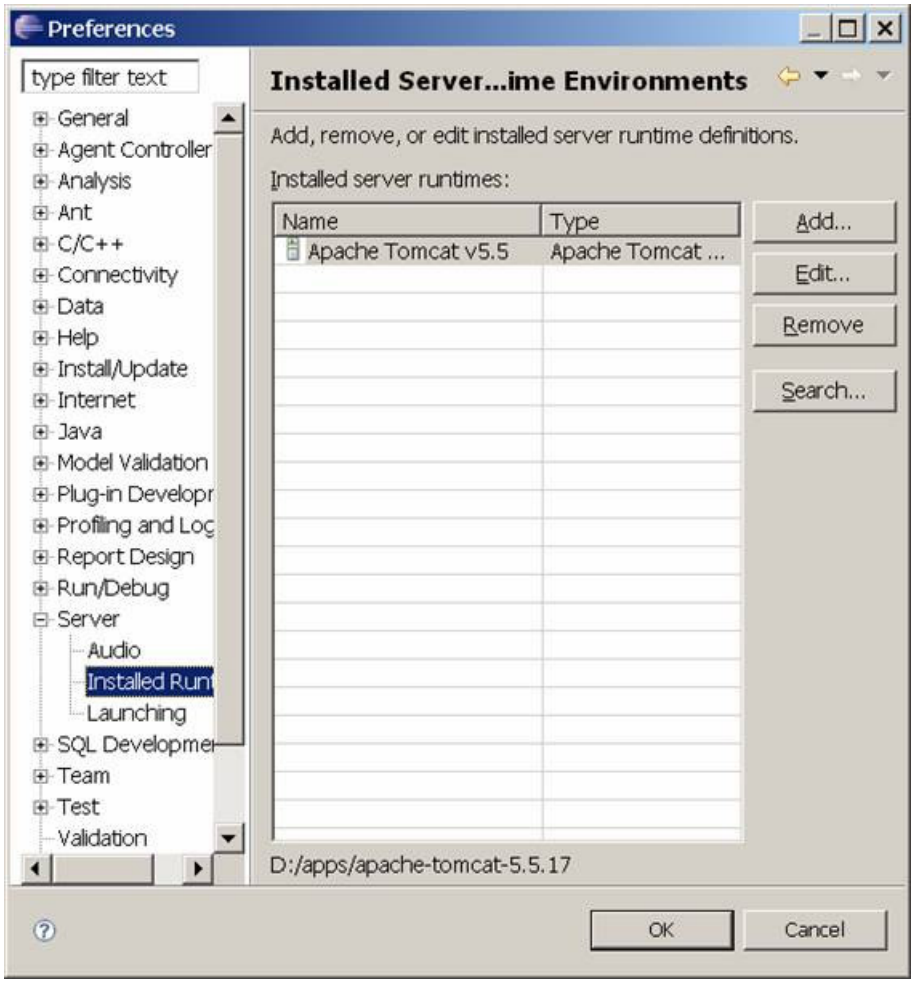
Pulsamos “Finish”



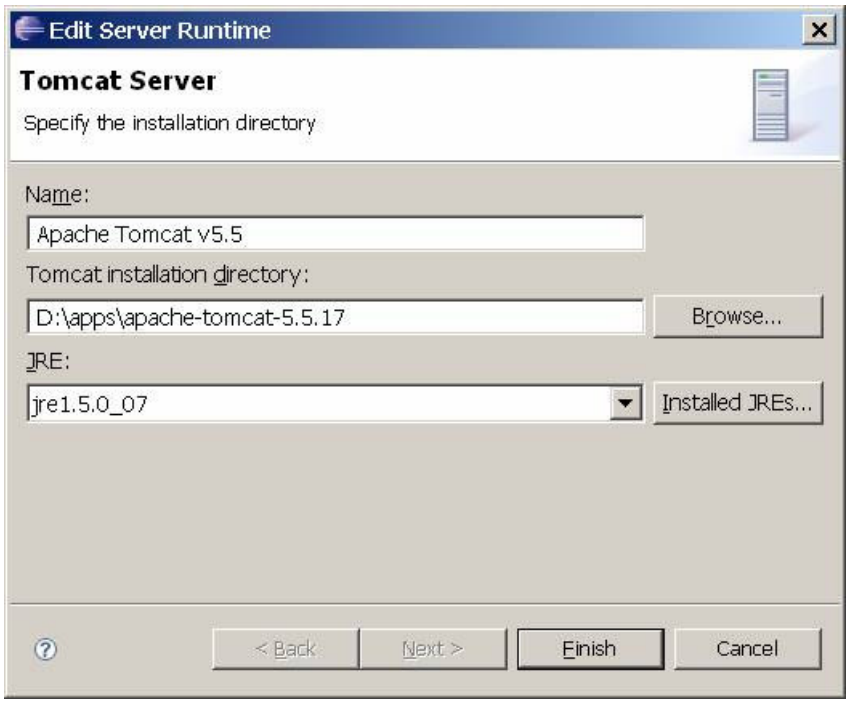
Como vemos, ha cambiado un poco la cosa…

Probando la aplicación

Para probar la aplicación vamos a utilizar el servidor Tomcat integrado en Eclipse. Si no lo tienes configurado, es sencillo. Descomprime el Tomcat 5.5 en una carpeta, por ejemplo “D:\apps\apache-tomcat-5.5.17”, y luego abre el cuadro de diálogo de preferencias del Eclipse, seleccionando “Windows\Preferences…”

