



Desarrollo de Aplicaciones Informáticas

CICLO FORMATIVO DE GRADO SUPERIOR

FORMACIÓN PROFESIONAL A DISTANCIA

Unidad

8

Programación WEB - J2EE Fundamentos

MÓDULO

Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE



FORMACIÓN PROFESIONAL

Principado de Asturias

Título del Ciclo: DESARROLLO DE APLICACIONES INFORMATICAS

Título del Módulo: DESARROLLO DE APLICACIONES EN ENTORNOS DE CUARTA GENERACIÓN Y CON HERRAMIENTAS CASE

Unidad 8: Programación WEB - J2EE Fundamentos

Dirección: Dirección General de Políticas Educativas, Ordenación Académica y Formación Profesional
Servicio de Formación Profesional Inicial y Aprendizaje Permanente

Dirección de la obra:

Alfonso Careaga Herrera
Antonio Reguera García
Arturo García Fernández
Ascensión Solís Fernández
Juan Carlos Quirós Quirós
Luís M^a Palacio Junquera
Yolanda Álvarez Granda

Coordinador de los contenidos:

Juan Manuel Fernández Gutiérrez

Autores:

Juan Manuel Fernández Gutiérrez

Colección:

Materiales didácticos de aula

Serie:

Formación profesional específica

Edita:

Consejería de Educación y Ciencia

ISBN:

978-84-692-3443-2

Deposito Legal:

AS-3564-2009

Copyright:

2009. Consejería de Educación y Ciencia

Esta publicación tiene fines exclusivamente educativos. Queda prohibida la venta, así como la reproducción total o parcial de sus contenidos sin autorización expresa de los autores y del Copyright

Introducción

JSP (Java Server Pages) es una tecnología que permite generar páginas dinámicas, en nuestro caso con formato HTML. Es un producto desarrollado por la compañía Sun Microsystems. La programación con JSP se basa en código Java además de utilizar otros recursos como las etiquetas.

En esta Unidad se describen y comentan los elementos necesarios de JSP para realizar páginas dinámicas.

Para desarrollar aplicaciones WEB es necesario tener instalado el Development Kit y el Runtime Environment de Java. Un servidor de aplicaciones, en nuestro caso el TOMCAT y una herramienta de desarrollo, en este caso utilizaremos el ECLIPSE. El software propuesto es de libre distribución y se puede bajar de la red.

Objetivos

- Conocer las características de las aplicaciones WEB
- Trabajar con la herramienta de desarrollo *Eclipse*
- Trabajar con los diferentes elementos del lenguaje JSP
- Saber instalar el software del entorno

Contenidos Generales

1. INTRODUCCIÓN AL DESARROLLO DE APLICACIONES.....	3
2. SERVIDORES WEB.....	5
3. APLICACIONES WEB.....	5
4. LENGUAJES DE PROGRAMACIÓN PARA LA WEB.....	6
5. TECNOLOGÍAS DE DESARROLLO.....	6
6. LENGUAJE JSP.....	7
7. ELEMENTOS DE UNA PÁGINA JSP.....	8
7.1. HTML y XHTML.....	9
7.2. Directivas JSP.....	9
7.3. Elementos script.....	12
7.3.1. Declaraciones.....	13
7.3.2. Scriptlets.....	14
7.3.3. Expresiones.....	15
7.4. Comentario JSP.....	16
7.5. Objetos predefinidos.....	16
7.6. Etiquetas estándar.....	24
8. ENTORNO DE DESARROLLO.....	28
8.1. JDK.....	28
8.2. TOMCAT.....	29
8.3. ECLIPSE.....	30
8.4. CONFIGURACIÓN.....	35

1. Introducción al Desarrollo de Aplicaciones

Tradicionalmente a los aplicativos no se les exigía mucho más que los requerimientos básicos, es decir que operasen según las especificaciones de los usuarios en cuanto a funcionalidad fiabilidad y seguridad. Hoy en día dado que las aplicaciones cada vez son más complejas se tiende a utilizar patrones para el desarrollo y que cumplan minimamente una serie de requerimientos como la de ser escalables, poder acceder a las bases de datos con distintas tecnologías, que implementen sistemas de seguridad y tener distintos interfaces de usuario: web, ventanas, acceso a través de diferentes dispositivos (PDA's , móviles, etc)

Para construir aplicaciones que puedan ser mantenidas fácilmente y puedan contener partes reutilizables, deben estar diseñadas siguiendo la arquitectura que fijan los patrones arquitectónicos Layers y Model View Controller (MVC)

El patrón arquitectónico Layers permite la estructuración de la aplicación, el software está estructurado en capas, permite ocultar las tecnologías que usa nuestro software, cuando hay un cambio de versión en una de las tecnologías o se reemplaza por otra se minimiza el impacto sobre las capas superiores. Establece una división clara de trabajo entre los miembros de un equipo y da soporte a la arquitectura MVC.

El patrón arquitectónico MVC establece una estructura lógica de la aplicación, separando el modelo (lógica de negocio) y la vista (interface gráfica). Tiene la ventaja de que el modelo es reutilizable con distintas vistas, se puede establecer una división clara de trabajo entre los miembros de un equipo con distintos niveles de especialización. Por último, la independencia de la vista permite distintas representaciones de la misma aplicación.

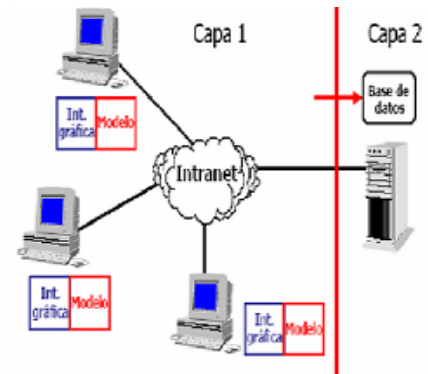
La vista (también llamada capa de presentación) es el interface de usuario. Captura y muestra la información al usuario, puede tener cambios, lo que provoca que se tengan que hacer retoques en su diseño. Se comunica únicamente con la capa de negocio.

En la lógica de negocio se establecen todas las reglas que deben cumplirse, se comunica con la capa de datos y de presentación, es independiente de ambas

La capa de datos es la encargada de acceder a los datos (bases de datos), no suele modificarse, recibe las solicitudes de la capa de negocio y le sirve la información pedida.

- **Arquitectura de dos capas**

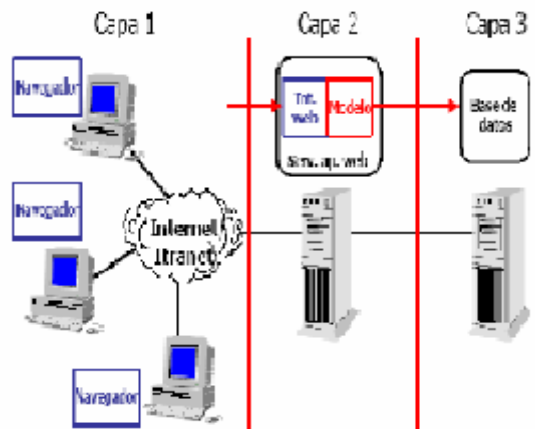
En este caso el servidor de la B.D está en otra máquina ejecutándose en los clientes la interface grafica y el acceso a datos. Este sistema provoca que los cambios en el modelo de datos o en la presentación obliga a recompilar la aplicación y la reinstalación en los clientes. Los cambios pueden venir dados por modificaciones en la lógica del modelo, variaciones en la base de datos o en los drivers.



- **Arquitectura de tres capas**

En este caso el interface de usuario se ubica en la primera capa (un navegador), luego se establece una capa intermedia con la lógica de negocio donde se encuentra el servidor de aplicaciones web y por último en la tercera capa se ubica la base de datos.

Con este modelo se pueden hacer cambios en el programa (lógica de negocio) sin tener que reinstalar los clientes, en ellos solo está el navegador.



