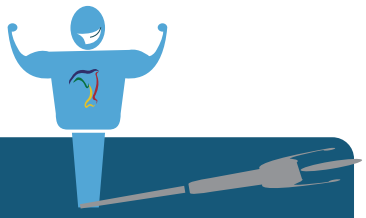
autentia

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...

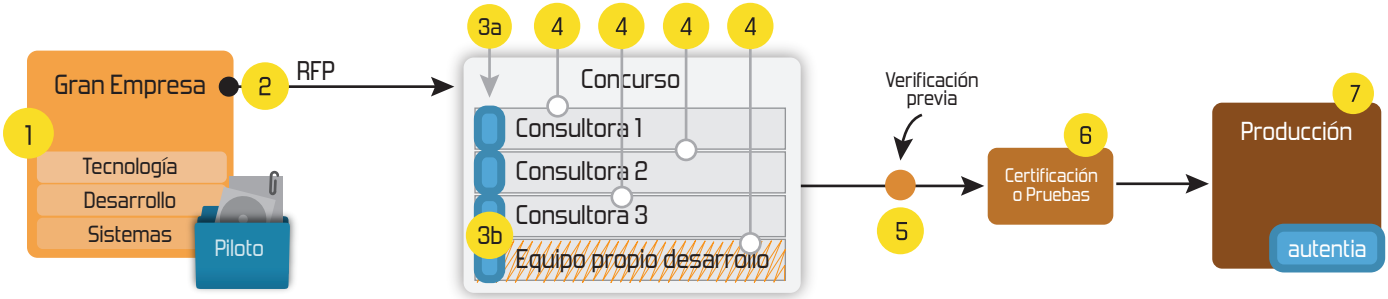
1. Desarrollo de componentes y proyectos a medida



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 (au**Te**ia).

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/

Fecha de creación del tutorial: 2007-11-01

Como autocompletar Facelets en Eclipse Europa

Creación: 01-11-2007

Índice de contenidos

1. Introducción

Los que usamos Facelets como alternativa a las JSP para renderizar JSF, a menudo sufrimos el escaso soporte que proporcionan los entornos de desarrollo.

En este tutorial vamos a ver como conseguir que en Eclipse funcione el autocompletar (Ctrl + espacio) cuando estamos

editando nuestras páginas de Facelets. Eclipse no soporta directamente Facelets, pero podemos aprovecharnos de las funcionalidades de WTP (Web Tool Project) para conseguir que funcione el autocompletar.

Esto va a suponer una gran *ayuda* ya que no tendremos que sabernos de memoria todos los posibles componentes o etiquetas con sus correspondientes atributos, y una gran *comodidad* porque tendremos que escribir menos ;)

Hay que recordar que todo lo que vamos a hacer en este tutorial sólo es necesario para que Eclipse sea capaz de autocompletar cuando escribimos nuestras páginas de Facelets; pero que en ningún caso es necesario desde el punto de vista de Facelets. Facelets funciona igual sin que hiciéramos nada de esto.

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Asus G1 (Core 2 Duo a 2.1 GHz, 2048 MB RAM, 120 GB HD).
- Sistema Operativo: GNU / Linux, Debian (unstable), Kernel 2.6.22, KDE 3.5
- JDK 1.5.0_13 (instalada con el paquete sun-java5-jdk de
- Eclipse Europa 3.3, con soporte para desarrollo JEE (WTP 2.0)
- JSF RI 1.2_5
- Facelets 1.1.14

3. Distinguiendo tipos de páginas

Antes de nada vamos a fijarnos en un pequeño ejemplo. Supongamos que tenemos las siguientes páginas:

defaultLayout.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

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

<head>
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<title><ui:insert name="title" /> - autentia-web</title>
<link rel="stylesheet" type="text/css" media="all"
   href="#{facesContext.externalContext.requestContextPath}/css/style.css" />
</head>

<body>

<div id="layout">
   <ui:insert name="header">
      <ui:include src="header.xhtml" />
   </ui:insert>