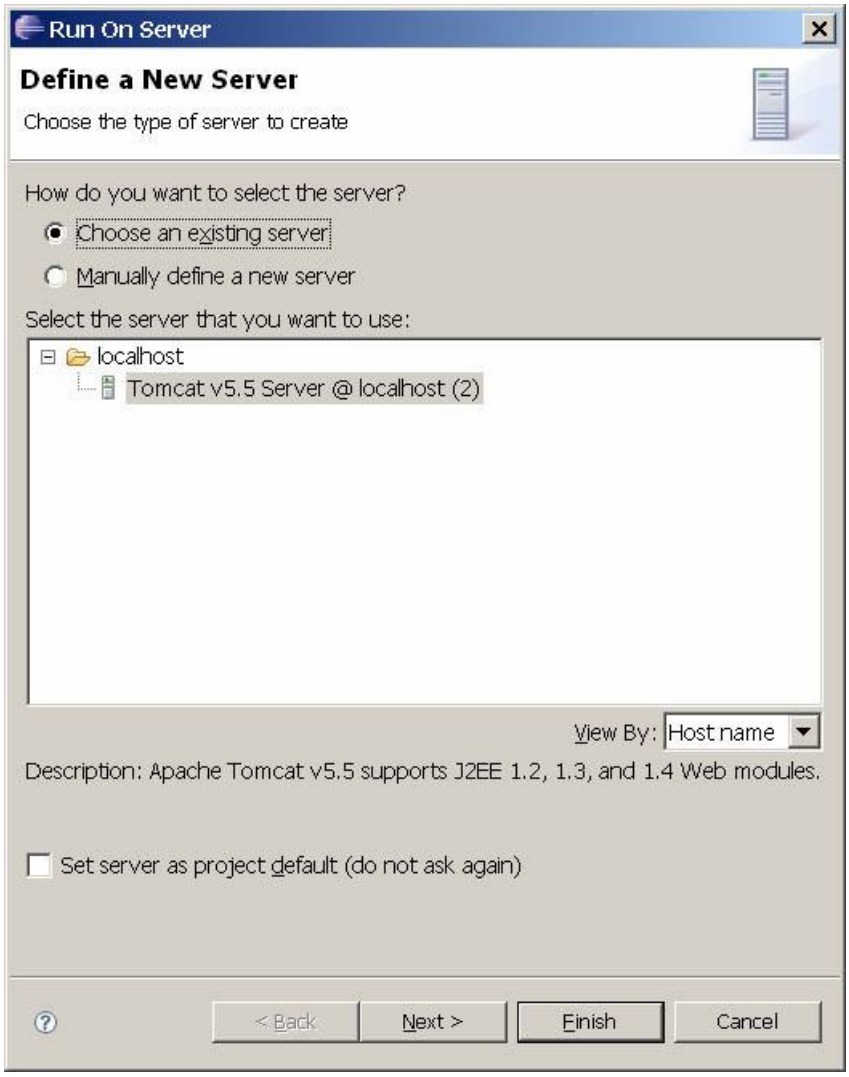


Si no lo tienes añadido, tal y como aparece en la imagen, pulsa en “Add…”

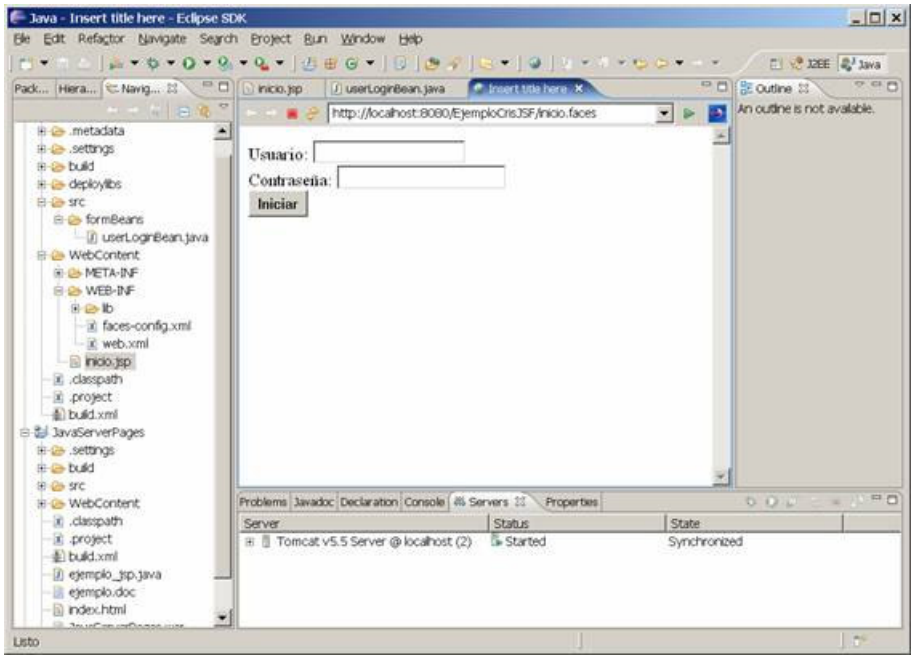


Como vemos, sólo hay que añadir el nombre, la carpeta donde está instalado y el JRE que va a utilizar.

Probamos la aplicación. Para ello pulsamos en con el botón derecho sobre el archivo inicio.jsp, y elegimos “Run as…\Run on Server”



y elegimos el nuevo servidor para la ejecución quen nos aparece en el cuadro de diálogo que nos ofrece el Eclipse. Finalmente el resultado lo vemos con el navegador integrado en el Eclipse.