2. Servidores WEB

Son los ordenadores en los que se alojan las aplicaciones web. En ellos está instalado un software que actúa como servidor y que responde a las peticiones de los navegadores. Algunos de los más utilizados son los siguientes:

- Internet Information Server (IIS). Para aplicaciones .NET, con páginas asp.
Es de Microsoft y no es gratuito.
- Apache. Gratuito, muy utilizado sobre UNIX y con páginas php.
- Tomcat. Contenedor de Servlets y de páginas JSP. Gratuito y perteneciente a Apache Software Foundation.
- JBoss. En la misma línea que Tomcat, pero mucho más completo ya que incluye soporte para EJB (Enterprise Java Beans).
- OC4J (Oracle Application Server Containers for J2EE). Oracle también intenta abrirse un hueco en el mercado de servidores J2EE.

3. Aplicaciones WEB

Las aplicaciones o portales WEB se componen de páginas que pueden ser estáticas o dinámicas

Páginas web estáticas

Al principio las páginas web eran estáticas. Cuando el cliente solicitaba una página web a un servidor, éste se la enviaba sin más. El navegador del cliente interpretaba la página y se la mostraba al usuario.

Páginas dinámicas

Las páginas dinámicas son ejecutadas en el servidor, el resultado de esta ejecución es una página estática que se envía al cliente.

Por ejemplo, el servidor puede acceder a una base de datos, extraer información, crear una página HTML con los datos obtenidos y enviársela al cliente.

4. Lenguajes de programación para la WEB

- HTML. Es el estándar de programación de páginas web. Permite indicar qué información es la que vamos a mostrar en la página por medio de “etiquetas”.
- XHTML. Surge al aplicar las normas de construcción de los documentos XML al HTML anterior. Por ejemplo, todas las etiquetas tienen que cerrarse, deben ir escritas en minúscula, etc.
- JavaScript, VBScript. Son lenguajes de script que permiten crear pequeños programas incrustados entre las etiquetas de las páginas HTML. Son ejecutados por el navegador del cliente cuando éste recibe la página.
- CGI, Perl, PHP, ASP y JSP, son lenguajes que permiten crear aplicaciones que se ejecutan en el servidor.
- Hojas de estilo (CSS – Cascade Style Sheets). Se utilizan para separar la forma del contenido, es decir, para cambiar colores, tamaños, posiciones, de todos los objetos que componen la página web.

5. Tecnologías de desarrollo

Plataforma J2EE (Java 2 Enterprise Edition).

J2EE (Java 2 Enterprise Edition) es una tecnología desarrollada por Sun Microsystems, es gratuita e independiente por lo que cuenta con una gran cantidad de desarrolladores. Consiste en un conjunto de especificaciones y prácticas coordinadas que juntas permiten soluciones para el desarrollo, despliegue, y gestión de aplicaciones multicapa centradas en servidor. Está basada en componentes. Se ejecuta sobre un servidor de aplicaciones, no fuerza a usar ningún producto específico.

Entre los componentes que constituyen la tecnología J2EE se encuentra JSP (Java Server Pages) que permite generar páginas dinámicas, en nuestro caso con formato HTML. Es un producto desarrollado por la compañía Sun. La programación con JSP se basa en código Java además de utilizar otros recursos como las etiquetas.

En esta Unidad se describen y comentan los elementos necesarios de JSP para realizar páginas dinámicas.

Para desarrollar aplicaciones WEB es necesario tener instalado el siguiente software:

- Development Kit y el Runtime Environment de Java.
- Un servidor de aplicaciones, en nuestro caso el TOMCAT
- Una herramienta de desarrollo, en este caso utilizaremos el ECLIPSE.

El software propuesto es de libre distribución y se puede bajar de la red.

Plataforma .NET.

Desarrollado por Microsoft. Conceptualmente es muy similar a J2EE. Las aplicaciones web sólo se pueden publicar con Internet Information Server de Microsoft (IIS).

6. Lenguaje JSP

JSP (Java Server Pages) es una especificación de Sun Microsystems, que permite generar contenido dinámico para web en forma de documento HTML, XML o de otro tipo. Básicamente consiste en combinar código HTML para generar la parte estática de la página con contenido dinámico generado a partir de marcas especiales. El contenido dinámico se obtiene, en esencia, gracias a la posibilidad de incrustar dentro de la página código Java de diferentes formas, su objetivo final es separar la interfaz (presentación visual) de la implementación.

El servidor de aplicaciones interpreta el código contenido en la página JSP para construir el código Java del Servlet a generar. Este servlet será el que genere el documento HTML que se presentará en la pantalla del navegador del usuario.

Los JSP's se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada JSP se ejecuta en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo (cargar programa + intérprete). Su persistencia le permite también hacer una serie de cosas de forma más eficiente, por ejemplo, conexión a bases de datos y manejo de sesiones.

También se pueden programar páginas servlet en vez de JSP, la diferencia fundamental es que en JSP se incluye código Java en HTML, en cambio con los servlets se incrusta HTML en código Java. En las páginas JSP el código Java se introduce encerrándolo entre etiquetas especiales `<%.....%>`, y en el caso de los servlets el código HTML se incluye a través de las sentencias `println`.

Muchas veces los programadores deciden entre utilizar servlets o JSP en función del tipo de página, dependiendo de la carga de etiquetas XHTML o de código java que haya que utilizar. Se suelen utilizar servlets si no hay código HTML o su presencia es mínima, en caso contrario se utiliza JSP. Además, separando el formato del contenido podemos poner diferentes personas en diferentes tareas: nuestros expertos en diseño de páginas Web pueden construir el HTML, dejando espacio para que nuestros programadores de java inserten el contenido dinámico.

7. Elementos de una página JSP

- Código HTML
- Directivas.
- Scriptlets
- Declaraciones
- Expresiones
- Comentarios

7.1. HTML y XHTML

HTML son las iniciales de *Hiper Text Markup Lenguaje*. Es un lenguaje de programación, más o menos estándar que se usa para que podamos crear documentos que se puedan ver con cualquier navegador. La descripción y elementos del mismo se han visto en el tema dedicado a este lenguaje.

XHTML (Lenguaje de Marcado de Hipertexto Extensible) es una versión más estricta y limpia de HTML, que nace precisamente con el objetivo de reemplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML.

XHTML surge como el lenguaje cuyo etiquetado, más estricto que HTML, va a permitir una correcta interpretación de la información independientemente del dispositivo desde el que se accede a ella.

7.2. Directivas JSP

Utilizadas para definir y manipular una serie de atributos dependientes de la página que afectan a todo el JSP. Suelen ser incluidas al comienzo de la página.

Sintaxis

<%@ *directiva* {atributo = "valor"}%>

Las directivas son las siguientes:

- Page
- Include
- Taglib

Directiva Page

Define atributos para la página.

Sintaxis <%@ page ATRIBUTOS %>

Donde ATRIBUTOS son parejas: nombre="valor"

```
<%@ page language="Java"
    import="Java.rmi.*,java.util.*" session="true"
    buffer="12kb"
%>
```

Existe una lista de atributos que pueden ser usados

Algunos de los atributos más usados

language="java"

Este atributo define el lenguaje de script usado en la página. En JSP será siempre Java.

import= "{ package.class / package.* }, ..."

Esta lista especifica una lista separada por comas de uno o más paquetes o clases que el fichero JSP debería importar. Las clases de los paquetes se ponen a disposición de los scriptlets, expresiones, declaraciones y etiquetas dentro del fichero JSP.

El atributo **import** debe aparecer antes de cualquier etiqueta que refiera la clase importada. Para importar varios paquetes, podemos usar una lista separada por comas, más de una directiva **import** o una combinación de ambas.

session="true|false"

Todo cliente debe unirse a una sesión HTTP para poder usar una página JSP. Si el valor es **true**, el objeto **session** se refiere a la sesión actual o a una nueva sesión. Si el valor es **false**, no podemos utilizar el objeto **session** en el fichero JSP. El valor por defecto es **true**.

buffer="none|8kb|sizekb"

Este atributo especifica el tamaño del buffer en kilobytes que será usado por el objeto **out** para manejar la salida enviada desde la página JSP compilada hasta el navegador del cliente.

info="text"

Este atributo nos permite especificar una cadena de texto que es incorporada en el página JSP compilada. Podemos recuperar el string más tarde con el método **getServletInfo()**.

errorPage="URLrelativa"

Este atributo especifica un path a un fichero JSP al que este fichero JSP envía excepciones. Si el path empieza con una "/", el path es relativo al directorio raíz de documentos de la aplicación JSP y es resuelto por el servidor Web. Si no, el path es relativo al fichero JSP actual.

