



Capítulo 3

Conceptos básicos de diseño de aplicaciones Web

Herramientas de desarrollo

Aspectos docentes

En este capítulo se introducen los conceptos fundamentales del diseño de aplicaciones WEB, una introducción básica a la tecnología Servlet y JSP y el uso del IDE Eclipse. Se trata de un capítulo que aborda principios teóricos de diseño fundamentales para la elaboración de las aplicaciones WEB. Así se aprenderá a diseñar una pequeña aplicación Java, ejecutarla y, de paso, aprender la sintaxis básica de JSP. Para ello se empleará como hilo argumental el caso práctico El Fotograma Perdido, ya planteado al final del capítulo anterior. Se toma como partida el enunciado básico, y se ponen ejemplos que versan sobre el mismo, abordando posteriormente las ampliaciones correspondientes. También se requiere que esté instalado un servidor de aplicaciones, pudiéndose optar entre Tomcat 6 o JBoss 5. En nuestro caso, desplegaremos las aplicaciones en JBoss 5, aunque en equipos con cierta limitación de memoria o procesador puede elegirse Tomcat 6.

Herramientas necesarias

- Eclipse Helios.
- SDK J2SE 1.6.0 o superior.
- Servidor de aplicaciones JBoss 5.

Bibliografía

Sobre Java en general

Curso de Java	Ian F. Darwin	Anaya Multimedia	84-415-1792-4
Java 2 Características avanzadas. Vol II	Cay S. Horstmann y Cornell	GaryPrentice Hall	84-205-3701-2
Java 2 Fundamentos. Vol I	Cay S. Horstmann y Cornell	GaryPrentice Hall	84-205-3700-4
Java 2 Manual de Referencia	Herbert Schildt	McGraw-Hill	84-481-3173-8
Piensa en Java – Segunda Edición	Bruce Eckel	Prentice Hall	84-205-3192-8

Sobre diseño de aplicaciones web

JSP. Ejemplos prácticos	Andrew Patzer	Anaya Multimedia	84-41514666
Desarrollo Web con JSP	Falkner, Galbraith, Irani	Anaya Multimedia	84-41513525
Programación Java Server con J2EE 1.3	VV.AA.	Anaya Multimedia	84-41513589

Contenido

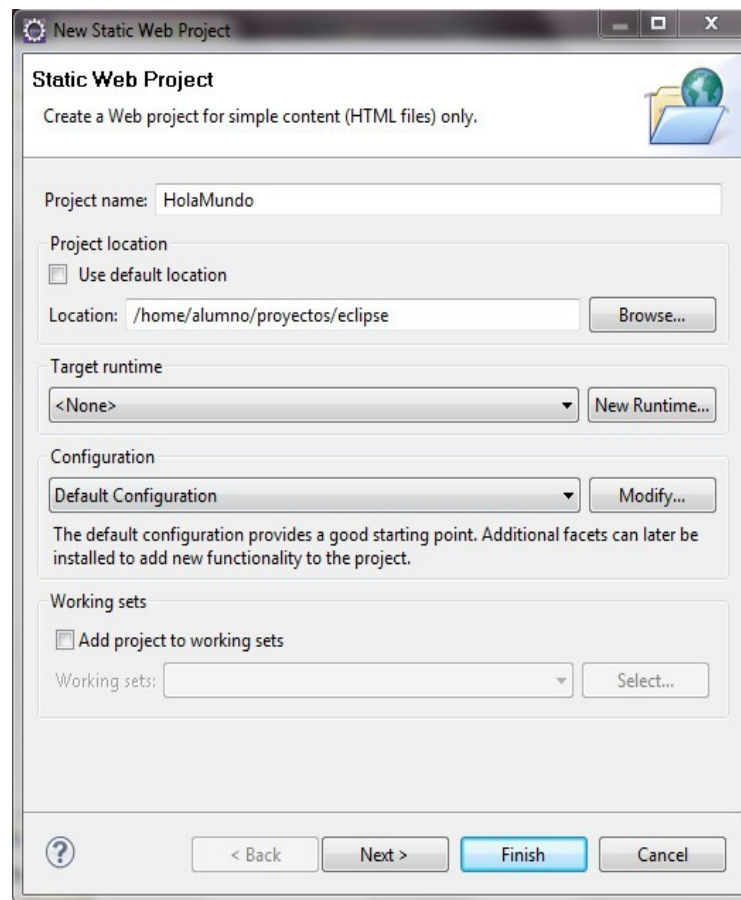
Hola Mundo WEB

El primer paso que daremos será el de crear una pequeña aplicación web tipo “HolaMundo” tal como suele hacerse en el aprendizaje del lenguaje C. La aplicación sólo tendrá páginas estáticas, por lo que realmente no se requiere un servidor de aplicaciones para ejecutarla, basta con un servidor web como Apache o cualquier otro similar.

El objetivo de la aplicación es mostrar una pantalla en el navegador que indique al usuario que pulse un botón para recibir un saludo, de tal forma que al pulsarlo se salte a una página que muestre ese saludo. Ambas páginas son estáticas, con formato HTML. La primera se llama *index.html* y la segunda *saludo.html*.

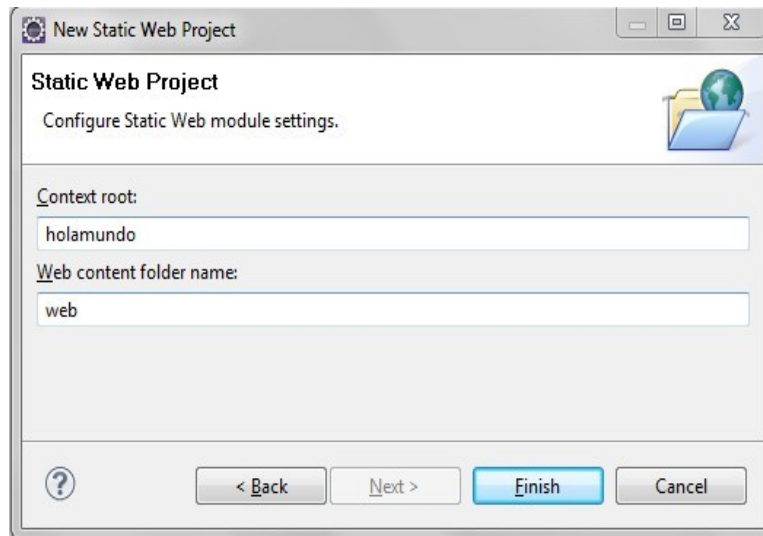


Se arranca el IDE, y se escoge la opción File->New->Static Web Project. Aparecerá un cuadro de diálogo donde se introducirá el nombre del proyecto.