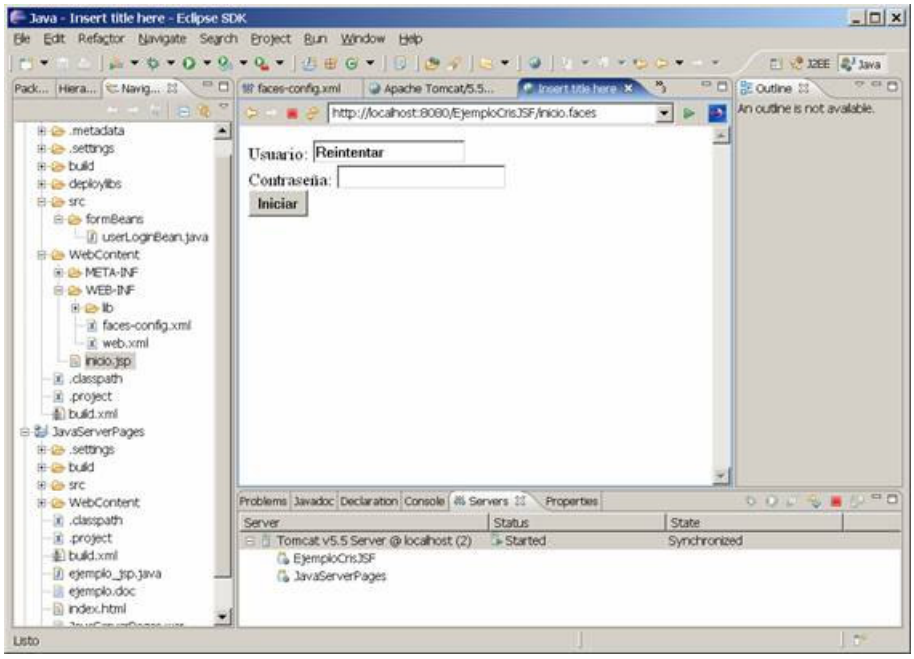


Ahora vamos a añadir interacción a la página. Para ello editamos la función de validación del bean, dejándola de este modo:

```
public String validarUsuario() {
```

```
            if (username.equals("test"))
                  return "test";
            else {
                  username = "Reintentar";
                  return "error";
            }
      }
```

Y al ejecutar de nuevo la página de inicio obtenemos:



Añadiendo navegación entre páginas

Ahora vamos a modificar la página y vamos a añadir una página principal, en la que suponemos que ya se ha validado el usuario. Para ello copiamos y modificamos el archivo inicio.jsp en otro nuevo testInicio.jsp como sigue:

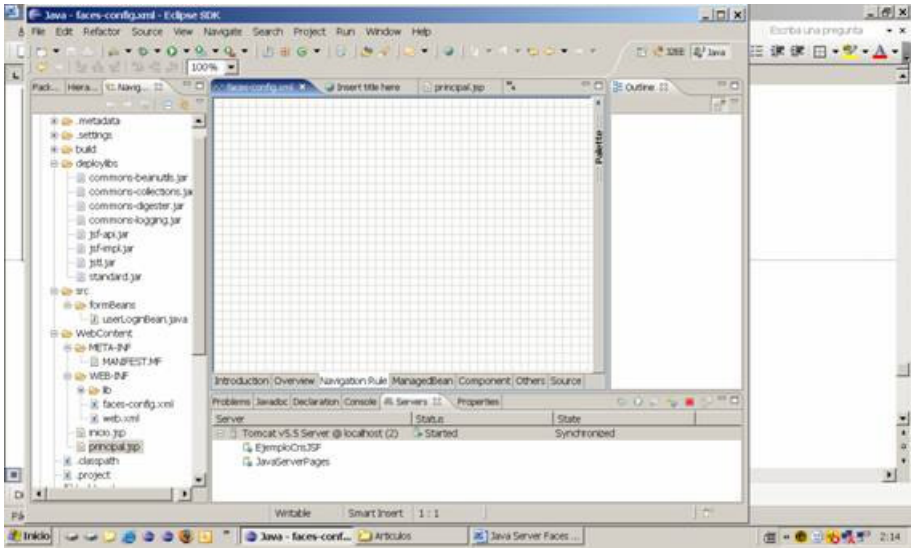
testInicio.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>Insert title here</title>
</head>
<body>
Usuario: <c:out value="${userLoginBean.username}"></c:out>