Directiva Include

Incluye estáticamente el contenido de un archivo insertándolo en el lugar donde se ubica la directiva del JSP. El contenido del fichero incluido es analizado en el momento de la traducción del fichero JSP y se incluye una copia del mismo dentro del servlet generado. Una vez incluido, si se modifica el fichero no se verá reflejado en el servlet. El tipo de fichero a incluir puede ser un fichero HTML, JSP, XML, texto...

Con esta directiva se consigue una mayor modularidad de la página incluyendo contenidos que se repiten, por ejemplo: cabeceras, banners, etc. Y facilita el mantenimiento porque centraliza las modificaciones y minimiza el trabajo de correcciones y pruebas.

Sintaxis `<%@ includefile="Nombre del fichero" %>`

Ejemplo: Página JSP que incluye el contenido de dos ficheros (una página HTML y una página JSP)

```
<HTML>
<head>
<title> Página de prueba de directivas de
compilación </title>
```

```
</head>
<body>
<h1> Página de prueba de directivas de
compilación </h1>
<% include file="/fichero.html" %>
<%@ include file="/fichero.jsp" %>
</body>
</HTML>
```

Directiva Taglib

Indican al compilador que se va a utilizar una librería de etiquetas

Sintaxis

<%@ taglib uri="librería" prefix="prefijo a utilizar" %>

Son etiquetas escritas de forma independiente en código Java y que se utilizan directamente en la JSP, fomentando la reutilización, facilitando las pruebas y la independencia.

La etiquetas tendrán la siguiente forma: **<prefijo:método> cuerpo </prefijo:método>**

7.3. Elementos script

Los elementos de script nos permiten insertar código Java dentro del servlet que se generará desde la página JSP actual. Hay tres tipos

- Declaraciones de la forma **<%! código %>** que se insertan en el cuerpo de la clase, fuera de cualquier método existente.
- Scriptlets de la forma **<% código %>** que se insertan dentro del método.
- Expresiones de la forma **<%= expresión %>** que son evaluadas e insertadas en la salida

7.3.1. Declaraciones

Estas variables o métodos declarados pasarán a ser variables de instancia de la clase servlet generada. Esto significa que serán globales a todo el servlet generado para la página

Sintaxis

<% ! Declaración %>

Ejemplo:

```
<%! int contador >
```

Como las declaraciones no generan ninguna salida, normalmente se usan en conjunción con expresiones JSP o scriptlets.

Por ejemplo, aquí tenemos un fragmento de JSP que cuenta el número de veces que se ha solicitado la página actual desde que el servidor se arrancó.

```
<%! private int accessCount = 0; %>
        Número de veces que se consultó a la página:
<%= ++accessCount %>
```

Uso de un contador que indica el número de veces que se accede a una página.

```
<HTML> <head>
<title> Página de control de declaraciones
</title> </head>
<body>
<h1> Página de control de declaraciones </h1>
<%! int i=0 ; %> <!-- Esto es una declaración
(una variable de instancia en este caso) -->
<%
i++;
%> <!-- Esto es un scriptlet (código Java) que se ejecuta-->
HOLA MUNDO
<%= "Esto ha sido un JSP accedido " + i + " veces" %>
<!-- Esto es una expresión que se evalúa y se
sustituye en la página por su resultado-->
</body></HTML>
```

7.3.2. Scriptlets

Un scriptlet es un bloque de código Java insertado en la página y ejecutado durante el procesamiento de la respuesta. El código introducido se inserta directamente en el método `_jspService()` del servlet generado para la página.

Sintaxis:

<% código Java %>

Ejemplo

```
<% int i,j;
for (i=0;i<3;i++) {
j=j+1;
}
%>
```

Página JSP que usa código Java para repetir 10 veces un saludo

```
<HTML> <head>
<title> Página de ejemplo de scriptlet
</title> </head>
<body> <h1> Página de ejemplo de scriptlet </h1>
<%
for (int i=0; i<10; i++){
    out.println("<b> Hola a todos. Esto es un
    ejemplo de scriptlet " + i + "</b>");
    System.out.println("Esto va al stream System.out" + i );
//Esto último va a la consola del Java, no al cliente.
//out a secas es para la respuesta al cliente.
}
%>
</body>
</HTML>
```

El código dentro de un scriptlet se insertará **exactamente** como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias **print**. Esto significa que los scriptlets no necesitan completar las sentencias Java, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets. Por ejemplo, el siguiente fragmento JSP, contiene una mezcla de texto y scriptlets:


```
<% if (Math.random() < 0.5) { %>
  Tiene un <B>día</B> feliz!
<% } else { %>
  Tiene un <B>dia</B> aburrido!
<% } %>
```

que se convertirá en algo así

```
<% if (Math.random() < 0.5) {
out.println("Tiene un <B>día</B> feliz!");
} else {
  out.println("Tiene un <B>día</B> aburrido!");
} %>
```

7.3.3. Expresiones

Las expresiones se evalúan y se insertan en la salida. La expresión se traduce por la llamada al método `println` del objeto `out` dentro del método `_jspService()`, con lo que en cada petición, la expresión es evaluada y el resultado se convierte a un `String` y se visualiza

Sintaxis

<%= Expresión Java a evaluar %>

```
<%= "Esta expresión muestra el valor de un contador " + contador %>
```

(será necesario que previamente contador haya tomado un valor a través de un scriptlet)

En esta página JSP la expresión consiste en crear un objeto y llamar a uno de sus métodos. El resultado es un string que se muestra al cliente

```
<HTML>
<head>
<title> Página de ejemplo de expresiones
</title>
</head>
<body>
<h1> Página de ejemplo de expresiones </h1>
Hola a todos, son las <%= new Date().toString()%>
Hoy estamos a <%= (new java.util.Date()) %>
</body>
</HTML>
```

7.4. Comentarios JSP

Hay tres tipos de comentarios:

- Los comentarios JSP, que son del tipo `<%-- comment --%>` es ignorado cuando se traduce la página JSP en un servlet. Se pueden colocar en cualquier parte, pero no dentro de los scriptlets.
- Los de HTML (XHTML) `<!--comentario -->` No se puede colocar dentro de los scriptlets
- Los de Java `/* comentario */` junto con los de fin de línea de java `//`

Los comentarios *jsp* y *java* se ignoran y no aparecen en la respuesta al cliente. Cuando los clientes ven el código fuente de una respuesta *jsp* solo ven los comentarios *html*

7.5. Objetos predefinidos.

Para simplificar el código en expresiones y scriptlets JSP, se dispone de una serie de **objetos implícitos o predefinidos**. No se declaran previamente, ni se inicializan. Con ellos se tendrá acceso a la navegación de la página, al HTML de salida generado, a las cookies, a datos a cerca de la máquina cliente,....

- REQUEST: Contiene información de la página y de las propiedades y atributos definidos en este ámbito. Permite mirar los parámetros de la petición (mediante `getParameter`), el tipo de petición (GET, POST, HEAD, etc.), y las cabeceras http entrantes (cookies, Referer,).
- RESPONSE: Contendrá la respuesta generada para la nueva página a mostrar.
 - Establece el tipo del mensaje de respuesta `response.setContentType(String)`.
 - Redirigir a otra página `response.sendRedirect(String)`.
 - Coloca una cookie en el cliente `response.addCookie(Cookie)`.
- OUT: Escribe directamente en el HTML generado el resultado de la ejecución de la JSP. Este es el `PrintWriter` usado para enviar la salida al cliente.

- SESSION y APPLICATION: son dos ámbitos en los que se pueden ubicar atributos. Si se usa el atributo session de la directiva page para desactivar las sesiones, los intentos de referenciar el objeto SESSION causarán un error en el momento de traducir la página JSP a un servlet.

