Índice

Contenido

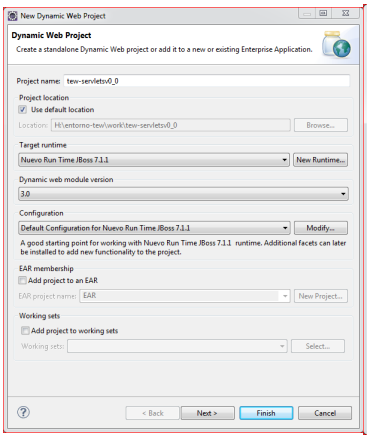
Sesión 5. Introducción al desarrollo de Servlets

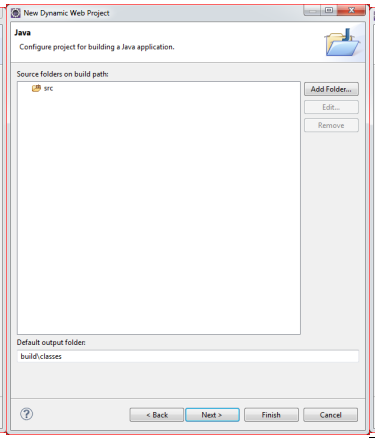
Desarrollo del HolaMundo!

En primer lugar, vamos a desarrollar el servlet HolaMundo que ante una petición HTTP, retorne una página HTML con el saludo de rigor. El servlet recibirá como parámetro el nombre del usuario, por lo que en lugar de invocarlo directamente, necesitaremos hacerlo desde una página HTML que contenga un formulario.

Para desarrollar el servlet:

1. Debemos crear un proyecto eclipse de tipo "Dynamic Web Proyect. Vete a File/New/ Dynamic Web Proyect" y crea el proyecto tew-servletsv0_0 con las características siguientes:





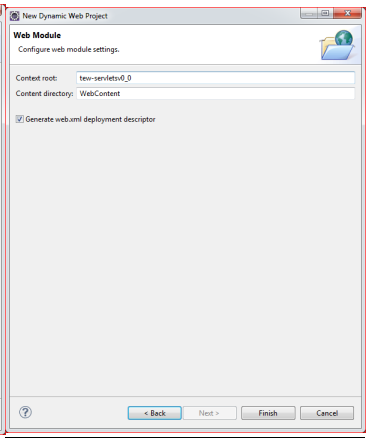


Fig. 1 **Fig. 2** **Fig. 3**

Hemos seleccionado en el primer formulario del asistente (Fig 1):

a. Target runtime: Nuevo Run Time JBoss 7.1.1, que ya deberías haber creado en la sesión 1 de la asignatura.
b. Dynamic web module versión: seleccionamos la versión 3.0, correspondiente a la versión de Servlets que soporta JBoss 7.1.1.
c. Configuration: Default Configuration for Nuevo Run Time JBoss 7.1.1.

En el resto de formularios las opciones son las que aparecen en las figuras 2 y 3. No olvides marcar el check-box "Generate web.xml" deployment descriptor en el formulario Fig. C.

2. Ahora que ya tenemos creado el proyecto, añadimos la clase `com.tew.Servlets.HolaMundoServlet` por medio del asistente de creación de Servlets (Ir a la opción File/New/Servlet). Debemos especificarle que extiende la clase `javax.servlet.http.HttpServlet` (ver Fig. 4).

    a. Fíjate en la figura 5, donde hemos seleccionado como nombre "interno" del servlet, "HolaMundo" (Name) y como patrón de URL, "HolaMundoCordial" (URlMapping).