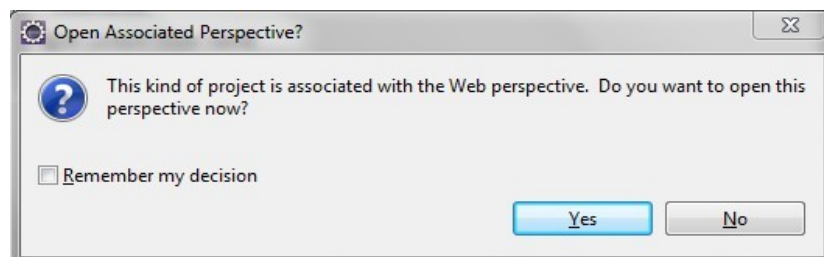


La ubicación del proyecto será un directorio denominado *proyectos/eclipse*, que se ubica en el directorio raíz de la cuenta del usuario. Si se trabaja en Windows, el directorio podría ser *c:\Users\alumno\proyectos\eclipse*. Como las capturas de pantalla se han realizado tanto desde Windows como desde Linux, los caminos de los directorios podrán seguir cualquiera de ambas convenciones.

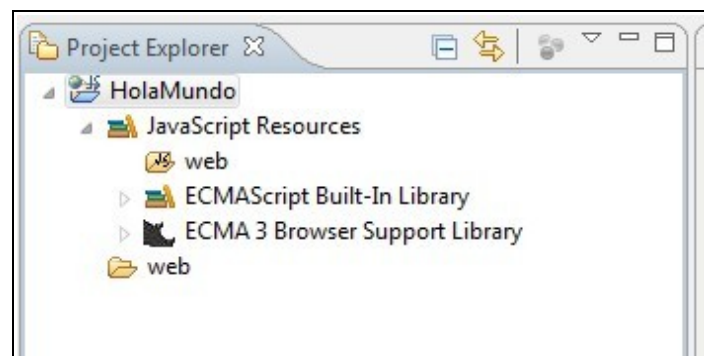
Después de introducir el nombre del proyecto y de establecer su ubicación, se pulsa el botón *Next*. Aparece un cuadro de diálogo que solicita el contexto raíz de la aplicación y el nombre del directorio que contendrá las páginas webs estáticas (HTML y similares). Por defecto, el contexto raíz se corresponde con el nombre del proyecto, pero puede cambiarse, aunque esto suponga tener que hacer *algo más* para poder desplegar la aplicación en JBoss. Cambiaremos el contexto raíz a *holamundo*, y el nombre del directorio a simplemente *web*, ambas en minúsculas. El contexto raíz de una aplicación web es el nombre de aplicación que aparece en la URL de acceso al servidor. En este caso, al ser el contexto raíz *holamundo*, la URL será algo *http://<servidor>:<puerto>/holamundo*, que se continuará con el nombre del recurso al que se quiera acceder. Estos cambios se observan en la imagen siguiente:



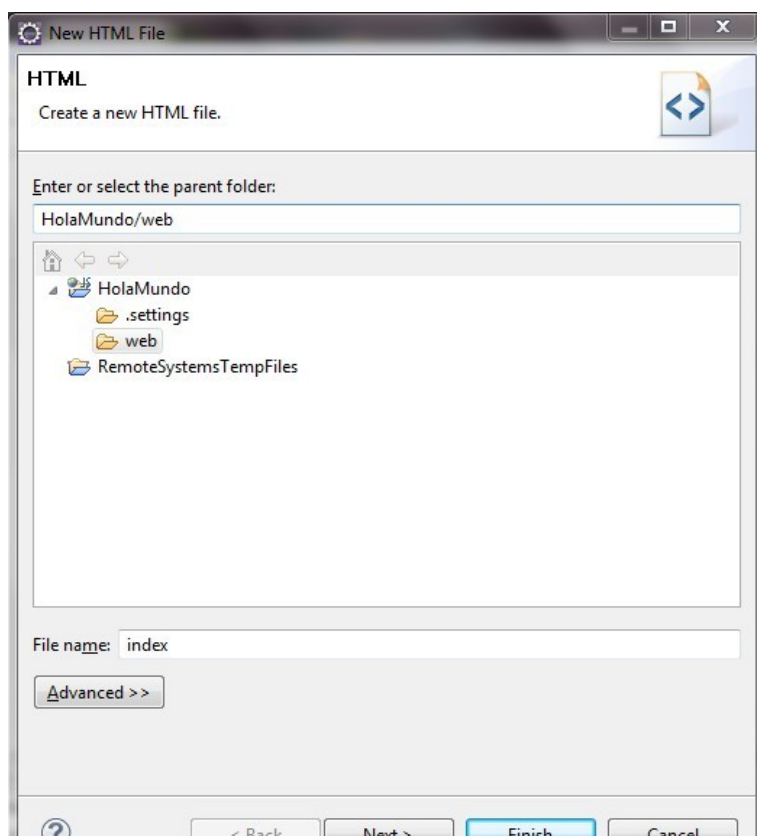
Se pulsa el botón *Finish*. Entonces Eclipse nos informa de que este tipo de proyecto no está asociado a una perspectiva de tipo *Java EE*, sino a una perspectiva *Web*. Le permitimos que abra la nueva perspectiva pulsando el botón *Yes*.



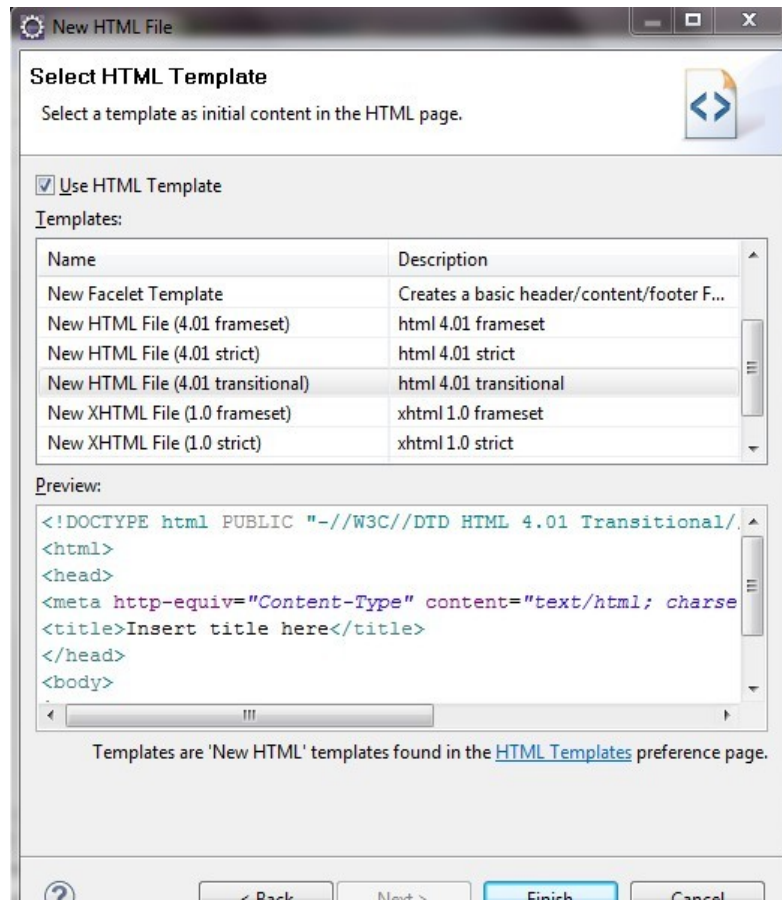
Se habrá creado un proyecto con una estructura como la que se observa en la siguiente imagen:



Una vez creado el esqueleto del proyecto se procede a la inserción de las páginas html. Se crea la primera página, *index.html*, mediante *File->New->HTML File* o bien accediendo al menú contextual del proyecto y seleccionando *New->HTML File*. Aparece la siguiente pantalla, en la que introducimos el nombre del archivo en cuestión sin la extensión html.