Ejemplo

Se trata de pasar un parámetro con el nombre; en el caso de que no se pase el nombre lo pide desde un formulario visualizando un saludo

```
<%@ 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">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> entrega el nombre como parámetro, de lo contrario lo pide y
    saluda
</title>
</head>
<body>
    <%
        String nombre = request.getParameter("nombrePila");
        if (nombre != null) {
    %>
    <h1>
        Hola <%= nombre %>, <br />
        ¡ Bienvenido a la página !
    </h1>
    <%}
    else {
    %>
        <form action="saludo.jsp" method = "get">
        <p>Escriba su nombre y pulse Enviar </p>
        <p><input type = "text" name = "nombrePila" />
            <input type = "submit" value = "Enviar" />
        </p>
        </form>
    <%}%>
</body>
</html>
```

En el ejemplo anterior hay código estático XHTML

```
Hola <%= nombre %>, <br />
¡ Bienvenido a la página !
```

Dentro del mismo hay una expresión jsp <%= nombre %> y el formulario para pedir el nombre

```

<form action="saludo.jsp" method = "get">
    <p>Escriba su nombre y pulse Enviar </p>
    <p><input type = "text" name = "nombrePila" />
        <input type = "submit" value = "Enviar" />
    </p>
</form>

```

Y código dinámico

```

String nombre = request.getParameter("nombrePila");
    if (nombre != null)
    {
        // aquí va el código XHTML de bienvenido
    }
    else
    {
        // aquí va el código del formulario
    }

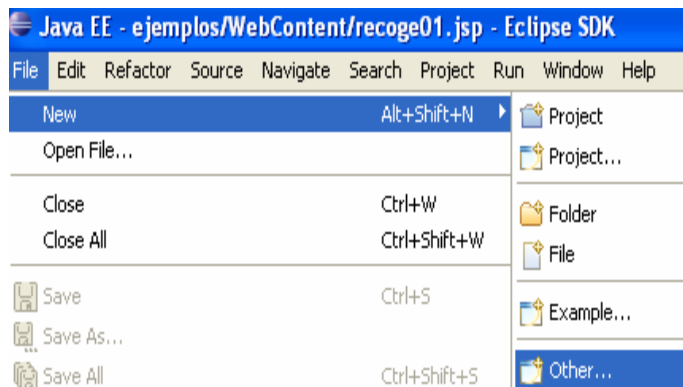
```

El código dinámico se ubica en los scriptes `<% código %>`

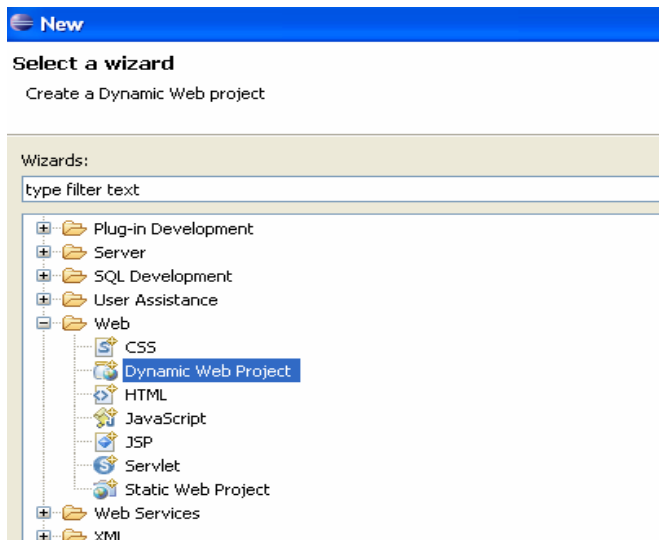
Vamos a comprobar el ejemplo desde eclipse, primero creamos un nuevo proyecto, lo llamamos **saludo** y luego un fichero **jsp** con el mismo nombre.

Se arranca el Eclipse, validamos el espacio de trabajo y creamos un nuevo proyecto siguiendo la secuencia

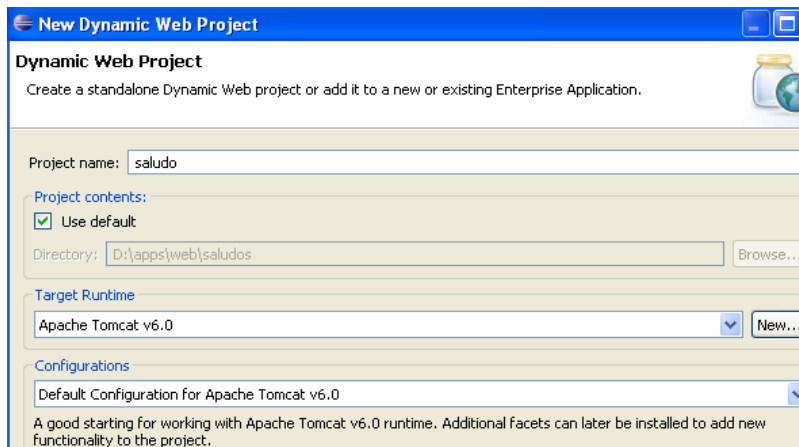
File → New → Other



Y se elige Dynamic Web Project

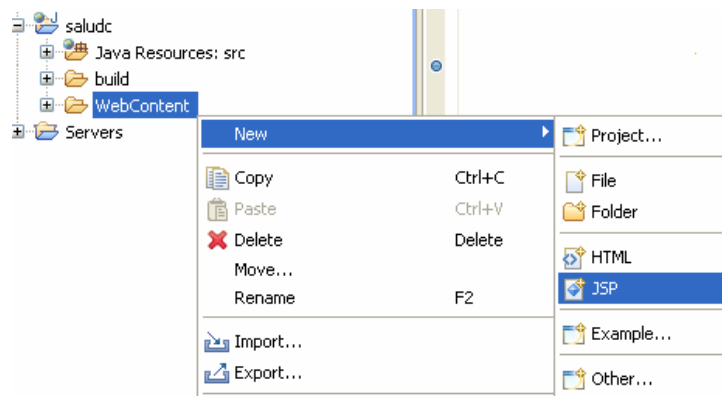


Le damos nombre al proyecto. En Target Runtime y en Configurations tiene que aparecer la versión del Tomcat con la que trabajamos, como se ve en la siguiente figura. Seleccionar Finish para terminar.



Una vez creado el proyecto, posicionarse en **WebContent** → **botón derecho** → **New** →

JSP



Se visualiza la siguiente plantilla. Es donde se escribe el código

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

</body>
</html>
```

Finalmente, una vez introducido el código, el contenido del fichero será el que aparece en la siguiente captura