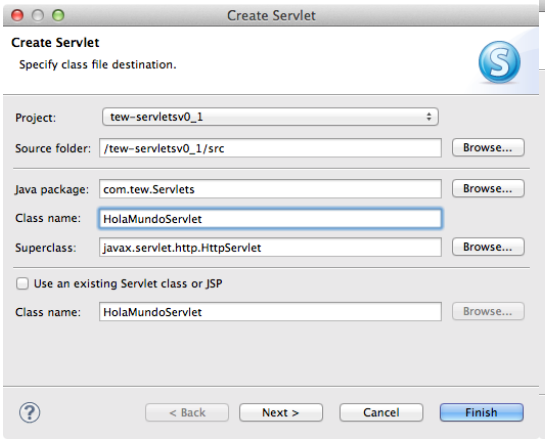


Fig. 4

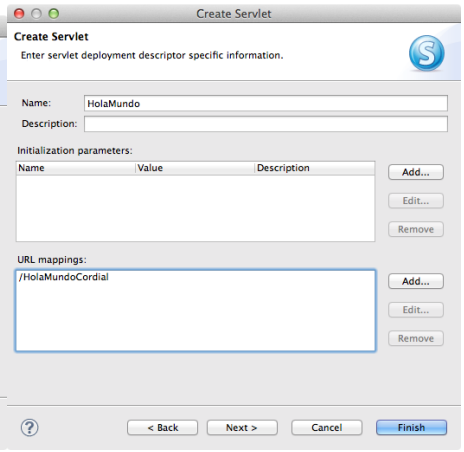


Fig. 5

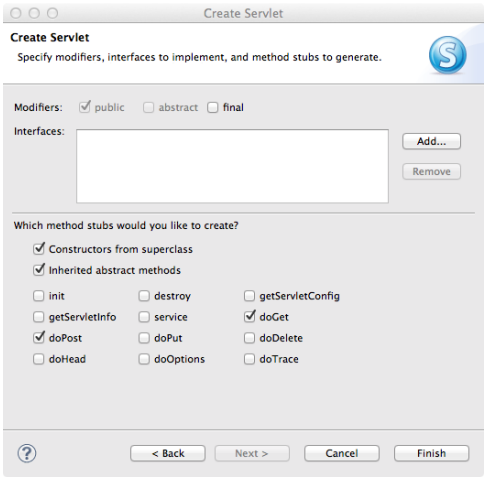


Fig. 6

3. Llegado este punto, tenemos ya el esqueleto del servlet que vamos a desarrollar. En una primera versión, lo invocaremos directamente desde el navegador, omitiendo la recepción de parámetros. La lógica del servlet debe ser implementada en los métodos `doGet()` y/o `doPost()`, dependiendo si queremos que nuestro servlet responda a peticiones de uno, otro o ambos tipos[1]. En primer lugar entonces, implementamos el `doGet(…)`.

```
public void doGet
(HttpServletRequest request,
HttpServletResponse response)throws IOException, ServletException {

      response.setCharacterEncoding("UTF-8");
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      out.println("<HTML>");
```

[1] En los servlets genéricos (cuando no trabajamos con protocolo http), el método que contiene la lógica del servlet es el método service(), que debemos sobrescribir. Esto, sin embargo, no es recomendable en los HttpServlet, puesto que el método service delega la petición en doGet o doPost dependiendo de la request.

```
      out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("Bienvenido a mi primera página web!");
      out.println("</BODY></HTML>");
}
```

4. El método `doGet()` será invocado por el `service()` cuando la petición que llegue sea de tipo GET. Haremos que el `doPost` delegue también en el `doGet`.

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)   throws IOException, ServletException {

      doGet(request,response );
}
```

5. Ya tenemos el servlet implementado. Eclipse señalará y propondrá los `imports` que son necesarios para poder compilar el código del servlet. Una vez añadidos, el servlet está terminado, aunque aún no podemos acceder a él desde el navegador. Para que el servidor de aplicaciones JBoss reconozca nuestro servlets tenemos dos posibilidades:
   a. En versiones anteriores a la especificación 3.0 de Servlets la única posibilidad para registrar un Servlet era empleando el descriptor de despliegue de la aplicación (el fichero WebContent/WEB-INF/`web.xml`).
      i. Debemos editar el fichero `web.xml` y añadir las líneas en negrita:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>tew-servletsv0_0</display-name>


  <servlet>
    <servlet-name>HolaMundo</servlet-name>
    <servlet-class>com.tew.Servlets.HolaMundoCordial</servlet-class>
  </servlet>


  <!-- Standard Action Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>HolaMundo</servlet-name>
    <url-pattern>/HolaMundoCordial</url-pattern>
  </servlet-mapping>

  <!-- Página de entrada por defecto -->
  <welcome-file-list>
        <welcome-file>index.html</welcome-file>
  </welcome-file-list>


</web-app>
```

ii. Desplegamos la aplicación y probamos el servlet accediendo desde el navegador mediante:

http://localhost:8080/tew-servletsv0_0/HolaMundoCordial

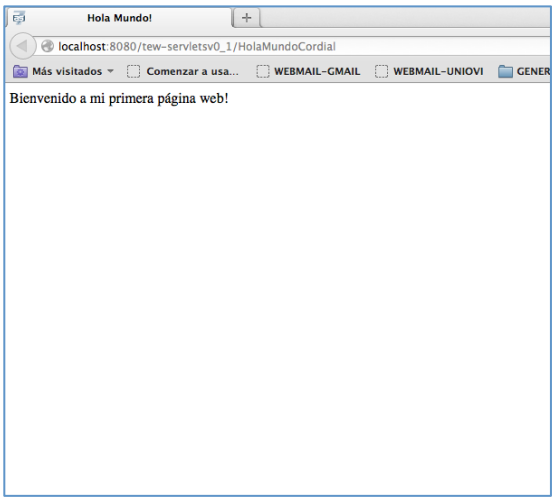
b. A partir de la versión 3.0 de la especificación de Servlets en JEE 6, es posible registrar un Servlet en un servidor de aplicaciones JEE empleando la anotación de clases: `@WebServlet`[2]

```
@WebServlet(name = "HolaMundo", urlPatterns = { "/HolaMundoCordial" })
public class HolaMundoServlet extends HttpServlet {

	private static final long serialVersionUID = 1L;

……
```

Por otro lado el descriptor de despliegue web.xml quedará así:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>tew-servletsv0_0</display-name>
<!-- Página de entrada por defecto -->
  <welcome-file-list>
     <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

6. En ambos casos, debemos obtener la página web generada por el servlet como respuesta. Examinar el código fuente de la misma desde el navegador mediante Ver/Código Fuente.



[2] http://docs.oracle.com/javaee/6/api/javax/servlet/annotation/WebServlet.html

7. ¡¡¡¡¡IMPORTANTE!!!!. La última versión liberada y compilada de JBoss[3] presenta ciertos problemas para mostrar los cambios cuando se realizan re-despliegues en caliente, tanto en servlets como en JSPs. De momento vamos a solucionar el problema de los servlets:
    a. Vete a la pestaña "Servers" y pincha doble sobre tu servidor JBoss.
    b. Ahora, en la sección "Publishing", asegúrate de marcar la opción (Fig. 7):
        i. Automatic publish when recources change
    c. Y en la sección Application Reload Behavior (Fig. 7):
        i. Modifica los cambios que harán que JBoss recargue un proyecto:
            1. \.jar$|\.class$

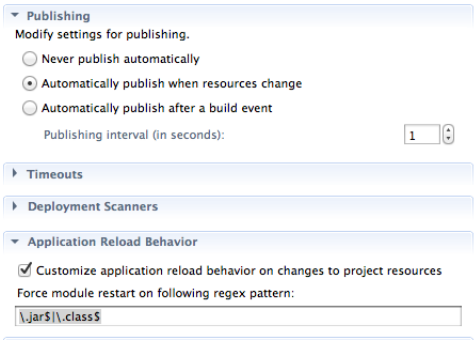


Fig. 7

8. Comprueba ahora que los cambios en el servlet son recargados correctamente por JBoss.
9. El siguiente paso consiste en personalizar el saludo para que el servlet pueda recibir el nombre del usuario como parámetro de entrada y emplearlo a la hora de generar la salida. Para ello, en primer lugar modificamos el código del servlet para que, en caso de que exista el parámetro nombre, lo recupere de la variable `request`:

```
String nombre = (String) request.getParameter("NombreUsuario");

response.setCharacterEncoding("UTF-8");
response.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<HTML>");
out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");
out.println("<BODY>");

if ( nombre != null ){
      out.println("<br>Hola "+nombre+"<br>");
}
out.println("Bienvenido a mi primera página Web!");
out.println("</BODY></HTML>");
```

10. Dado que nuestro servlet es capaz de servir peticiones GET y peticiones POST, ya estamos en disposición de comprobar su funcionamiento, puesto que en las peticiones GET los parámetros van concatenados al final de la URL.

[3] JBoss 7.1.1, http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.zip

http://<servidor>:[<puerto>]/directorio/../recurso?param1=valor&param2=...