Se pulsa el botón *Next*. Aparece un cuadro de diálogo para escoger la plantilla HTML, en la siguiente imagen se muestra la escogida para este archivo.



Y se pulsa el botón *Finish*. Ahora el proyecto cuenta con un archivo más, *index.html*, ubicado dentro de la carpeta *web*. El código de este archivo se modifica tal como se muestra seguidamente, incorporando un formulario para saltar a la página de saludo.

```
<!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>Hola Mundo Web</title>
</head>
<body>
Pulse el botón para recibir un saludo.
<form action="saludo.html">
  <input type="submit" value="Enviar">
</form>
</body>
</html>
```

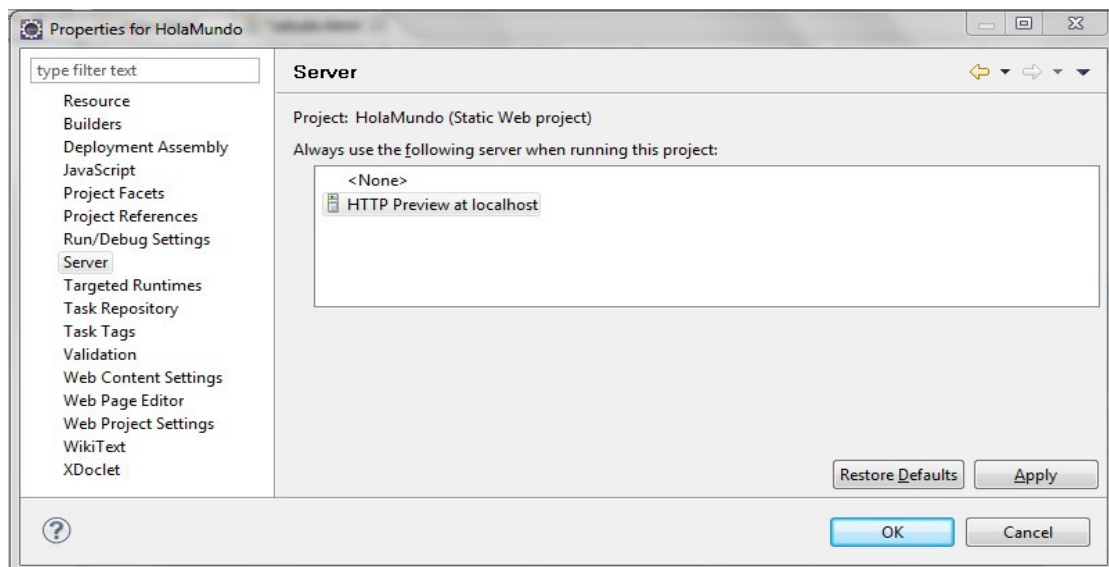
Procedemos de la misma forma para crear *saludo.html*, elaborando un código como el siguiente.


```
<!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>i iSaludos!!</title>
</head>
<body>
HOL@S ME ALEGRO DE VEROS
Pulse el botón para volver
<form action="index.html">
  <input type="submit" value="Volver"/>
</form>
</body>
</html>
```

Se guardan todos los archivos pulsando sobre el icono *Save All* (*CTRL+Shift+S*). Llega el momento de ejecutar la aplicación, en este caso, simplemente mostrar las páginas en el navegador del usuario haciéndolas pasar por un servidor HTTP. Para ello accederemos a las propiedades del proyecto mediante el menú contextual y la opción *Properties*. Seguidamente, en la opción *Server* de la pantalla de propiedades se podrá establecer el servidor asociado, que será *HTTP Preview at localhost*.



Se pulsa el botón *OK*. Para ejecutar el proyecto puede pulsarse el botón  o bien desde el menú contextual del proyecto, *Run As->Run on Server*. Se lanzará en el navegador externo²⁰ y se mostrará *index.html*. El puerto de escucha será el 8080.

Todo esto ha servido como toma de contacto con el entorno de desarrollo, aunque el tipo de proyecto a partir de ahora será el de *Dynamic Web Project* y no *Static Web Project*, lo que supone trabajar con el servidor de aplicaciones.

Un paso más


Llegados a este punto se hace necesario hacer algún que otro tutorial de Eclipse para familiarizarse con conceptos fundamentales del entorno, tales como *workbench*, *perspectivas*, *vistas*, *workspace* (o espacio de trabajo), etc. Yo recomiendo hacer el tutorial del *workbench*

²⁰ Es posible que como navegador esté configurado un navegador empotrado o interno de Eclipse. Para hacer que el navegador sea Internet Explorer, Firefox o cualquier otro externo, habrá que seleccionar en el menú *Windows* la opción *Web Server->Default System Web Server* o cualquiera otra que no sea *Internal Web Server*.



que se proporciona con Eclipse, y que se accede desde la página de bienvenida (Welcome). Si no se tiene abierta puede accederse mediante Help->Welcome. En esa página se pulsa sobre el icono *Overview*, y posteriormente en *workbench basics*. Es importante saber desenvolverse bien con los conceptos de perspectivas, que son conjuntos de vistas orientadas a determinados tipos de proyectos o tareas sobre Eclipse. Por ejemplo, la perspectiva Web posibilita trabajar con el diseño de páginas webs, pero para elaborar aplicaciones webs dinámicas, se requiere la perspectiva *Java EE*. Las perspectivas son accesibles desde la esquina superior derecha del entorno, o bien, si no están disponibles allí, mediante el menú *Window* y la opción *Open Perspective*.

Las perspectivas son grupos de vistas y editores. En una misma ventana de desarrollo pueden existir varias perspectivas, compartiendo todas ellas el mismo conjunto de editores. Un editor es un componente visual del workbench, pero se emplea para editar o mostrar un recurso, ya sea de texto o de tipo gráfico.

Las vistas son componentes visuales del workbench y pueden mostrar información diversa, como por ejemplo propiedades, lista o jerarquía de información, etc. Las modificaciones realizadas en una vista se salvan automáticamente. Su uso principal es proporcionar la navegación a través de la información del workbench. Las vistas poseen dos menús. Al primero se accede mediante el botón derecho sobre la pestaña de la vista. El segundo menú, denominado *pull-down*, es accedido mediante el icono , que muestra las operaciones específicas para el contenido mostrado.