   <div id="content">
      <h1><ui:insert name="title" /></h1>
      <ui:insert name="content">
         <ui:include src="content.xhtml" />
      </ui:insert>
   </div>
</div>

</body>
</html>
```

Esta página es la que define la composición de la página. Es la plantilla que define como se va a distribuir los diferentes elementos. A través de `<ui:insert>` definimos que puntos pueden ser "redefinidos" y cuales serán sus valores por defecto.

Este fichero empieza con la típica cabecera de xml, luego el `DOCTYPE` y luego la etiqueta `<html>`. Esto es fundamental para que el navegador que recibe la página sepa interpretarla correctamente.

Básicamente lo que hace Facelets es volcar este fichero a la salida, sustituyendo los elementos `<ui:insert>` por los valores que correspondan.

Llamaremos *plantilla* a este tipo de ficheros.

header.xhtml

```
<div id="header">
   <span id="title">Aplicación para el tutorial de <a href="https://facelets.dev.java.net" target="_blank">Facelets</a>
```

```
 de <a href="http://www.adictosaltrabajo.com" target="_blank">www.adictosaltrabajo.com</a>
 y <a href="http://www.autentia.com" target="_blank">Autentia</a> </span>
</div>
```

Este fichero supone el valor por defecto para el `<ui:insert name="header">` del fichero anterior.

Se puede ver que aquí no definimos cabecera xml ni `DOCTYPE` ni etiqueta `<html>`. Esto es así porque el contenido de este fichero se "incrusta" directamente donde está el elemento `<ui:include src="header.xhml" />`; por lo que si tuviéramos la cabecera xml o el `DOCTYPE` o la etiqueta `<html>` el resultado de "incrustarlo" en el fichero anterior daría como resultado un documento HTML incorrecto.

Como no definimos la cabecera xml, habrá que tener especial cuidado con el encoding (codificación de caracteres) con el que guardamos estos ficheros.

Llamaremos a estos ficheros *ficheros de inclusión*.

holaMundo.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:t="http://myfaces.apache.org/tomahawk">

<body>

This text above will not be displayed.

<ui:composition template="defaultLayout.xhtml">
   <ui:define name="title">Ejemplo de plantilla</ui:define>
   <ui:define name="content">
      <h:outputText value="Hola Mundo !!!" />
   </ui:define>
</ui:composition>

This text below will also not be displayed.

</body>
</html>
```

Este fichero usa la *plantilla* defaultLayout.xhtml y redefine algunos de sus elementos con `<ui:define>`.

Todo el texto que está fuera de `<ui:composition>` no viaja nunca al cliente. Por esto aunque el fichero comienza con la cabecera xml, el `DOCTYPE` y la etiqueta `<html>`, esto no va a aparecer nunca en el documento HTML resultado de aplicar la plantilla. Simplemente lo ponemos por conveniencia, para que la página se muestre correctamente en el editor que estemos usando.

Llamaremos a este tipo de fichero *cliente de plantilla*.

4. Consiguiendo que funcione el autocompletar

Para que funcione el autocompletar vamos a convertir las páginas xhtml en jpsx.

Estos dos tipos de páginas son, en definitiva, documentos XML, así que podemos usar cualquiera de los dos con Facelets, ya que este, como dice en la sección 1.6.1 de la documentación, lo único que necesita es un XML válido.

Desde el punto de vista de Eclipse, los xhtml son interpretados como documentos HTML, y por lo tanto sólo es capaz de autocompletar con las etiquetas de HTML. Sin embargo los jspx son interpretados como páginas JSP, y si tenemos el correspondiente fichero tld en WEB-INF, Eclipse será capaz de autocompletar con las etiquetas definidas en este fichero tld.