11. Comprobamos entonces que funciona bien por GET desplegando de nuevo y poniendo en el navegador:

http://localhost:8080/tew-servletsv0_0/HolaMundoCordial?NombreUsuario=Fulanito

12. Para realizar la petición por POST, necesitamos crear un formulario HTML. Creamos index.html en el directorio `/WebContent` del proyecto:

```
<!DOCTYPE html>
<HTML>
<HEAD> <TITLE>Mi primer formulario</TITLE> </HEAD>
<BODY>
  <FORM ACTION="http://localhost:8080/tew-servletsv0_0/HolaMundoCordial"
        METHOD="POST">

   <H1>Saludador</H1>
   <HR><BR>
   <TABLE>
         <TR>
    <TD ALIGN="RIGHT">¿Como te llamas?</TD>
       <TD><INPUT TYPE="TEXT" NAME="NombreUsuario" SIZE="15"></TD>
      </TR>
      <TR>
            <TD><INPUT TYPE="Submit" VALUE="Saluda!"> </TD>
      </TR>
      </TABLE>
   </FORM>
</BODY>
</HTML>
```

13. Desplegamos y accedemos a http://localhost:8080/tew-servletsv0_0

Manejo de la sesión

Esta parte de la práctica está orientada a ver como se utiliza el objeto `HttpSession` asociado a la sesión del usuario para almacenar información relativa al estado de la misma. Para comprobarlo, vamos a partir del servlet `HolaMundo` que acabamos de implementar y lo vamos a modificar para que vaya almacenando los nombres de las personas que solicitan el saludo **desde una misma instancia del navegador.**

14. Editamos el servlet y modificamos su código de tal forma que al final de la página muestre la lista de personas ya saludadas durante la sesión del usuario actual. Para ello:
    a. Buscaremos en la sesión un atributo del tipo `java.util.Vector` que se llame **listado.** En caso de que no exista, lo instanciamos.

```
Vector listado =
      (Vector)request.getSession().getAttribute("listado");

if (listado == null){
    listado = new Vector();
```

```
        }
```

i. El método getSession() sin parámetros es equivalente getSession(true)[4].

b. Al recuperar el nombre de la persona a saludar desde la `request`, añadimos dicho nombre al vector.

```
        if ( nombre != null ){
              out.println("<br>Hola "+nombre+"<br>");
              listado.addElement(nombre);
        }
```

c. Establecemos el `Vector` como atributo en la sesión bajo el nombre de **listado**. En caso de que ya existiera, machacaría la referencia con la nueva.

```
request.getSession().setAttribute("listado",listado);
```

d. Al final de la página, listamos los nombres que contenga el `Vector`. Además, añadimos un enlace a la página del formulario para evitar tener que andar pulsando el botón de atrás del navegador.

```
        out.println("<br>");
        out.println("Contigo, hoy me han visitado:<br>");
        for ( int i = 0 ; i < listado.size() ; i++ ){
              out.println("<br>"+(String)listado.elementAt(i));
        }
        out.println(
        "<center><a href=\"index.html\">volver</a></center>");
```

15. Desplegar y probar la aplicación. Nótese que, al almacenar la lista de personas en la sesión, habrá una lista por cada usuario activo en la aplicación. Para comprobarlo, acceded a la aplicación del compañero mediante `http://<ip del compañero>:8080/tew-servletsv0_0`. La ip en Windows la podéis averiguar abriendo una ventana de comandos y ejecutando `ipconfig`.

16. El método doGet finalmente quedará de la siguiente manera:

```
public void doGet(HttpServletRequest request,
     HttpServletResponse response)
     throws IOException, ServletException {
…….

    out.println("<BODY>");

    @SuppressWarnings("unchecked")
    Vector<String> listado =
           (Vector<String>)request.getSession().getAttribute("listado");

     if (listado == null){
      listado = new Vector<String>();
     }

     if ( nombre != null ){
       out.println("<br>Hola "+nombre+"<br>");
```

[4] http://docs.oracle.com/javaee/1.2.1/api/javax/servlet/http/HttpServletRequest.html#getSession%28%29

```
            listado.addElement(nombre);
        }

        request.getSession().setAttribute("listado",listado);

        out.println("Bienvenido a mi primera página web!");
        out.println("<br>");
        out.println("Contigo, hoy me han visitado:<br>");
        for ( int i = 0 ; i < listado.size() ; i++ ){
              out.println("<br>"+(String)listado.elementAt(i));
        }
        out.println("<a href=\"index.html\">volver</a>");


        out.println("</BODY></HTML>");
}
```

17. Prueba a ver que pasa si introduces un nombre con caracteres especiales (acentos, ñ, etc).

Manejo del Contexto

A continuación vamos a implementar un contador de visitas al servlet. Como lo que queremos contar son las visitas totales, independientemente de la sesión a la que pertenezcan, no lo podemos implementar apoyándonos en la sesión del usuario, sino que lo haremos en el contexto de la aplicación. El objeto que representa el contexto del servlet lo obtenemos mediante el método `getServletContext()`, al cual tenemos acceso por heredarlo de la superclase del `HttpServlet`. El objeto de contexto que nos devuelve tiene los métodos `getAttribute(...)` y `setAttribute(…)` análogos a los del objeto de sesión.