Las vistas se pueden mostrar mediante el menú *Window* con la opción *Show view*, y la lista dependerá de la perspectiva mostrada. Para restaurar una perspectiva a su estado original de vistas se selecciona la opción *Reset Perspective* del menú *Window*.

Los editores están especializados en diferentes tipos de formatos o recursos, y suelen ya estar correctamente asociados a ellos. Si un recurso no posee un editor asociado, el entorno lanzará un editor externo previamente configurado en la plataforma. Cuando en la pestaña del editor aparece un "*", se está indicando que hay cambios pendientes de grabar (habrá que pulsar el icono de Save o Save All).

Los proyectos web

Los proyectos web J2EE se organizan mediante una estructura normalizada y que está formada por un grupo de directorios que albergan los diferentes archivos de la aplicación. En este momento bastará con saber que, partiendo del directorio raíz de la aplicación, cuelgan tres directorios importantes. Uno de ellos, *build*, recoge los archivos bytecode correspondientes a las clases que forman la aplicación. El directorio *src* contiene los archivos fuente de esas clases y el directorio *web* contiene a su vez otros dos directorios. En web se almacenan las páginas estáticas. En el directorio *web/WEB-INF* se guardan aquellos archivos que no pueden ser accedidos desde fuera de la aplicación, entre ellos, *web.xml* que es el archivo descriptor de despliegue de la aplicación y que proporciona los parámetros necesarios para que el servidor de aplicaciones pueda incorporar la aplicación en su marco de trabajo. Además también está *web/META-INF*, que contiene el archivo de manifiesto *manifest.mf*, que contiene metadatos relacionados con la aplicación y que pueden ser de interés para la JVM.

Estos proyectos pueden ser dinámicos o estáticos, en función del tipo de contenido. Las aplicaciones WEB se empaquetan en archivos WAR (Web Archive). Estos archivos se emplean para desplegar las aplicaciones web en los servidores de aplicaciones.

Todas las aplicaciones WEB disponen de un archivo descriptor de despliegue denominado *web.xml*, y en algunos casos, los servidores pueden requerir de otros archivos descriptores similares, además del mencionado.



El fotograma perdido, versión 0

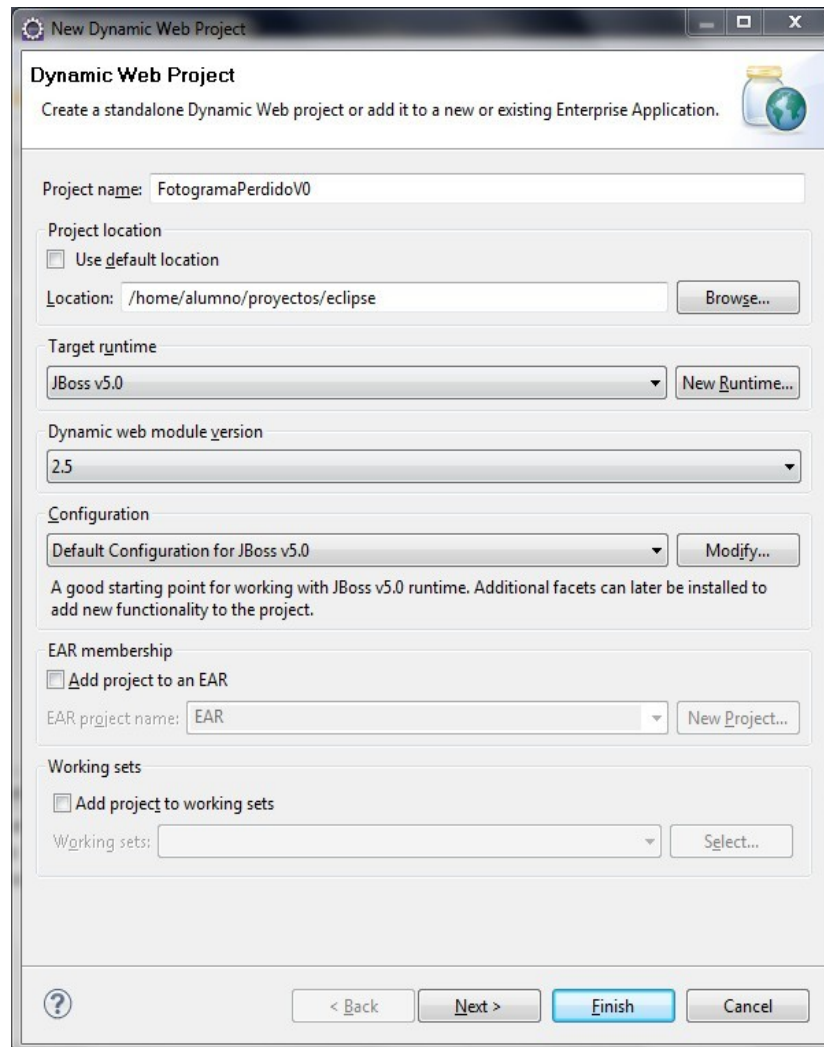
Ha llegado el momento de crear el primer proyecto web dinámico, y para ello crearemos la aplicación web que resuelve el enunciado básico del fotograma perdido.

Enunciado

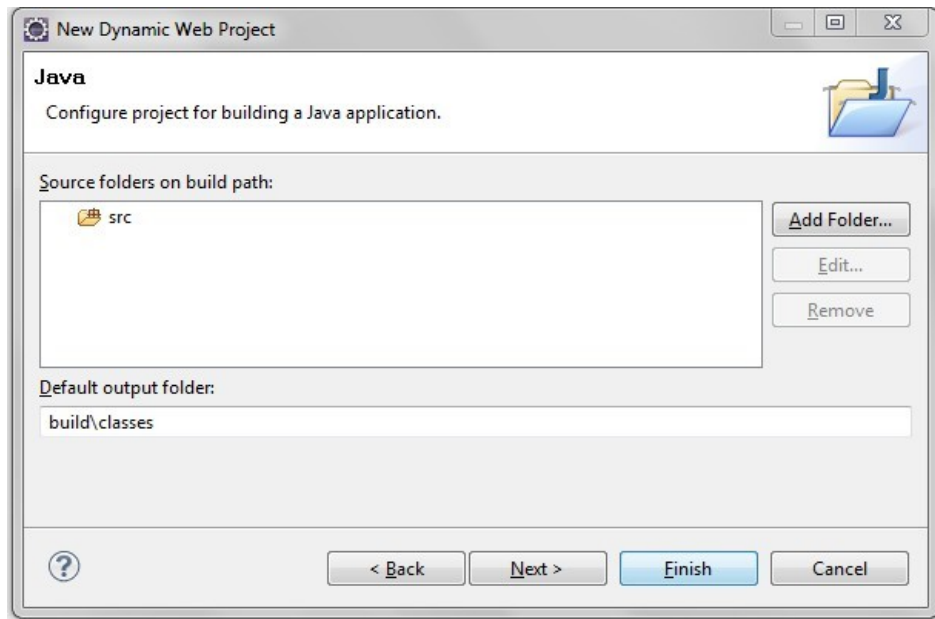
Se pretende diseñar una aplicación web que posibilite adivinar la película a la que pertenece un fotograma mostrado. El usuario introducirá la URL `http://<servidor>:<puerto>/fotogramav0/index.html`. La página mostrará el fotograma y le plantea al usuario que piense en una película. Pulsando un botón se le ofrecerá el título de la película. El archivo de imagen del fotograma se almacena en una carpeta denominada fotogramas con el nombre f1.jpg y se carga de forma estática en la página index.html. La respuesta se ofrece a través del archivo respuesta.jsp, que contiene el título de la película escrito de forma estática en el código (podía no haberse usado el tipo de archivo jsp).