```
<%@ 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/T
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title> entrega el nombre como parámetro, de lo contrario lo pide y saluda </title>
</head>
<body>
    <%
        String nombre = request.getParameter("nombrePila");
        if (nombre != null) {
    >%
        <h1>
            Hola <%= nombre %>, <br />
            ¡ Bienvenido a la página !
        </h1>
    <%
        }
        else {
    >%
            <form action="saludo.jsp" method = "get">
                <p>Escriba su nombre y pulse Enviar </p>
                <p><input type = "text" name = "nombrePila" />
                    <input type = "submit" value = "Enviar" />
                </p>
            </form>
    <%>>
    </body>
</html>
```

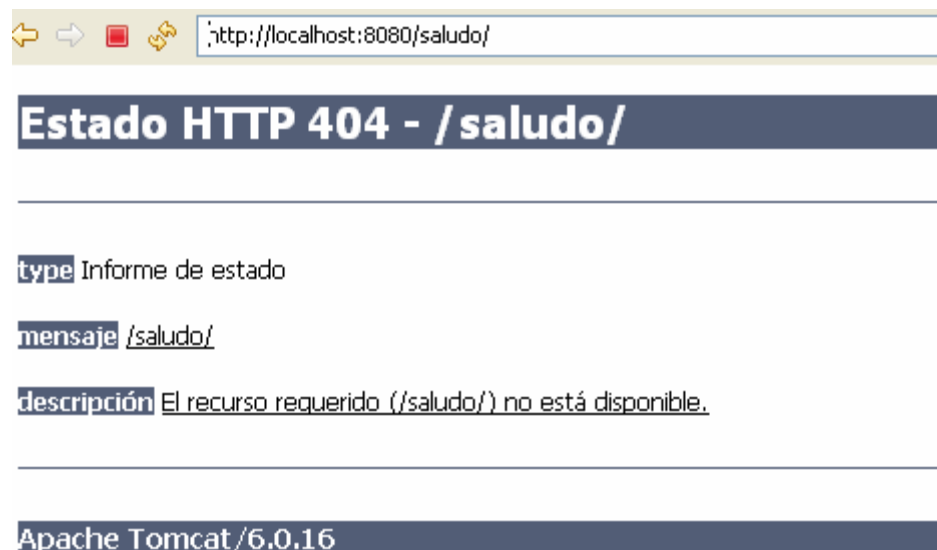
La estructura es la siguiente



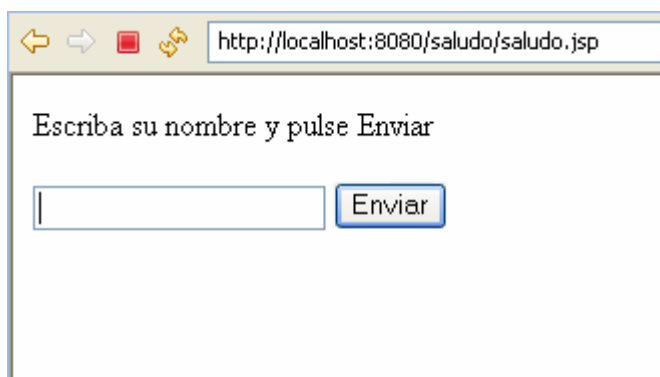
Para ejecutar la página seguimos la siguiente secuencia:

Posicionarse sobre el fichero jsp **saludo** → **botón derecho** → **Run As** → **Run on Server** → **Finish**

Si no encuentra la página es porque nos posicionamos sobre el proyecto y no sobre la *jsp*. Visualizándose el siguiente error.

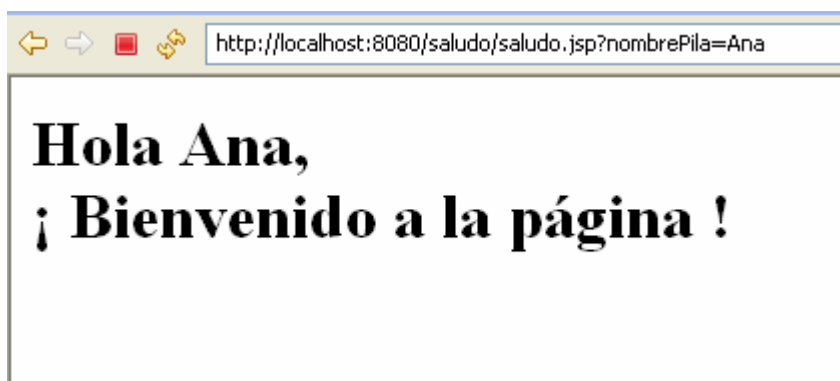


Se debe a que la URL no es la correcta, como se ve a <http://localhost:8080/saludo/> le falta la página *jsp*. La correcta sería <http://localhost:8080/saludo/saludo.jsp> En este caso se visualiza la siguiente página



Como la primera vez que se ejecuta la URL no le pasa ningún valor como parte de la petición el método *getParameter* devuelve *null*, mostrándose el formulario anterior para que se teclee un nombre.

Tecleamos el nombre (por ejemplo Ana) y pulsamos en *Enviar* para pedir la *jsp* de nuevo con el valor Ana, visualizándose la página.



En este caso la *jsp* recibió un valor para el nombre de pila como parte de la petición. El método *getParameter* del objeto implícito *request* asigna el resultado a la variable *nombre*

Fijaros en la URL de la nueva página

<http://localhost:8080/saludo/saludo.jsp?nombrePila=Ana>

Otro ejemplo

Formulario `index.html`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"
/>
<title>Ejercicios saludos</title>
</head>
<body>
<p>Nos devuelve un saludo con el nombre que le pasemos en el
formulario</p>
<form action="saludosbis.jsp" method="post">
  <input type="text" name="nombre">
  <input type="submit" value="enviar">
</form>
</body>
</html>
```

Página JSP: `saludosbis.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">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<% if ( request.getParameter("nombre") != null &&
request.getParameter("nombre") != "" ) { %>
  Hola <%= request.getParameter("nombre") %>
<% } else { %>
  Hola Desconocido
<% } %><br />
<a href="index.html">Volver</a>
</body>
</html>
```

7.6. Etiquetas estándar

También se denominan *acciones estándar*. Son marcas con formato XML, que afectan al comportamiento en tiempo de ejecución del JSP y la respuesta se devuelve al cliente. Estas acciones proporcionan el acceso a las tareas más comunes que se realizan en una página jsp, como incluir contenidos de otros recursos, reenviar peticiones, interactuar con JavaBeans.

Los elementos XML, al contrario que los del HTML, son sensibles a las mayúsculas, asegúrate de usar minúsculas.

Las etiquetas están delimitadas por las marcas `<jsp:etiqueta >` y `</jsp:etiqueta>`. En la traducción de JSP al servlet, la marca se reemplaza por cierto código Java que define a dicha marca. Una marca por tanto define un cierto código Java (es como una macro)

Constan de un prefijo y un sufijo además de una serie de atributos. El prefijo es siempre "jsp" en las etiquetas estándar

Sintaxis

`<jsp:sufijo atributos/>`

Ejemplo

```
<jsp:include page="mijsp.jsp" flush="true " />
```

Otro ejemplo

La siguiente directiva

```
<%@ page import="java.util.*" %>
```

En formato XML es

```
<jsp:directive.page import="java.util.*" />
```

El equivalente XML de `<%! Código %>` es:

```
<jsp:declaration>
Código
</jsp:declaration>
```

Tipos de etiquetas estándar

- `<jsp:include>` Incluye de forma dinámica otro recurso de JSP
- `<jsp:forward>` Reenvía el procesamiento de la petición hacia otra página JSP o a una página estática. Esta acción termina la ejecución de la JSP actual.
- `<jsp:param>` Se utiliza con las acciones anteriores. Permite pasar valores de una página a otra o a un componente externo.
- `<jsp:useBean>` Especifica que la JSP utiliza una instancia de JavaBeans. Permite recuperar una referencia a un objeto que está almacenado en *request* o *session* dependiendo de lo que indiquemos en el parámetro *scope* de la etiqueta.
- `<jsp:setProperty>` Permite cambiar el valor de una propiedad del objeto previamente recuperado con *useBean*
- `<jsp:getProperty>` Permite obtener el valor de una propiedad del objeto previamente recuperado con *useBean*

<jsp:include> Permite incluir contenido dinámico en tiempo de petición, no en tiempo de traducción, como es el caso de la directiva *include*.

```
<jsp:include page="relative URL" flush="true" />
```

Atributo **page** Especifica la URL del recurso a incluir. Debe formar parte de la misma aplicación web.

Atributo **flush** Especifica que el búfer debe vaciarse después de realizar la inclusión.

jsp:useBean permite cargar y utilizar un JavaBean en la página JSP y así utilizar la reusabilidad de las clases Java. Los JavaBeans son pequeños componentes con funcionalidad propia.

```
<jsp:useBean id="name" class="package.class" />
```

Esto normalmente significa "usa un objeto de la clase especificada por *class*, y únelo a una variable con el nombre especificado por *id*".

Ahora podemos modificar sus propiedades mediante **jsp:setProperty**, o usando un scriptlet y llamando a un método de id. Para recoger una propiedad se usa **Jsp:getProperty**

La forma más sencilla de usar un Bean es usar:

```
<jsp:useBean id="name" class="package.class" />
```

- **id** Da un nombre a la variable que referenciará el bean. Se usará un objeto bean anterior en lugar de instanciar uno nuevo si se puede encontrar uno con el mismo id y scope.
- **class** Designa el nombre cualificado completo del bean.
- **scope** Indica el contexto en el que el bean debería estar disponible. Hay cuatro posibles valores: page, request, session, y application.
- **type** Especifica el tipo de la variable a la que se referirá el objeto.

Ejemplos

El siguiente ejemplo nos muestra el uso de la etiqueta include.

Desde eclipse creamos un nuevo proyecto. File → New → Other → Dynamic Web Project → Next al proyecto lo llamamos *ejemplos* y pulsamos Finish

Posicionarse sobre el proyecto *ejemplos* → botón derecho → New → JSP el nombre del fichero *jsp*, le llamamos *incluir* y pulsar Finish

Se modifica la plantilla para obtener un fichero como muestra el siguiente listado