</body>
</html>
```

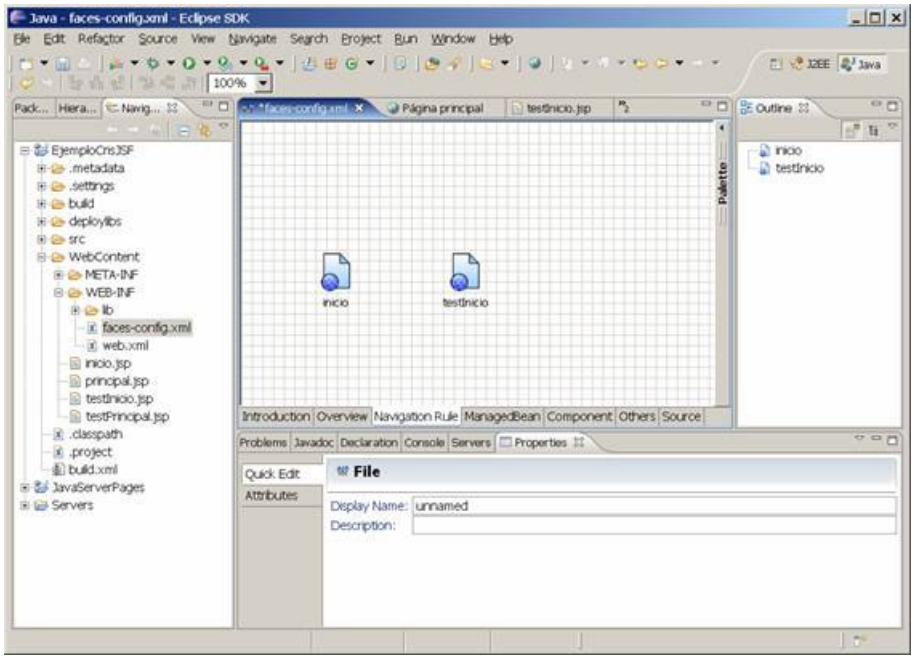
Es una pantalla de prueba muy sosa, pues lo único que queremos ver es que la aplicación navega. Ahora hay que añadir la regla de navegación. Una regla de navegación nos lleva desde una página jsp a otra al realizarse una determinada acción, y según un cierto resultado de la acción. El resultado de la acción es la cadena que devuelve la acción al ejecutarse. Nuestra función de validación del usuario devuelve la cadena “OK” si el usuario y la contraseña coinciden con los prefijados (la validación más sencilla posible).

El método de navegación es muy simple. Cuando el usuario pulsa el botón de inicio, se envía el formulario y el servlet de JSF llama a la función validarUsuaro del bean userLoginBean, que devuelve la cadena “test” si el usuario es “test”.

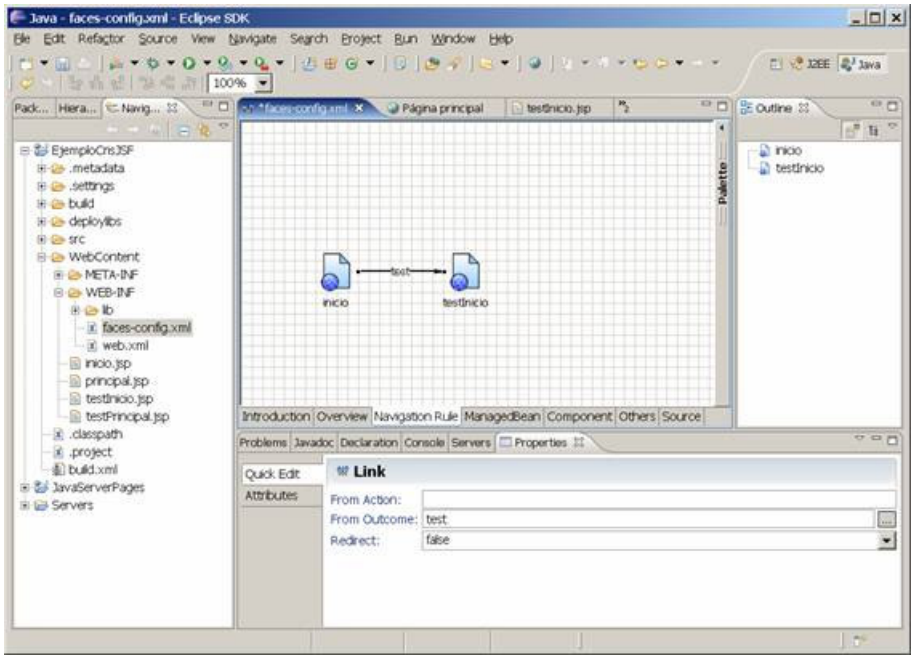
La regla de navegación la vamos a añadir con el plugin de Eclipse para modificar el faces-config.xml, en la pestaña “Navigation rule”



En principio partimos de una rejilla en blanco, ya que no tenemos definidas reglas de navegación. Arrastramos a la rejilla las dos páginas.



Aparecen los iconos de las páginas, y la barra de herramientas está oculta a la derecha de la cuadrícula, que se mostrará al pinchar sobre la etiqueta “Palette”. Pulsamos sobre el botón “Link” de la paleta de herramientas y pulsamos primero sobre el icono de la página de inicio y luego sobre el icono de la página de prueba de inicio.



Al añadir el enlace nos sale la flecha. Para modificar las propiedades del enlace pulsamos el botón “Select” de la paleta. Luego pinchamos en la línea del enlace y con el botón derecho seleccionamos “Show view…\properties”. Debajo nos aparece la ventana de propiedades del enlace. Añadimos el valor “OK” a la propiedad “From-outcome” y guardamos los cambios. Ahora nos aparece la etiqueta “OK” sobre la flecha del enlace.

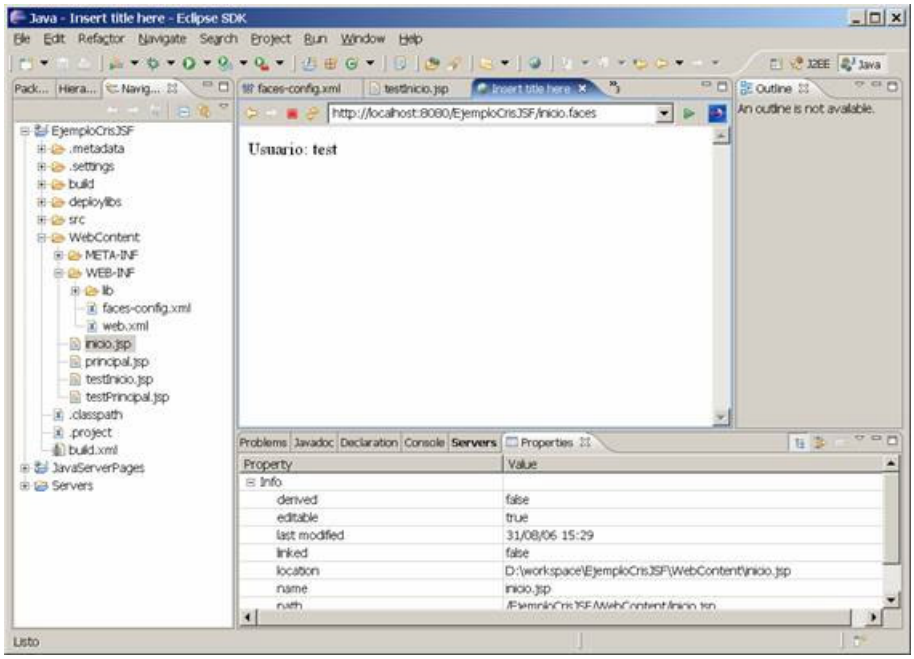
Podemos ver cómo se ha modificado el fichero de configuración faces-config.xml en la pestaña “Source” del plugin JSF. En nuestro caso nos queda el siguiente fichero de configuración:

Faces-config.xml