Este proyecto se compone de una página estática con código HTML y otra, dinámica, con código JSP, si bien esta última no incorpora código Java alguno, por lo que puede considerarse como una página estática, aunque generará una clase de Java. Más adelante nos adentraremos en la tecnología JSP y se comprenderá mucho mejor cuando se aborde la versión 1.0 de este proyecto.

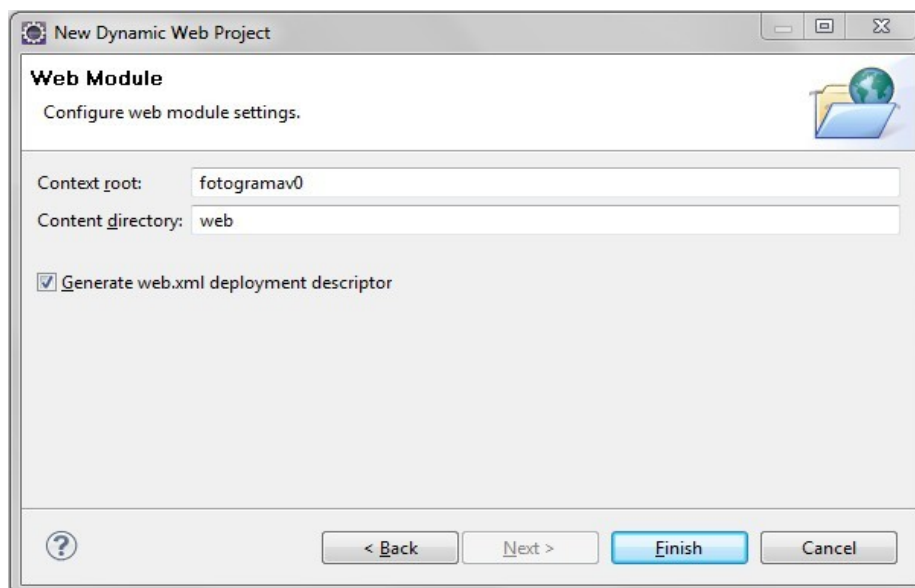
Por ahora comenzaremos creando un proyecto web dinámico. *File->New->Dynamic Web Project*, lo llamaremos *FotogramaPerdidoV0*, se ubicará en el espacio de trabajo considerado a lo largo de este manual, se escoge como target runtime (servidor de aplicaciones) a *JBoss v5.0*. Todas las demás se dejan con los valores por defecto. Este proyecto requerirá la perspectiva Java EE, que si no está seleccionada, la activará el propio entorno.



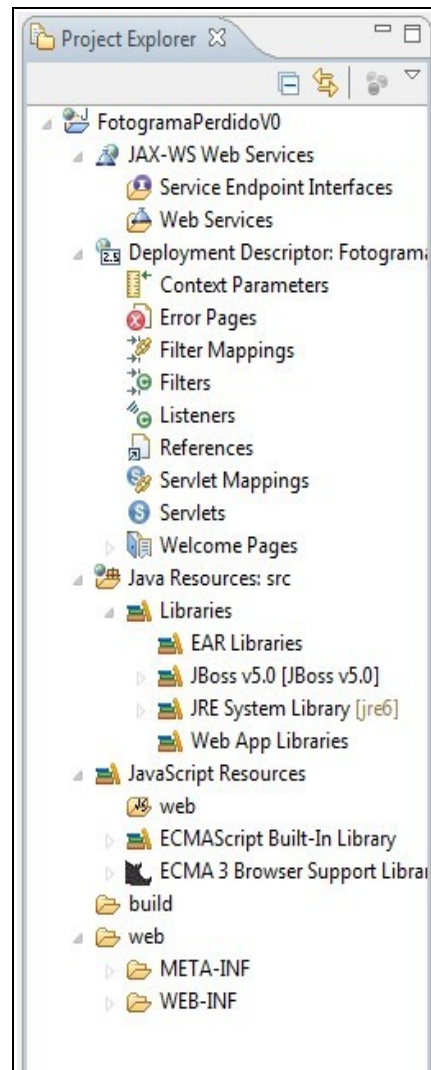
Después de pulsar *Next*, el asistente solicita que se especifiquen los directorios que contendrán los archivos fuente Java, que en nuestro caso sólo será el src en el que el servidor de aplicaciones almacenará las clases Java generadas a partir de las páginas JSP. Además, se indica también el directorio en el que se ubicarán los archivos de clase ya compilados.



Se pulsa el botón *Next*. Llega el momento de configurar el context root y el directorio web. En el primer caso será *fotogramav0* y en el segundo simplemente *web*.



Se pulsa el botón *Finish*. Llegados a este punto, ya tenemos el esqueleto del proyecto.



Por ahora sólo nos interesa el directorio *web* y sus subdirectorios *META-INF* y *WEB-INF*. Procedemos a añadir la página estática *index.html*, en la que se mostrará un fotograma de una película. La imagen se tomará de un directorio denominado *fotogramas*, y que se ubicará dentro del directorio *web* de la aplicación. Por tanto, lo primero será añadir ese directorio, que haremos desde el árbol de componentes del proyecto, seleccionando el directorio *web* y accediendo al menú contextual para añadir un nuevo directorio.

Ahora se almacenará en ese directorio el archivo de imagen del fotograma, que se denomina *f1.jpg*. Para ello, accedemos al menú contextual teniendo seleccionado el directorio *fotogramas*, y escogemos la opción *Import*. En el cuadro de diálogo que se abre se escoge *File System* (haciendo doble clic) y se selecciona el archivo *f1.jpg*, que estará en un determinado directorio. Una vez seleccionado el directorio, se marca el archivo a importar (pueden importarse más de uno) y se pulsa el botón *Finish*.

Ahora se incorpora al directorio *web* la página *index.html*, creándola tal como se ha visto en páginas anteriores y elaborando el código que se muestra a continuación.