```
<%@ 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">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Ejemplo que muestra la utilización de la etiqueta include</title>
</head>
<body>
Página para probar la etiqueta jsp:include
<br><br>
<jsp:include page="fecha.jsp"></jsp:include>
</body>
</html>
```

Lo realmente importante en el fichero anterior es la línea

```
<jsp:include page="fecha.jsp"></jsp:include>
```

Donde se utiliza la etiqueta que incluye la página *fecha.jsp*

Creemos otro fichero al que llamamos *fecha*. La plantilla que crea por defecto la modificamos eliminando todo el contenido excepto la primera línea y le añadimos el siguiente código:

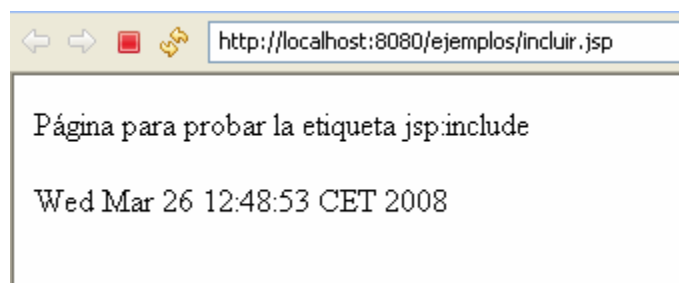
```
<%@ page import="java.util.*" %>  
<%Date fecha = new Date();  
    out.println(fecha);  
%>
```

Finalmente el fichero *fecha.jsp* tiene el siguiente contenido

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>  
<%@ page import="java.util.*" %>  
<%Date fecha = new Date();  
    out.println(fecha);  
%>
```

Para ejecutar posicionarse sobre el fichero incluir → botón derecho → Run as → Run on Server → Finísh

Se visualiza la siguiente página jsp



Escriba su nombre y pulse Enviar para que se repita cinco veces

Pedro
Pedro
Pedro
Pedro
Pedro

8. Entorno de Desarrollo.

Software que debe estar instalado para desarrollar aplicaciones Web: JDK, TOMCAT y ECLIPSE

8.1. JDK

Para trabajar con JAVA se necesita el kit de desarrollo JDK (Java Development Kit) entre otras herramientas dispone de:

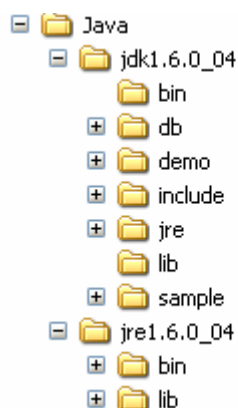
- javac.exe Compilador de Java
- java.exe Intérprete de Java (JVM)
- appletviewer.exe Intérprete de applets
- jdb.exe Depurador
- javadoc.exe Generador de documentación
- javah.exe Integrador del código C/C++ para JNI
- javap.exe Desensamblador.

Si no lo tenéis instalado lo podéis bajar de la página oficial de Sun

<http://java.sun.com/javase/downloads>

Junto con el JDK se descarga el **JRE** (Java Runtime Environment) que permite ejecutar aplicaciones que se han escrito utilizando el lenguaje de programación Java.

Después de instalar en **archivos de programa** tiene que aparecer la siguiente estructura de directorios



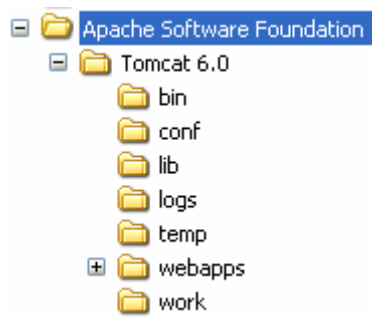
8.2. TOMCAT

Para poder desarrollar aplicaciones J2EE tenemos que tener un software para alojar aplicaciones web que actúa como servidor y que responde a las peticiones de los navegadores. En nuestro caso para trabajar con páginas JSP se necesita tener instalado el TOMCAT que es gratuito y pertenece a Apache Software Foundation.

La página desde donde se puede bajar es la siguiente

<http://tomcat.apache.org/>

Una vez instalado debería haber creado la siguiente estructura



/bin	Contiene fichero ejecutables. Desde aquí se puede arrancar y cerrar el <i>tomcat</i>
/temp	Tiene archivos temporales
/conf	Archivos para configuración del <i>tomcat</i> . El más importante es el <i>server.xml</i>
/log	Archivos de registro
/webapps	Directorio con las aplicaciones web
/work	Almacenamiento temporal de ficheros

En la carpeta *Conf* podemos actuar sobre el fichero *Server* para modificar la configuración del *Tomcat*, que entre otras cosas contiene la definición del puerto asociado a http, como se ve en la siguiente imagen por defecto es el 8080 que como vimos choca con oracle.

```
<Connector port="8080" protocol="HTTP/1.1"  
    connectionTimeout="20000"  
    redirectPort="8443" />
```