```
<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
      <managed-bean>
            <description>
            Guarda la información de usuario y contraseña para el
inicio de sesión y realiza la validación.x</description>
            <managed-bean-name>
            userLoginBean</managed-bean-name>
            <managed-bean-class>
            formBeans.userLoginBean</managed-bean-class>
            <managed-bean-scope>
            request</managed-bean-scope>
      </managed-bean>
      <navigation-rule>
            <display-name>
            inicio</display-name>
            <from-view-id>
            /inicio.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>
                  ok</from-outcome>
                  <to-view-id>
                  /testInicio.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>


</faces-config>
```

Vemos que la regla de navegación nos lleva de la página de inicio a la principal si el resultado de la acción (no hemos especificado el nombre de la acción pero se puede hacer en la propiedad From-action) tiene el valor “OK”.

Probamos la aplicación con la nueva regla, poniendo el usuario y contraseña correcto. Nos aparece la siguiente página tras pulsar el botón “Inicio”



Como vemos no nos muestra usuario.

Cambiamos en la pestaña “Managed beans” el tipo de ámbito del bean userLoginBean de “request” a “session” y guardamos los cambios. Ejecutamos de nuevo la aplicación y vemos el resultado:

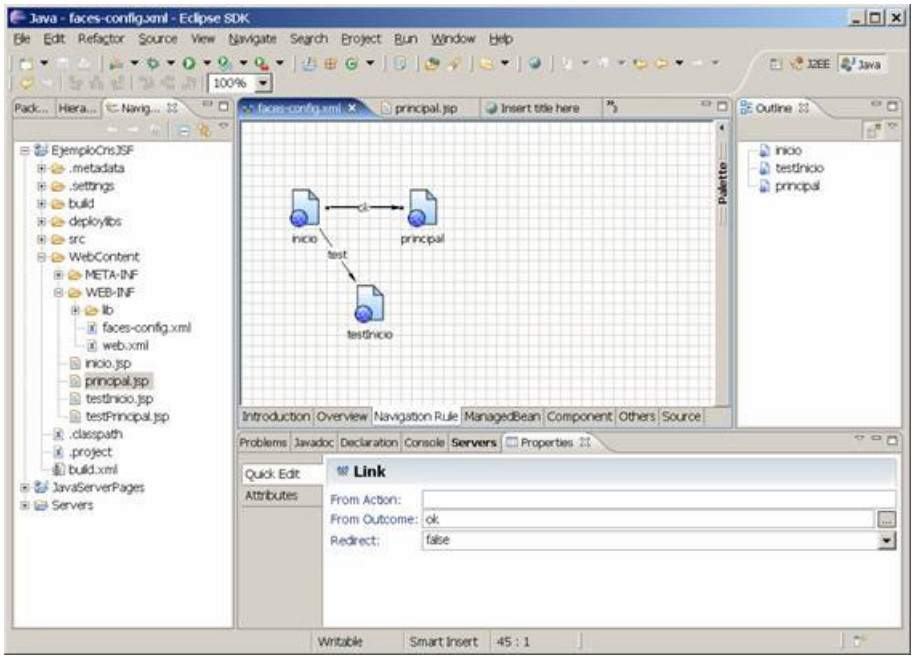
Añadiendo navegación entre formularios

Vamos a modificar un poco nuestras páginas. Añadimos una página principal como sigue:

principal.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>Página principal</title>
</head>
<body>
Bienvenido, <c:out value="${userLoginBean.username}" />
<f:view>
<h:form>
      <h:commandButton action="buscar"
value="Buscar"></h:commandButton>
      <h:commandButton action="mostrar"
value="Mostrar"></h:commandButton>


      <br />
      Este es el texto del formulario

</h:form>
</f:view>

</body>
</html>
```

Y una regla de navegación con nuestro editor visual:



Lo que nos deja un fichero de configuración como sigue:

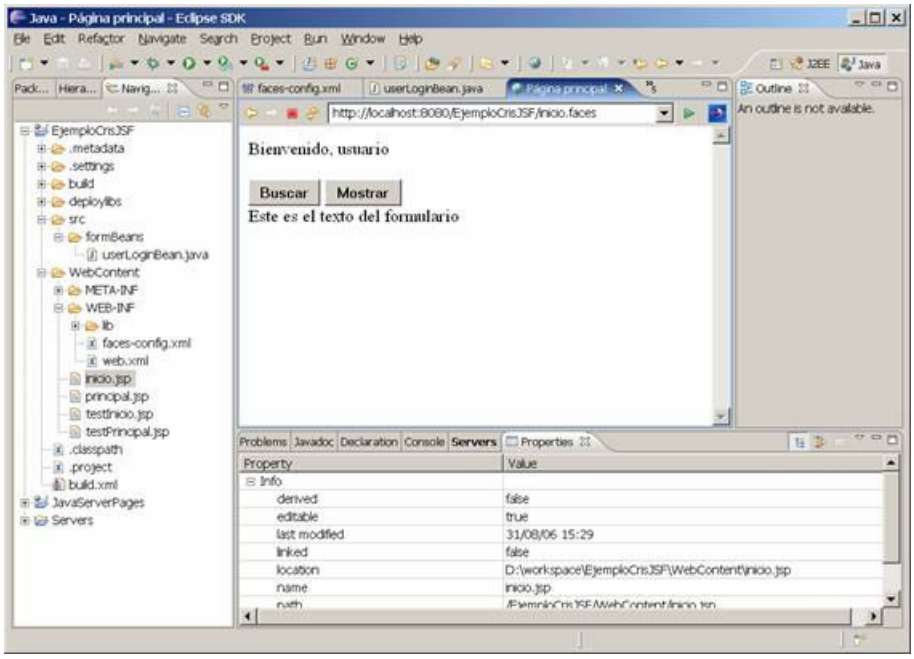
faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
      <managed-bean>
            <description>
            Guarda la información de usuario y contraseña para el
inicio de sesión y realiza la validación.x</description>
            <managed-bean-name>
            userLoginBean</managed-bean-name>
            <managed-bean-class>
            formBeans.userLoginBean</managed-bean-class>
            <managed-bean-scope>
            session</managed-bean-scope>
      </managed-bean>
      <navigation-rule>
            <display-name>
            inicio</display-name>
            <from-view-id>
            /inicio.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>
                  test</from-outcome>
                  <to-view-id>
                  /testInicio.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <display-name>
            inicio</display-name>
            <from-view-id>
            /inicio.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>
                  ok</from-outcome>
                  <to-view-id>
                  /principal.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>