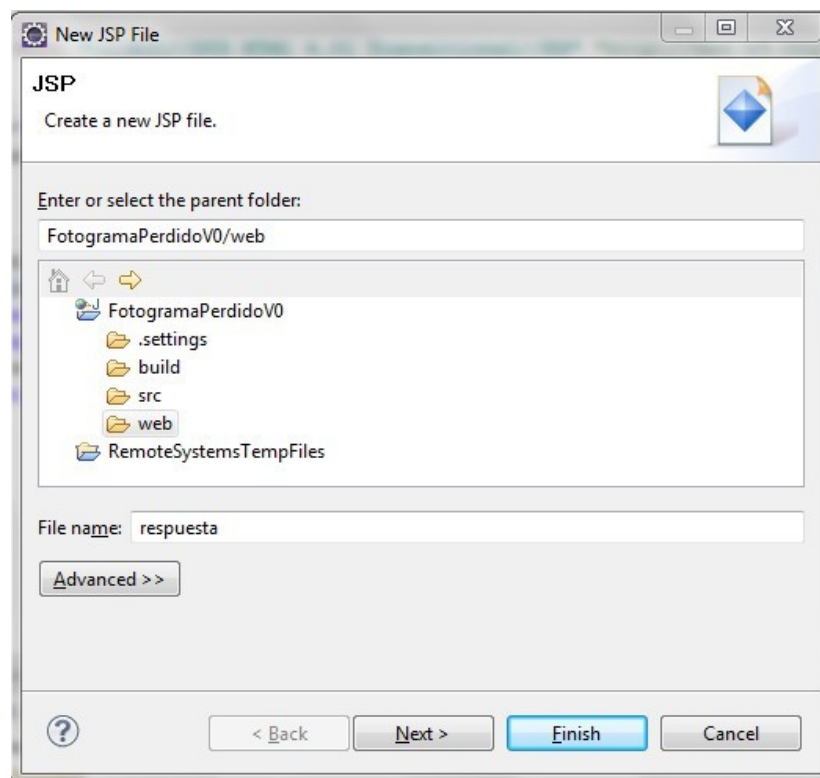
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```



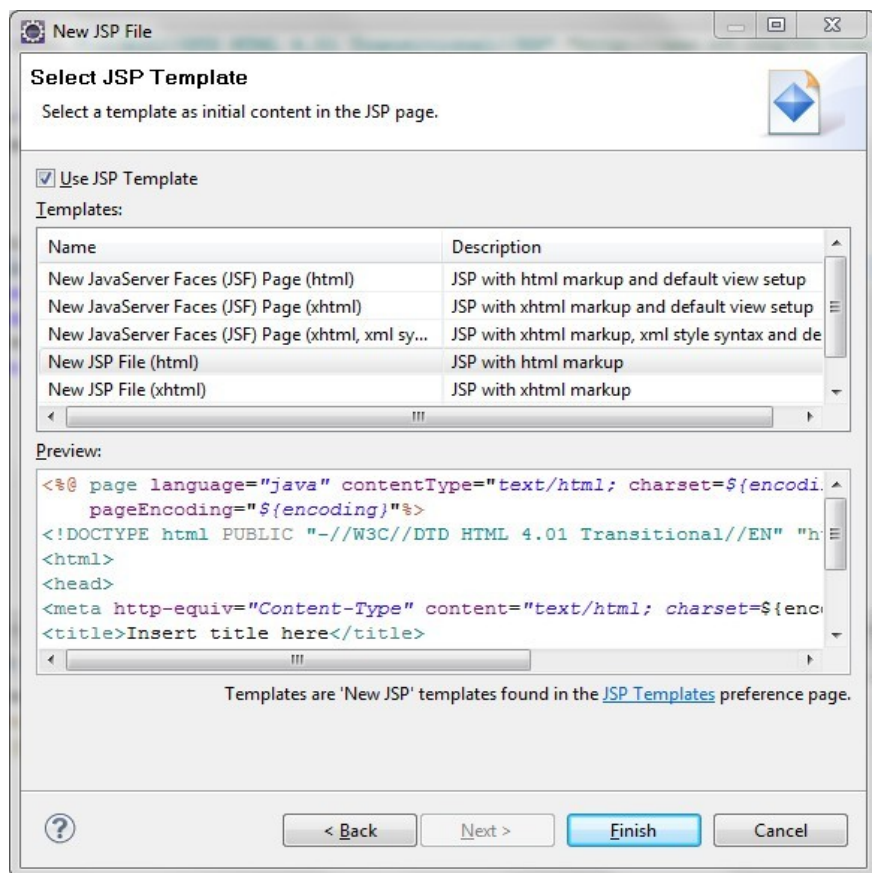
```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>El Fotograma Perdido versión 0</title>
</head>
<body>
<center>
El fotograma perdido.
¿Sabes a qué película pertenece este fotograma?<br>

<form action="respuesta.jsp" method="post">
Pulse el siguiente botón para saber la respuesta
<input type="submit" value="Respuesta"/>
</form>
</center>
</body>
</html>
```

Y se procede de igual manera con `respuesta.jsp`, teniendo en cuenta que se trata de una página JSP y que se escogerá la opción *New-> JSP File*.



Se pulsa el botón siguiente, y se mantienen las opciones que se muestran en la siguiente imagen.