Se puede dejar el de oracle como 8080 y cambiar el de Tomcat por otro, de esta forma no chocarían.

```
<Connector port="8085" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

Si tenéis instalado el Tomcat 6.0 podéis arrancar el servicio desde la carpeta Bin pinchando en el fichero *tomcat6* o bien en *tomcat6w* desde donde se puede arrancar o parar.

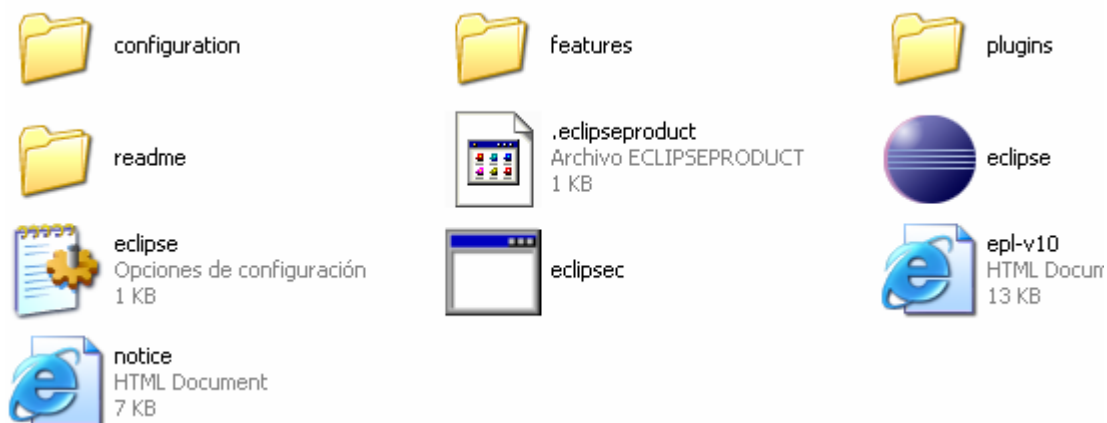
8.3. ECLIPSE

Se trata de un entorno de desarrollo integrado (IDE) es un programa compuesto por un conjunto de herramientas que se utiliza para desarrollo con java y otros lenguajes como C, C++. Está hecho en Java, por lo que necesitaremos tener la JVM instalada en el ordenador para poder ejecutarlo. No lleva instalador, se descomprime en una carpeta cualquiera del equipo y ya podemos utilizarlo, incluso podemos cambiarlo de carpeta en cualquier momento.

Se descarga de la siguiente página web

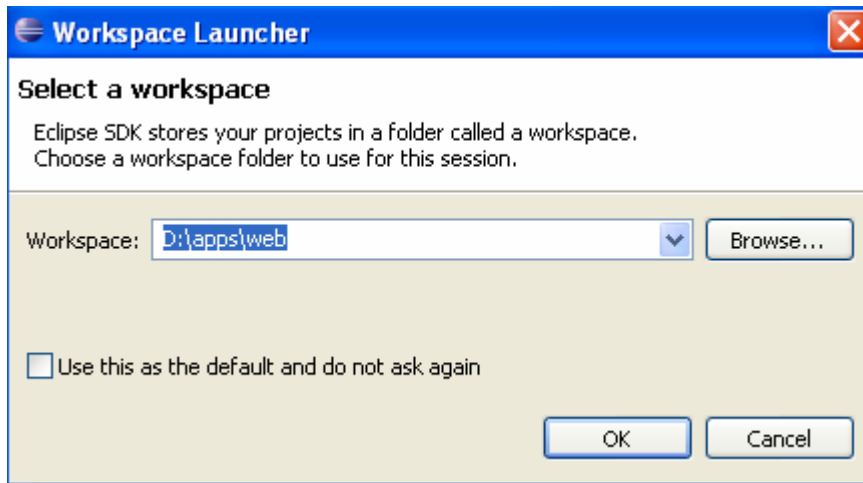
<http://www.eclipse.org/downloads/>

Una vez descargado se descomprime en una carpeta cuyo contenido se muestra en la siguiente captura



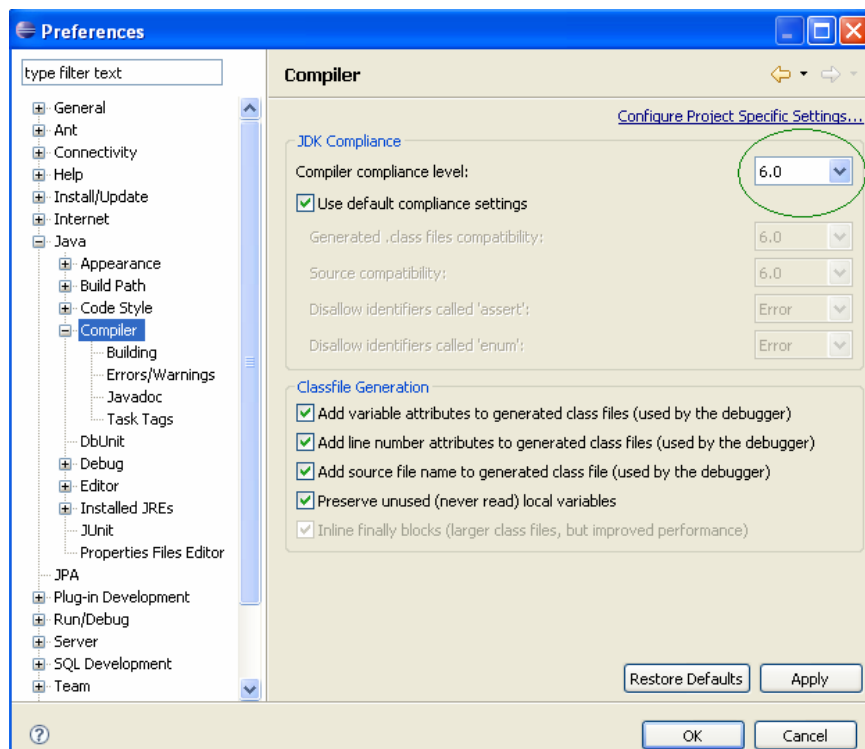
Antes de ejecutar recordad que tiene que estar instalado el JDK

Al ejecutar Eclipse, lo primero que nos pide es la ruta del entorno de trabajo (workspace). Es donde Eclipse va a ir guardando todos los archivos de los proyectos en desarrollo.

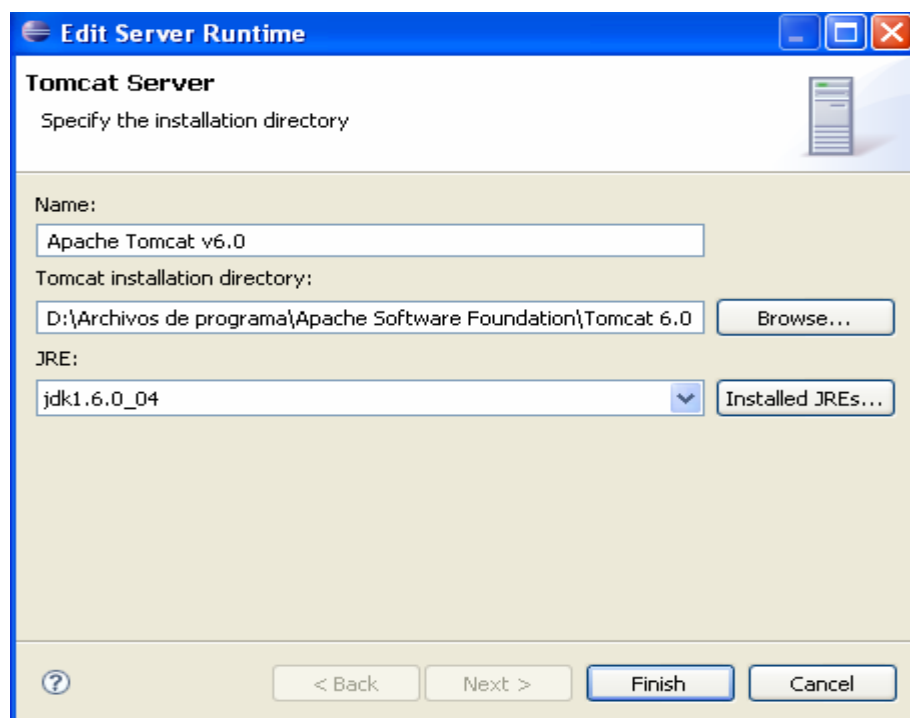


Se pueden buscar nuevas actualizaciones en **Help → Software Updates → Find and Install**.

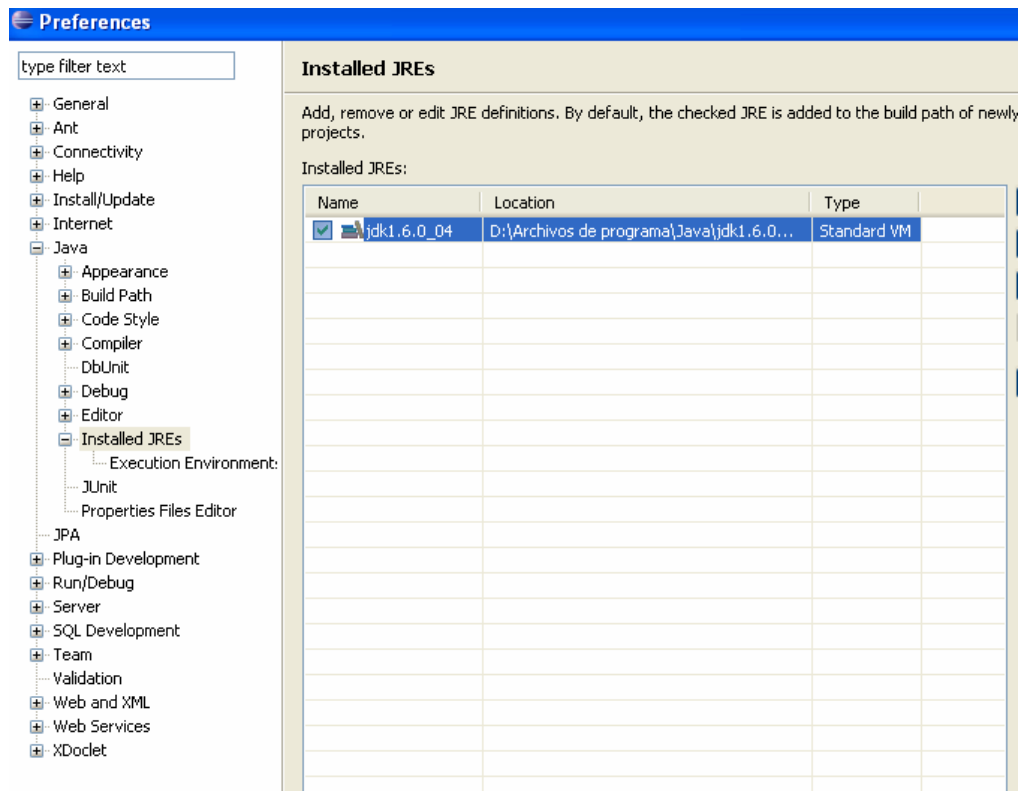
En **window → Preferentes → java → Compiler** le indicamos al compilador con que nivel de código estamos trabajando



En *window* → *Preferentes* → *Server* → *instaled runtimes* le indicamos el servidor web y dónde está



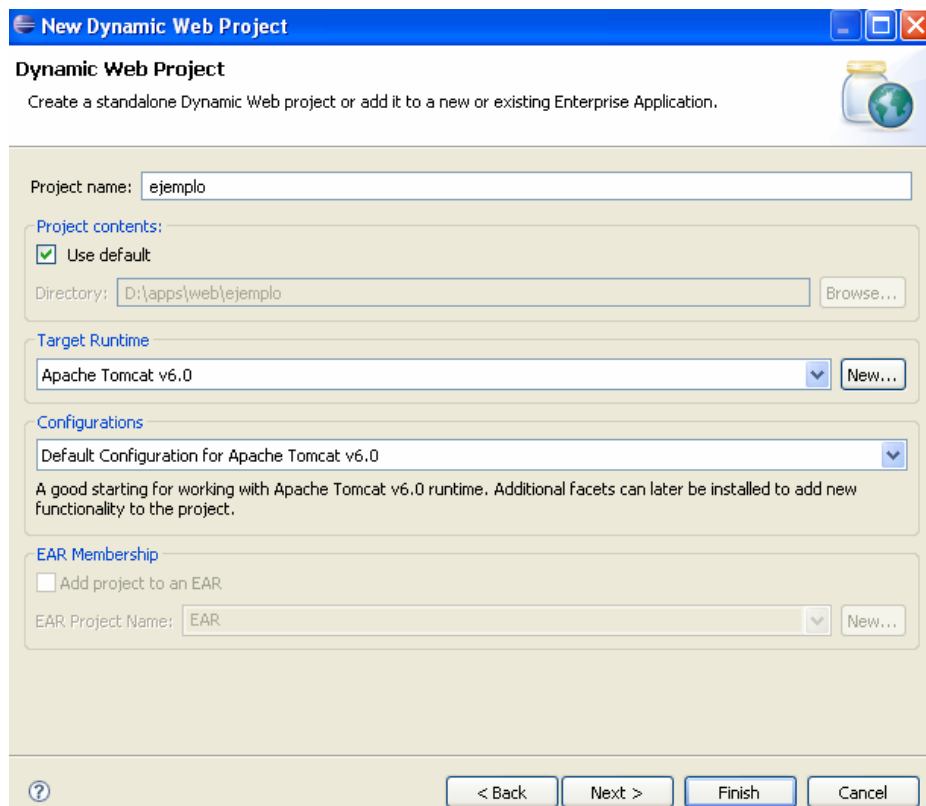
En **window** → **Preferentes** → **java** → **Instaled JREs** elegir el directorio donde está el JDK



Para crear un proyecto seleccionar **File** → **New** → **Other** → **Web** → **Dynamic web Project**

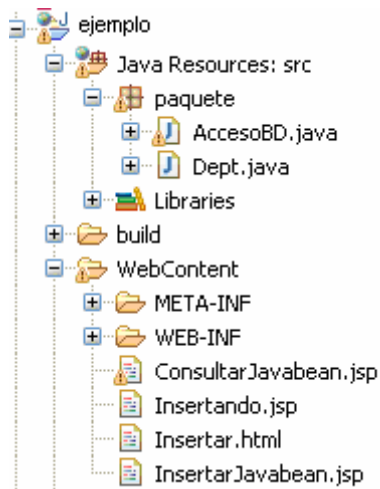
En **Project Name** le damos nombre al proyecto, fijaros en el resto de las entradas de la siguiente imagen

En **Target runtime**, tiene que aparecer apache **tomcat v6.0**. (la versión depende del que os hayáis bajado) y en **Configurations: default configuration for apache tomcat v6.0**



Si pregunta si quieres abrir la perspectiva **jee** le contestas que **yes**

Al crear un proyecto aparece una estructura de carpetas