</faces-config>
```

Modificamos la función de validación:

```
      public String validarUsuario() {
            if (username.equals("usuario") &&
password.equals("sesamo"))
                  return "ok";
            else if (username.equals("test"))
                  return "test";
            else {
                  username = "Reintentar";
                  return "badUsername";
            }
      }
```

Y lo ejecutamos, poniendo como usuario “usuario” y contraseña “sesamo”. El resultado es de esperar…



El formulario que se muestra tiene algo más de cosmética. Lo más interesante es ver cómo accedemos a los valores de los beans entre los diferentes formularios.

Acciones inmediatas

Como hemos visto, el flujo de navegación entre páginas JSF se produce al enviar el formulario. En nuestra pantalla de login, cuando el usuario pulsa el botón “Iniciar”, se produce la navegación a la página principal o se muestra una página de error.

El mecanismo de navegación por tanto se desencadena al enviar el formulario. En ocasiones necesitaremos modificar el formulario sin enviarlo, ya que puede que esté incompleto. Algunos ejemplos sencillos de esta situación serían:

- Habilitar partes ocultas del formulario. Por ejemplo, habilitar una caja de texto para introducir un dominio, si se necesitase además del nombre de usuario y contraseña.
- Modificar el formulario, por ejemplo modificando las opciones disponibles ya visibles.
- Realizar comprobaciones previas de alguno de los campos de texto introducidos. Por ejemplo, se podría comprobar si un cierto número de documento se ha introducido sin errores, ya que muchos documentos incluyen funciones de verificación en su numeración (el DNI español es un ejemplo).

Para verlo mejor con un ejemplo, vamos a añadir al formulario un campo dominio en el que el usuario podrá elegir tres posibles dominios, miempresa.com, yahoo.com y msn.com. Este campo estará oculto y se mostrará al pulsar un botón de “otro dominio…”

Modificamos nuestro bean de login:

userLoginBean.java

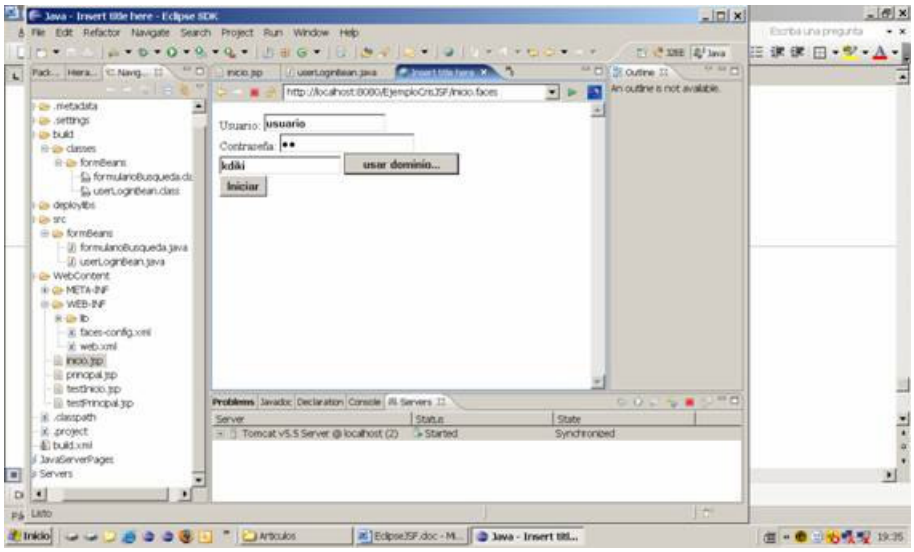
```
package formBeans;
import javax.faces.event.*;
```

```
public class userLoginBean {
	private String username;
	private String password;
	private boolean verDominioDeshabilitado = true;
	private String dominio;

	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;
	}


	public String getDominio() {
		return dominio;
	}
	public void setDominio(String dominio) {
		this.dominio = dominio;
	}
	public boolean isVerDominioDeshabilitado() {
		return verDominioDeshabilitado;
	}
	public void setVerDominioDeshabilitado(boolean
verDominioDeshabilitado) {
		this.verDominioDeshabilitado = verDominioDeshabilitado;
	}

	public String validarUsuario() {
		if (username.equals("usuario") && password.equals
("sesamo"))
			return "ok";
		else if (username.equals("test"))
			return "test";
		else {
			username = "Reintentar";
			return "badUsername";
		}
	}
	public void habilitarDominio(ActionEvent event) {
		verDominioDeshabilitado = false;
	}
}
```

Inicio.jsp

```
<title>Insert title here</title>
</head>
<body>
<f:view>
<h:form>
   Usuario:
      <h:inputText value="#{userLoginBean.username}" />
      <br/>
      Contraseña:
      <h:inputSecret value="#{userLoginBean.password}" />
      <br/>
      <h:inputText value="#{userLoginBean.dominio}" disabled="#
{userLoginBean.verDominioDeshabilitado}" />
      <h:commandButton actionListener="#
{userLoginBean.habilitarDominio}"
			immediate="true" value="usar dominio..."/>
		<br>
      <h:commandButton action="#{userLoginBean.validarUsuario}"
value="Iniciar" />