Se pulsa el botón *Finish*. Se reelabora el archivo respuesta.jsp para que muestre el siguiente 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/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>El Fotograma Perdido v:0 - Respuesta</title>
</head>
<body>
La respuesta es: 2001 Una Odisea del Espacio
</body>
</html>
```

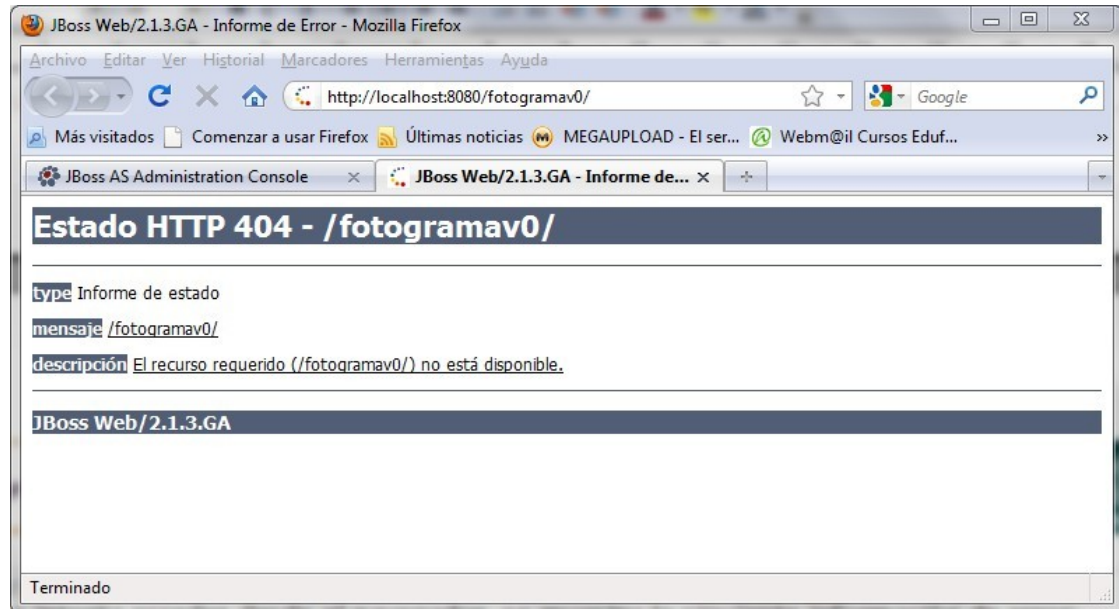
Ahora debe probarse la aplicación en JBoss. Para ello arrancamos el servidor de aplicaciones JBoss (asegurándonos de que no hay otro servidor atendiendo por el mismo puerto)²¹. Para arrancarlo nos ubicamos en la vista *Servers*, y allí, seleccionando JBoss, accedemos al menú contextual y seleccionamos la opción *Start*. En la vista *Console* se visualiza el log del proceso de arranque. Una vez arrancado comprobamos que está operativo correctamente, accediendo al navegador y enviando <http://localhost:8080>.

Ahora ejecutamos la aplicación mediante la opción *Run As->Run On Server* del menú

²¹ Existe la posibilidad de configurar el servidor para que escuche por otro puerto; lo veremos más adelante también.



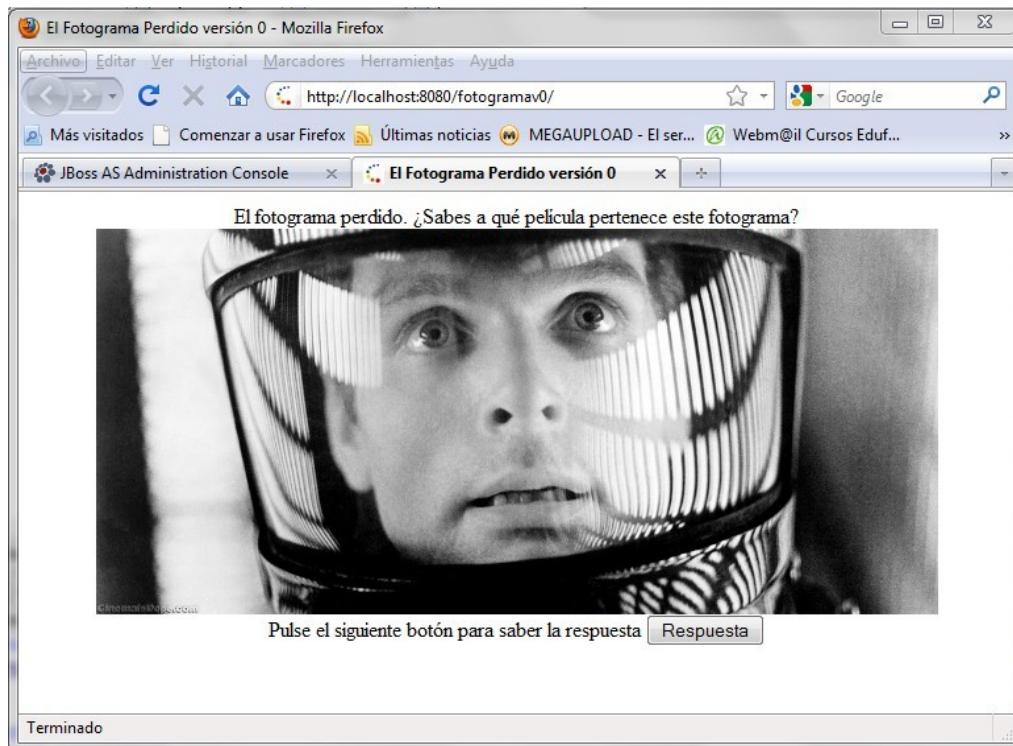
contextual de la misma y comprobamos que a pesar de haber establecido el context root en fotogramav0, cuando se intenta acceder desde el navegador, se muestra la siguiente información de error:



Esto es debido a que JBoss requiere de un archivo xml específico para ciertos parámetros de configuración de la aplicación, como es el context root. El archivo de configuración específico para JBoss es jboss-web.xml. Habrá que crearlo, en el directorio WEB-INF, e insertar las siguientes líneas:

```
<jboss-web>
  <context-root>fotogramav0</context-root>
</jboss-web>
```

Se vuelve a ejecutar el proyecto y comprobaremos que ya funciona correctamente.



Se recomienda tener configurado el entorno para que recompile automáticamente los proyectos (y los redespliegue) cada vez que se produzca una modificación en ellos. Para eso se debe tener marcada la opción *Build Automatically* del menú *Project*.

Ahora veremos qué ha sucedido con la página `respuesta.jsp`, ya que al pulsar el botón `Respuesta` de la página `index.html`, el servidor de aplicaciones, al recibir la petición sobre esa página, ha procedido a traducirla a un servlet Java, y a ejecutar (previa compilación a `.class`) el código correspondiente al método `service()` creado en ella. He aquí el código java del servlet generado:

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class respuesta_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();

    private static java.util.List _jspx_dependants;

    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.InstanceManager _jsp_instancemanager;

    public Object getDependants() {
        return _jspx_dependants;
    }

    public void _jspInit() {
        _el_expressionfactory =
            _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
        _jsp_instancemanager =
```



```

org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
}

public void _jspDestroy() {
}

public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;

    try {
        response.setContentType("text/html; charset=ISO-8859-1");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;

        out.write("\r\n");
        out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01  

Transitional//EN\" \"http://www.w3.org/TR/html4/loose.dtd\">\r\n");
        out.write("<html>\r\n");
        out.write("<head>\r\n");
        out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\">\r\n");
        out.write("<title>El Fotograma Perdido v:0 - Respuesta</title>\r\n");
        out.write("</head>\r\n");
        out.write("<body>\r\n");
        out.write("La respuesta es: 2001 Una Odisea del Espacio\r\n");
        out.write("</body>\r\n");
        out.write("</html>");
    } catch (Throwable t) {
        if (!(t instanceof SkipPageException)){
            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                try { out.clearBuffer(); } catch (java.io.IOException e) {}
            if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        }
    } finally {
        _jspxFactory.releasePageContext(_jspx_page_context);
    }
}
}