Veamos paso a paso como conseguimos está transformación:

4.1. Transformando el *cliente de plantilla* (holaMundo.xhtml)

Renombramos holaMundo.xhtml por holaMundo.jspx

Modificamos el contenido, de manera que el fichero quede de la siguiente manera:

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

<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:t="http://myfaces.apache.org/tomahawk">

   This text above will not be displayed.

   <ui:composition template="defaultLayout.xhtml">
      <ui:define name="title">Ejemplo de plantilla</ui:define>
```

```
    <ui:define name="content">
      <h:outputText value="Hola Mundo !!!" />
    </ui:define>
  </ui:composition>

  This text below will also not be displayed.

</jsp:root>
```

Nótese como hemos quitado el `DOCTYPE` y hemos sustituido la etiqueta `<html>` por `<jsp:root>`, donde igual que en la etiqueta `<html>` definimos los namespaces que vamos a usar.

Muy importante que no se nos olvide poner el primer namespace `xmlns:jsp="http://java.sun.com/JSP/Page"`.

Con este cambio Eclipse ya es capaz de autocompletar las etiquetas h: y f: del estándar de JSF.

4.2. Transformando la *plantilla* (defaultLayout.xhtml) y el *fichero de inclusión* (header.xhtml)

Estos tipos de documentos no los vamos a poder transformar en jspx y por lo tanto no vamos a conseguir la funcionalidad de autocompletar del Eclipse.

Esto es porque, como ya he dicho en el apartado 3, el contenido de estos ficheros sí viaja al cliente directamente, por lo que si cambiáramos en el caso de la *plantilla* la etiqueta `<html>` por `<jsp:root>`, o añadiéramos esta en el caso del *fichero de inclusión*. El documento HTML que viajaría al cliente no sería correcto y el navegador no sabría interpretarlo.

De todas formas esto sólo va a ser un pequeño inconveniente ya que lo normal en una aplicación es que tengamos unas pocas *plantillas* o *ficheros de inclusión*; y por el contrario tengamos muchos *clientes de plantilla*. Así que conseguimos la capacidad de autocompletar en los ficheros que realmente nos interesa.

4.3. Añadiendo los ficheros tld

Ya hemos dicho que, con el cambio de xhtml a jspx, Eclipse sabe autocompletar las etiquetas h: y f:. Pero ¿que pasa con el resto de etiquetas? ¿con las etiquetas ui: del propio Facelets, o con las etiquetas t: de Tomahawk?

Para que Eclipse sea capaz de autocompletar estas etiquetas necesitamos copiar sus ficheros tld en el directorio WEB-INF. Ojo, esto sólo es necesario para Eclipse, a Faceletes no el hace falta para funcionar correctamente.

El tld de Faceletes lo podemos descargar de https://facelets.dev.java.net/files/documents/3448/21641/jsf-ui.tld

El tld de Tomahawk lo podemos sacar del propio jar de Tomahawk (tomahawk-1.1.6.jar). Lo encontraremos en META-INF/tomahawk.tld

Esta técnica sería valida para cualquier otro espacio de nombres de etiquetas. Por ejemplo si quisiéramos trabajar con ICEfaces (http://www.icefaces.org), bastaría con copiar el tld correspondiente en WEB-INF, y por supuesto definir el espacio de nombres en la etiqueta `<jsp:root>`.

5. Reconfigurando Facelets

Hemos cambiado la extensión de los *clientes de plantilla* de xhtml a jspx. Esto a Facelets no le va a sentar nada bien. Por lo que será necesario editar el fichero WEB-INF/web.xml y cambiar:

```
  <context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.xhtml</param-value>
  </context-param>
```

por:

```
  <context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.jspx</param-value>
  </context-param>
```

Es decir, le estamos diciendo a JSF que cundo llegue una petición de JSF busque los ficheros con extensión .jspx, en lugar de .xhtml, que es lo habitual cuando se trabaja con Facelets.

¿Hay algún problema en tener los *clientes de plantilla* con extensión jspx y las *plantillas* y los *ficheros de inclusión* con extensión xhtml?

No, en principio no va ha haber ningún problema ya que nunca haremos una petición directamente a una *plantilla* o a un *fichero de inclusión*. Es decir, todas las peticiones JSF siempre estarán "invocando" un *cliente de plantilla*.

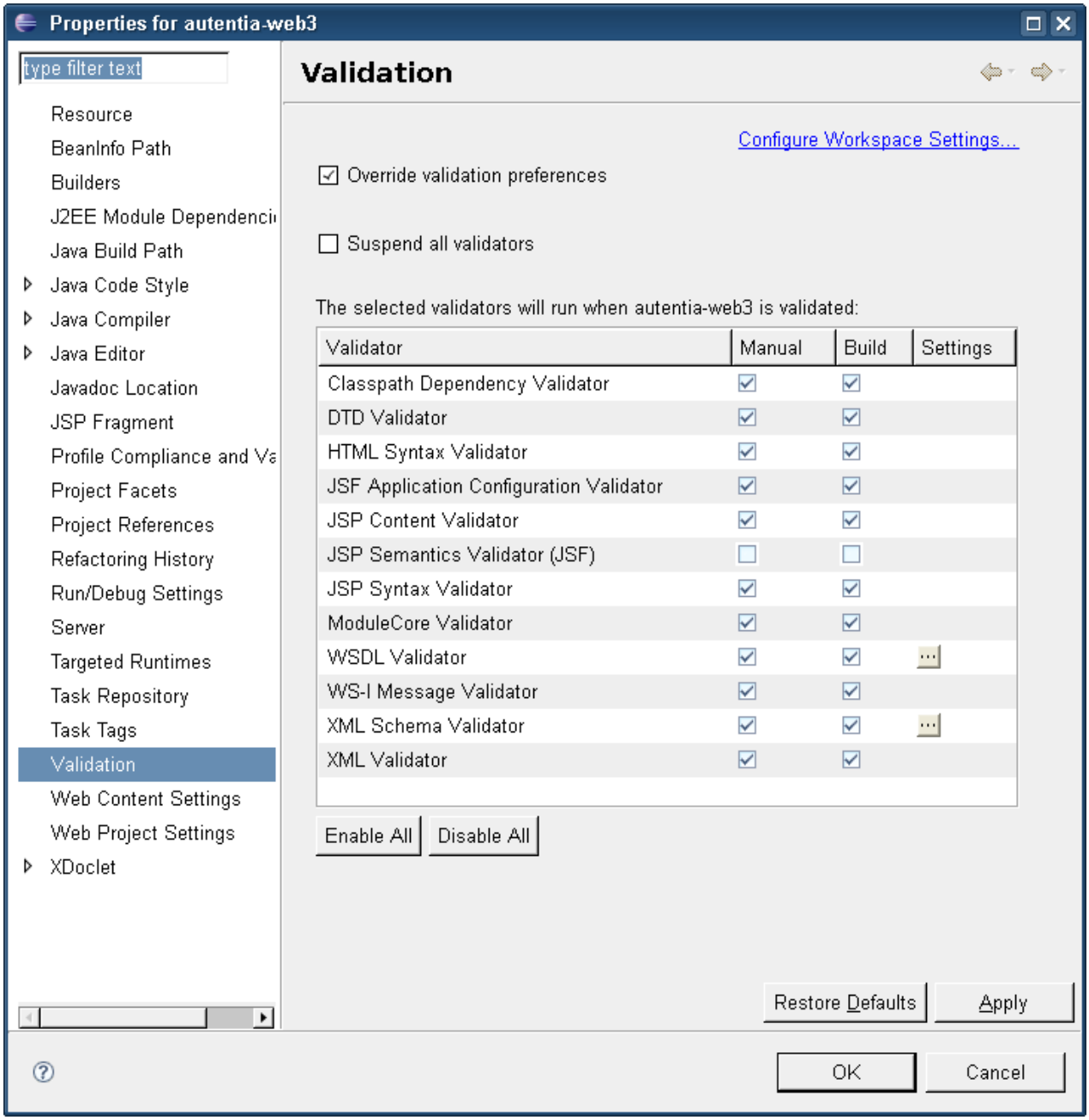
6. Validaciones de Eclipse

Si tenéis activas las validaciones de Eclipse (suele ser lo habitual) veréis como Eclipse empieza a subrayaros en amarillo (son warnings) ciertas partes de vuestra jspx.

Esto es porque realmente está intentando validar que se trata de una JSP de JSF válida. Cuando en realidad no es realmente una página JSP sino que se trata de un página que va a interpretar Facelets.

Para quitar estos warnings lo que podemos hacer es, sobre el *nombre del proyecto* ---> *botón derecho del ratón* ---> *Properties* ---> *Validation*

Y ahora activar: *Override validation preferences*, y desactivar: *JSP Semantics Validator* (JSF)



7. Conclusiones

Con este tutorial hemos visto como con unos sencillos pasos podemos sacarle mucho más partido a nuestro entorno de desarrollo Eclipse. De esta forma vamos a ser mucho más productivos y vamos a cometer menos errores tontos por escribir mal alguna letra, y vamos a poder dedicar nuestra memoria a cosas mucho más interesantes que aprendernos una lista interminable de etiquetas y atributos.

En Autentia (www.autentia.com) impartimos habitualmente cursos sobre Eclipse, Netbeans, ... ya que pensamos es fundamental dominar nuestros entornos de trabajo para vivir mucho mejor (si ahorro tiempo en el trabajo podre pasar más tiempo fuera de él ;)

8. Referencias

Documentación de Facelets: https://facelets.dev.java.net/nonav/docs/dev/docbook.html

Artículo sobre como empezar con Facelets:
http://www.thearcmind.com/confluence/display/SHJFT/Getting+started+with+JSF,+Facelets,+Eclipse+WTP+and+Tomcat

9. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Autentia S.L. Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..
y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	**Descripción**
Callisto, nunca antes resultó tan fácil desarrollar con Eclipse	En este tutorial os enseñamos a instalar y utilizar Callipso: una aplicación que permite instalar de manera fácil y cómoda plugins y sus dependencias en Eclipse
Manejo de Repositorios CVS desde Eclipse	En este tutorial os enseñamos a manejar el repositorio CVS desde la plataforma Eclipse
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: aplicación multimódulo	En este artículo se va a abordar el desarrollo de una aplicación Myfaces JSF con Maven que sea multimódulo.
Plugin PHPeclipse en Debian	Alejandro Pérez nos muestra como instalar el plugin de Eclipse: PHPeclipse, en GNU / Linux (Debian)

Java Server Faces con Eclipse	Este tutorial es un completo estudio de JSF, una moderna tecnología que nos permite desarrollar aplicaciones empresariales robustas.
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: pruebas con Jetty y Tomcat	Este es el tercer tutorial de la "saga" de Maven, JSF y Eclipse, donde se va a realizar las pruebas de la aplicación sobre dos servidores web diferentes: el servidor Jetty, integrado en Maven, y el servidor Tomcat, que lo integraremos con Eclipse.
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: Hibernate (segunda parte)	En este artículo se va a continuar con el desarrollo de la aplicación Myfaces JSF con Maven multimódulo que comenzamos en un tutorial anterior. Además también se tratará de la integración de Hibernate con las aplicaciones.
PMD, Eclipse y NetBeans	Tutorial que describe la instalación y uso de PMD en los entornos de desarrollo Eclipse y NetBeans
Optimización Java con Eclipse Profiler Plugin	Alejandro Pérez nos enseña como analizar el rendimiento de nuestras aplicaciones con Eclipse Profiler Plugin.
Instalación de Together para Eclipse	Os mostramos como instalar la versión de evaluación de Together sobre Eclipse. Estas dos herramientas constituyen una excelente base para el trabajo profesional y serio en entornos Java.