</h:form>
</f:view>

</body>
</html>
```

Ahora ejecutamos la página de inicio



Como vemos, cuando se ejecuta nuestra página aparece el campo dominio (el que está debajo de la contraseña) deshabilitado, no se puede escribir en él. Si pulsamos el botón "usar dominio…" se activa el campo y podremos escribir sobre él.

Añadiendo una lista dinámica a un combo

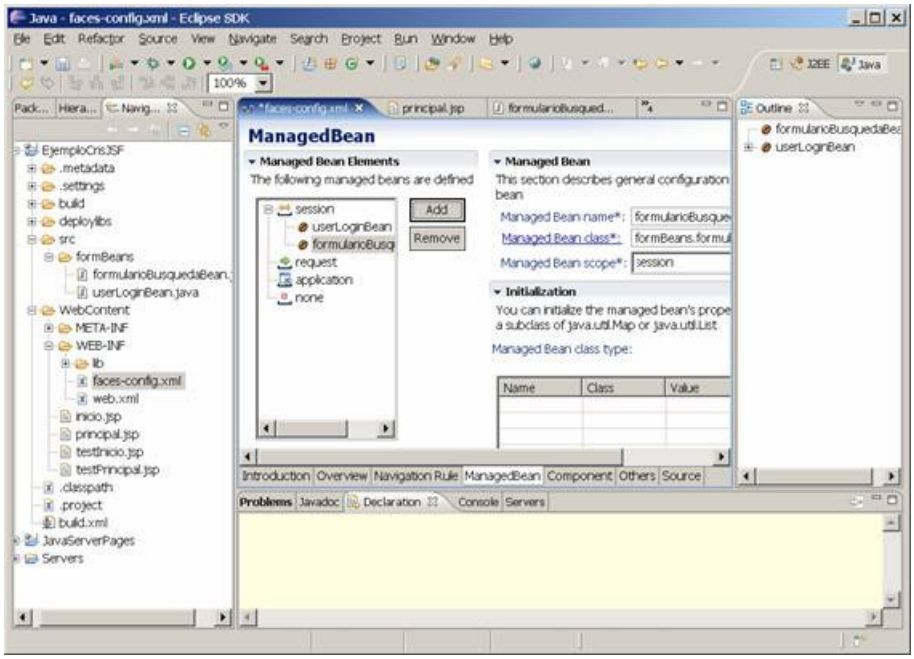
Una de las tareas más frecuentes en los formularios dinámicos es rellenar de manera dinámica los contenidos de las listas desplegables. Un caso típico suele ser los nombres de los países y ciudades. También es muy frecuente usar listas para seleccionar criterios en los campos de búsqueda.
Ahora vamos a añadir a nuestra página principal una lista cuyos valores se obtendrán dinámicamente. Para ello vamos a utilizar dos etiquetas de JSF, que son <h:selectOneMenu> y <f:selectItems>. La etiqueta selectOneMenu se encarga de pintar en el formulario una caja combo, es decir, una lista desplegable en la que se puede elegir un elemento. Hay más opciones disponibles, como son las etiquetas selectOneListbox (para crear listas de elementos), selectManyMenu, que son cajas combo de selección múltiple, y selectManyListbox, que son listas de selección múltiple.
Además, todos estos elementos tenemos que rellenarlos con elementos (algo hay que seleccionar). Para ello nos ayudaremos de la etiqueta f:selectItems. Esta etiqueta define un conjunto de elementos de tipo selectItem, que los recogerá de un bean, y los introduce en la etiqueta select que la contiene.
Veamos esto con un ejemplo. Modificamos el bean formularioBusquedas.java:

formularioBusquedaBean.java

```
package formBeans;
import javax.faces.model.*;


/**
 * @author cris
 *
 */
public class formularioBusquedaBean	{
	private String categoria;
	private String valor;

	/**
	 * por ahora hacemos una lista constante
	 * Luego se cambiará por una lista dinámica
	 */
	private SelectItem[] categorias = {
			new SelectItem("libros", "Libros"),
			new SelectItem("revistas", "Revistas"),
			new SelectItem("videos", "Videos")
	};

	public formularioBusquedaBean(){
		// aquí iría el código dinámico, quizás consulta a BD


	}

	public String getCategoria() {
```

```
            return categoria;
      }

      public void setCategoria(String categoria) {
            this.categoria = categoria;
      }

      public String getValor() {
            return valor;
      }

      public void setValor(String valor) {
            this.valor = valor;
      }

      public SelectItem[] getCategorias() {
            return categorias;
      }

      public void setCategorias(SelectItem[] categorias) {
            this.categorias = categorias;
      }