```

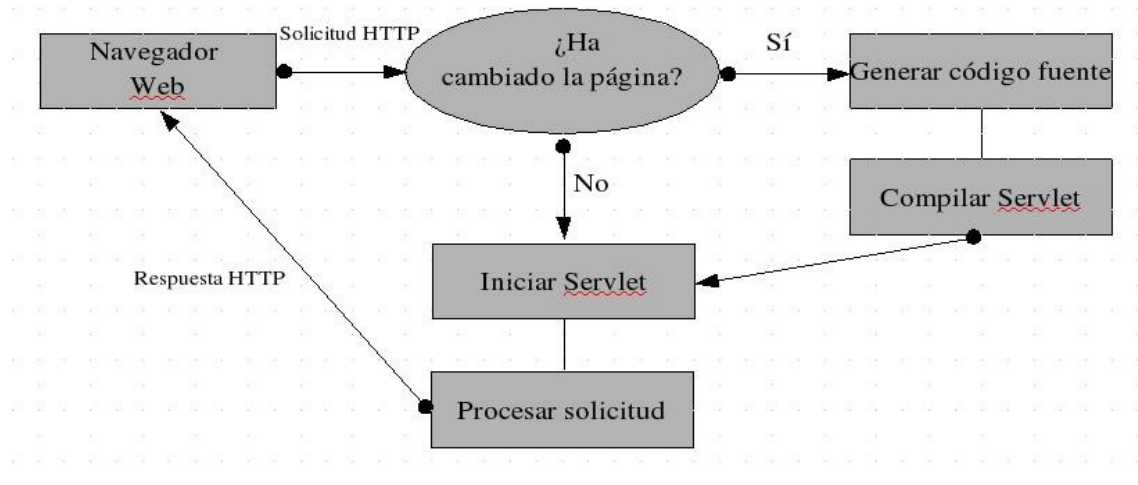
Este archivo se denomina `respuesta_jsp.java` y está ubicado en `JBOSS_HOME/server/default/work/jboss.web/localhost/fotogramav0/org/apache/jsp`. Como se observa, el nombre del archivo se corresponde con el nombre de la página JSP y además, el nombre de la clase, como todas las clases en Java, es el nombre del archivo, por lo que se da la curiosa circunstancia de que la clase tiene el nombre en minúsculas.

Este archivo es un servlet, por lo que conviene saber algunos conceptos fundamentales de los servlets antes de continuar.

Los servlets de Java



Cuando un usuario solicita una página JSP, mediante una solicitud HTTP, el contenedor reconocerá la extensión del documento solicitado (.jsp) y sabrá que tiene asociado un procesamiento determinado.



La primera vez que se solicita una página JSP, el contenedor leerá el archivo y generará código fuente para crear el servlet. El código se compila y se inicia el servlet. Una vez iniciado, administra las solicitudes del mismo modo que cualquier otro servlet. La única diferencia es que cada vez que se hace una solicitud de un archivo JSP, el registro del archivo se compara con otro registro del servlet. En caso de que el archivo JSP sea más nuevo, entonces tiene que volver a ser compilado en un servlet y después debe volver a iniciarse. La mayoría de contenedores JSP permiten compilar previamente los archivos JSP para evitar que el primer usuario que acceda a la aplicación soporte una lentitud manifiesta en el procesamiento.

Las páginas JSP poseen un ciclo de vida similar al de los servlets. Cuando se ejecuta por primera vez, se genera una llamada al método `_jspInit()`²². Una vez ejecutado, se llama al método `_jspService()`²³ para generar una respuesta a la solicitud. Este método contiene el código de servlet que aparecerá en HTML junto con el resultado del código Java contenido en el archivo JSP. Antes de que el servlet se deje de ejecutar desde el contenedor, se llama al método `_jspDestroy()`²⁴.

Los métodos `_jspInit()` y `_jspDestroy()` pueden anularse en la sección de declaración de la página JSP. El método `_jspService()` no puede anularse explícitamente, debido a que se trata de un método generado por el sistema que se corresponde con el cuerpo de la página JSP.

Veamos ahora un poco por encima el código generado.

```
package org.apache.jsp;
```

Indica el paquete al que pertenece la clase. Siempre se toma este paquete al generar automáticamente las clases servlets por parte del servidor de aplicaciones.

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
```

Librerías que se importan. En este caso son las que corresponden con las API Servlet y JSP.

```
public final class respuesta_jsp extends org.apache.jasper.runtime.HttpJspBase
```

²² Pertenece a la clase `javax.servlet.jsp.JspPage`.

²³ Que está definido en la clase servlet que se genera a partir de la página JSP.

²⁴ Que también pertenece a la clase `javax.servlet.jsp.JspPage`.

**implements org.apache.jasper.runtime.JspSourceDependent**

Clase que implementa el proceso asociado a la página JSP correspondiente. El nombre de la clase se forma partiendo del nombre de la página JSP seguido por “_jsp”. Este nombre de clase se nombra en minúsculas y hereda de *HttpJspBase* (clase abstracta) del paquete org.apache.jasper.runtime, proporcionado por el propio servidor de aplicaciones. Implementa la interfaz *JspSourceDependent*.

public void _jspInit()

Este método se ejecuta sólo cuando se crea la instancia de la clase y permite inicializar determinados parámetros en el servlet. Normalmente suele incluirse código para recuperar datasources (conexiones a bases de datos). Este método puede redefinirse en la propia página JSP mediante la etiqueta de declaración.

public void _jspDestroy()

Se invoca cada vez que el servidor de aplicaciones decide eliminar la instancia, normalmente por exceder el tiempo máximo de inactividad. Aquí puede incluirse código para liberar ciertos recursos, por ejemplo, los tomados en el proceso de inicialización con *_jspInit()*. Este método puede redefinirse en la propia página JSP mediante la etiqueta de declaración.

public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException, ServletException

Es el método que se encarga de conformar la página que recibirá el usuario, así como de llevar a cabo los procesos especificados en ella a través de las propias etiquetas JSP. Este método, dada su naturaleza, no puede sobrescribirse ni modificarse de forma explícita, sino que viene dado por el código de la propia página JSP. Este método se ejecuta cada vez que se recibe una petición sobre la página JSP. Recibe dos argumentos de entrada, dos objetos que representan la petición del usuario (request) y la respuesta del servidor (response) y que son proporcionados (creados) por éste, que es quien ejecuta este método. Puede lanzar dos tipos de excepciones, *IOException* para las situaciones en las que se producen errores de lectura o escritura sobre archivos u otras fuentes de datos enlazadas mediante flujos (y que pueden canalizarse mediante mecanismos de páginas de error que los gestionarán e intentarán recuperar a la aplicación), y *ServletException* para aquellos casos en los que la recuperación del servlet es prácticamente imposible.

response.setContentType("text/html; charset=ISO-8859-1");

Este método, aplicado sobre el objeto que representa la respuesta del servidor, establece el tipo de documento que se envía al usuario y que será establecido en la cabecera de la respuesta HTTP para que el navegador sepa interpretarlo adecuadamente.

out.write("\r\n");

Este método de out, y siguientes, implementan el mecanismo de respuesta al usuario, mediante el envío de las cadenas de texto que deberá procesar el navegador. Por tanto, mediante *out.write()* se envían todas las marcas HTML que conforman el documento que interpretará el navegador del usuario.