En la carpeta **src** es donde crearemos nuestras **clases java agrupadas en paquetes**. El resultado de su compilación pasará automáticamente a la carpeta lib de WebContent, aunque no se vea.

En la carpeta **WebContent** es donde tendremos que colocar **todas las páginas web** (dinámicas o no) que utilicemos en nuestra aplicación así como imágenes, animaciones, etc, pudiendo crear nuestra propia estructura de carpetas conservando siempre WEB-INF y META-INF para su cometido.

8.4. CONFIGURACIÓN

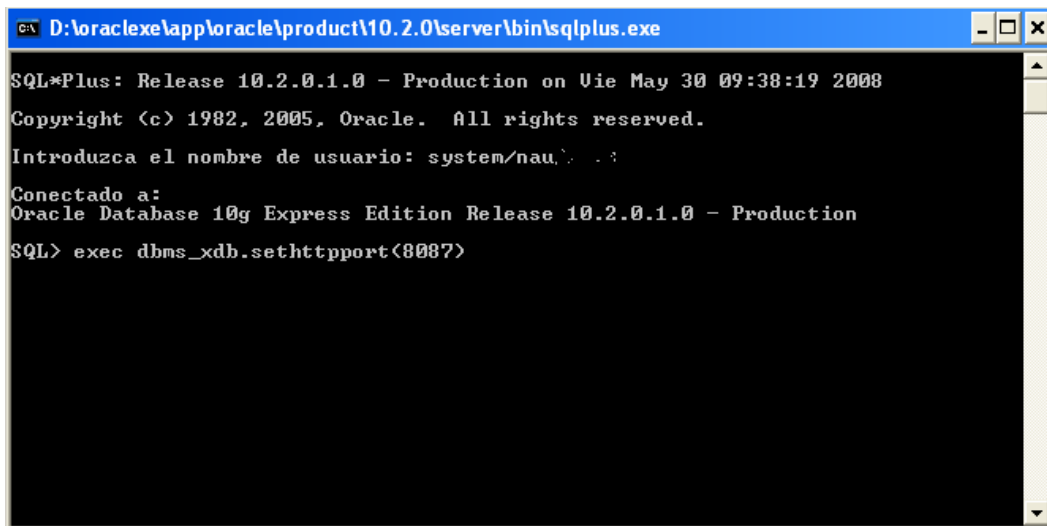
Hay que tener en cuidado con el puerto 8080 (http), tanto Oracle como Tomcat lo utilizan, por lo tanto el que arranca primero lo coge y el otro no funciona. Se soluciona cambiándolo en el momento de la instalación. En el caso de Oracle pide cambiarlo cuando se esta instalando si detecta que está ocupado, en caso contrario asume por defecto el 8080, con Tomcat pasa lo mismo. Suponiendo que estén los dos instalados con el puerto 8080 asignado, podemos cambiarlo después de la instalación en cualquiera de los dos productos.

- Para cambiarlo en el Tomcat hay que editar el fichero Server.xml que está en el directorio /conf. Podéis hacer la prueba cambiándolo aquí.
- Desde Oracle se puede modificar ejecutando el paquete DBMS_XDB.SETHTTPPORT. Se puede hacer desde el entorno SQLPLUS, siguiendo la siguiente secuencia:

Lanzar SQLPLUS /NOLOG. En la línea de comandos **inicio→ejecutar → sqlplus /nolog**

Conectarse como **system/contraseña que hayáis puesto en la instalación** y lanzar el siguiente paquete:

EXEC DBMS_XDB.SETHTTPPORT(8087)



```
D:\oracle\app\oracle\product\10.2.0\server\bin\sqlplus.exe
SQL*Plus: Release 10.2.0.1.0 - Production on Uie May 30 09:38:19 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Introduzca el nombre de usuario: system/nau.
Conectado a:
Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
SQL> exec dbms_xdb.sethttpport(8087)
```

Cuando vayáis a la página de inicio de Oracle puede daros un error porque por defecto asume 8080, la cambiáis por 8087

Por defecto es <http://host:8080/apex> hay que poner <http://host:8087/apex>

Host será el nombre o ip del ordenador. Veis un ejemplo en las siguientes capturas:



Se cambia el puerto 8080 por 8087



Ahora ya se puede trabajar con Tomcat y la base de datos arrancada

Desarrollo de Aplicaciones Informáticas

materiales didácticos de aula



UNIÓN EUROPEA
Fondo Social Europeo



Gobierno del Principado de Asturias
Consejería de Educación y Ciencia



FORMACIÓN PROFESIONAL
Principado de Asturias