}
```

Añadimos el bean a los Managed Beans



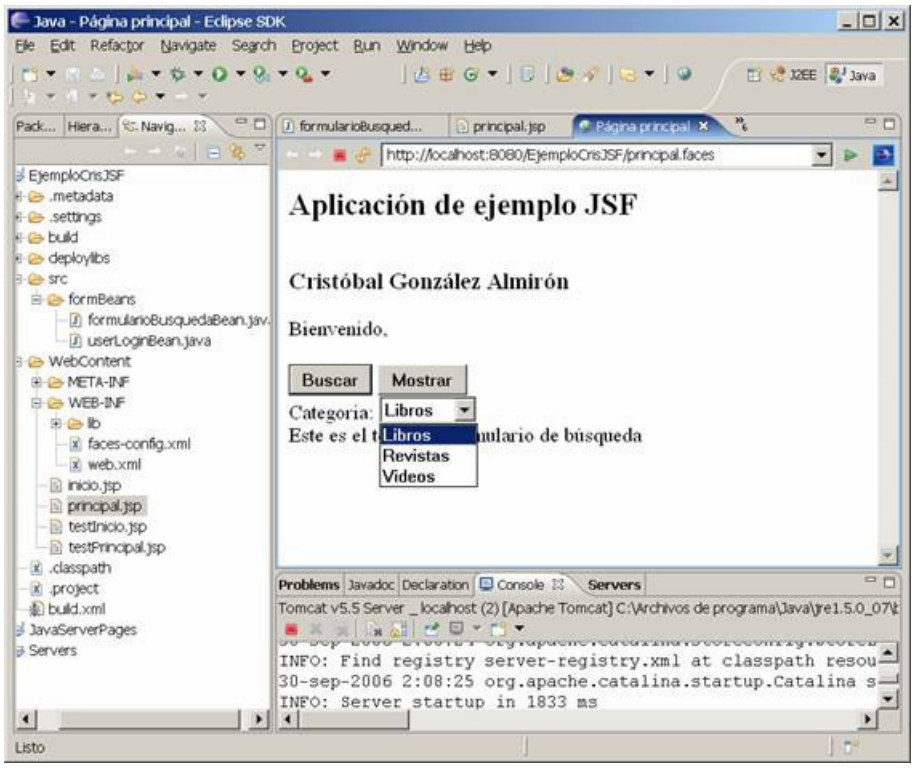
Y ahora modificamos la página del formulario de búsqueda

Principal.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

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

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>Página principal</title>
</head>
<body>
<h2>Aplicación de ejemplo JSF </h2><br>
<h3>Cristóbal González Almirón</h3>
Bienvenido, <c:out value="${userLoginBean.username}" />
<f:view>
<h:form>
	<h:commandButton action="buscar"
value="Buscar"></h:commandButton>
	<h:commandButton action="mostrar"
value="Mostrar"></h:commandButton>

	<br>
	Categoria:
	<h:selectOneMenu value="#{formularioBusquedaBean.categoria}">
		<f:selectItems value="#
{formularioBusquedaBean.categorias}" />
	</h:selectOneMenu>
	<br>
	Este es el texto del formulario de búsqueda

</h:form>
</f:view>

</body>
</html>
```

Al ejecutar la página vemos como el combo de elección aparece relleno;



Resumiendo

Según lo visto en este tutorial, el framework de JSF nos permite:

- Crear formularios web dinámicos, en los que el contenido del formulario se inserta automáticamente en un bean.
- Controlar el flujo de la aplicación de manera centralizada, mediante el uso de reglas de navegación, que se evalúan al enviar el formulario.
- Realizar cálculos y modificar los elementos del formulario dinámicamente, antes de que el usuario envie el formulario.

Todo esto vemos que es sencillo ya que el entorno JSF nos proporciona un lenguaje de expresiones que facilita la comunicación entre las páginas JSP y los beans, y un fichero de configuración que gestiona de forma centralizada la definición de beans y las reglas de navegación.

Y lo siguiente

Una vez que dominamos la comunicación entre el formulario y los bean, el control de la navegación y las acciones inmediatas, lo siguiente por lo que podemos avanzar es:

- Explorar el resto de las etiquetas que nos proporciona la biblioteca JSF, como son las
- Definir reglas de validación, mediante las etiquetas de validación que incluye JSF.
- Podemos utilizar controles JSF HTML más complejos, como los que nos proporciona la biblioteca MyFaces, que incluye componentes avanzados, calendarios, etc. que nos simplificarán el desarrollo de vistosas páginas web.
- Y si somos más atrevidos, podemos mezclas JSF con AJAX o con Struts. Pero eso ya es de nota.

Conclusión

El framework JSF nos permitirá crear aplicaciones Web siguiendo el patrón MVC de tipo 2 (modelo vista controlador con servlet controlador único), de manera sencilla y flexible. Podremos dotar de manera sencilla a nuestros formularios de capacidades dinámicas que facilitarán la usabilidad de nuestras páginas, dándole un aspecto más profesional, sin complicar en exceso el código de nuestra aplicación.
Además, con el plugin de JSF para Eclipse, integraremos de manera rápida el desarrollo de aplicaciones Web con JSF dentro de nuestro entorno de desarrollo favorito.

En Autentia sabemos que es complicado poner en marcha un proyecto Web. Por ello, si necesitas que te orientemos en la creación de un nuevo proyecto, o incluso si quieres que te lo pongamos en marcha en poco tiempo, aprovechando nuestra experiencia y tus recursos, no dudes en contactar con nosotros, y verás que tus desarrollos dejan de ser un problema para ser una herramienta de trabajo más.

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
JSF en Java Studio Creator 2	En este tutorial os mostramos como realizar una aplicación JSF utilizando la herramienta Java Studio Creator en su segunda versión
Construir un Servidor Web en Java	En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc.
Aplicaciones con JSPs	Os mostramos como construir una aplicación con JSP que acceda a MySQL
Optimización Java con Eclipse Profiler Plugin	Alejandro Pérez nos enseña como analizar el rendimiento de nuestras aplicaciones con Eclipse Profiler Plugin.
Aplicación de Patrones de Diseño en Java	En este tutorial os mostramos como las técnicas avanzadas de diseño (como patrones de diseño) contribuyen a la contrucción de aplicaciones profesionales en Java.
Probando entornos para JSF	En este tutorial os mostramos con ejemplos como utliizar dos conocidos entornos de desarrollo para JSF: Exadel Studio y Sun Studio Creator
Introducción al Java Web Start	Os mostramos como podéis distribuir aplicaciones de consola de un modo sencillo a través de Java Web Start. Es el mismo principio de los applets aplicado a aplicaciones.
JSF y NetBeans 5.5	Os mostramos como dar vuestros primeros pasos utilizando Java Server Faces (JSF) con ayuda del conocido entorno de desarrollo NetBeans
JSP´s y Modelo-Vista-Controlador	En este tutorial os enseñamos como crear un JSP, su relación con los servlets y como crear un ejemplo MVC en Tomcat
Utilizando JSTL en JSF	Os mostramos como utilizar la librería estandar de etiquetas en JSF, implementando una sencilla aplicación web