1. Editamos el servlet y modificamos su código de tal forma que al final de la página muestre el número de visitas que ha recibido el servlet desde que se levantó el servidor.
   a. Buscaremos en la sesión un atributo del tipo `Integer` que se llame `contador`**.** En caso de que no exista, lo instanciamos con valor 0. Tiene que ser `Integer` y no `int` porque sólo podemos almacenar sublases de `Object`.

```
Integer contador = (Integer)
      getServletContext().getAttribute("contador");

      if ( contador == null ) {
            contador = new Integer(0);
      }
```

   b. Establecemos el contador como atributo del contexto bajo el nombre de `contador`. En caso de que ya existiera, sobreescribiría la referencia con la nueva.

```
getServletContext().setAttribute(
   "contador",
   new Integer(contador.intValue()+1)
);
```

   c. Al final de la página, mostramos el número de visitas totales, es decir, el valor de la variable `contador`.

```
out.println("<br><br>" + contador +" visitas");
```

2. Desplegar y probar la aplicación. En este caso, el contador se incrementará si accedemos desde la misma sesión (mismo navegador), o desde sesiones distintas. Para comprobarlo, acceded a la aplicación del compañero mediante `http://<ip del compañero>:8080/tew-servletsv0_0`.

Encadenamiento de servlets

Como ejemplo del uso de encadenamiento de servlets vamos a separar la implementación del servlet anterior en dos servlets diferentes: el primero será el encargado de actualizar los objetos sesión y contexto; el segundo, será el encargado de generar el contenido HTML que se devolverá al navegador.

1. Crear un segundo servlet en el proyecto denominado `com.tew.Servlets.HolaMundoVistaServlet`.
2. Añadir el etiquetado @WebServlet al nuevo servlet HolaMundoVista:

```
@WebServlet(name = "HolaMundoVista", urlPatterns = { "/HolaMundoVista" })
public class HolaMundoVistaServlet extends HttpServlet {

    private static final long serialVersionUID = 2L;

……
```

3. Modificar el servlet `HolaMundoServlet`:
   a. Para que no contenga código alguno de generación de contenido HTML.
   b. Añadiendo al final del método `doGet()` el siguiente código, el cual permitirá que el servlet actual delegue en el siguiente servlet de la cadena la generación de la respuesta.

```
…
RequestDispatcher dispatcher =
getServletContext().getNamedDispatcher("HolaMundoVista");

dispatcher.forward(req, res);
…
```

4. Implementar el servlet `HolaMundoVistaServlet` para que, accediendo a los objetos sesión y contexto, genere el contenido HTML que será devuelto al navegador.

Ejercicio Propuesto

Desarrollar un servlet `com.tew.Servlets.tienda.CarritoCompraServlet` que actúe de carrito de la compra. La página que genere debe contener un formulario HTML que muestre al usuario un `combobox` con al menos 10 productos y, bajo el formulario, el estado actual del carrito de la compra, mostrando cuantas unidades de cada artículo hemos introducido en el carrito. Cada vez que el usuario seleccione un producto y pulse el botón de "`submit`", el servlet mirará si el vector contiene el identificador del producto. En caso de contenerlo, incrementará las unidades en uno. Si no, lo inserta en el carrito.

Para implementar el carrito, se sugiere emplear una `java.util.HashMap` en lugar del `Vector` del ejemplo anterior. En la `HashMap` asociamos a cada clave (identificador de producto) un `Object` (un entero indicando el número de unidades). No necesitamos ninguna página HTML, sino que la única de la aplicación será la generada por el Servlet y que además contendrá

el formulario apuntando al propio servlet. La forma de introducir un combo en HTML dentro de un formulario se muestra a continuación. El valor del atributo que se recibe como seleccionado en la request es lo indicado en VALUE.

```
<SELECT NAME="productos" SIZE="1">
      <OPTION VALUE="010">La trampa</OPTION>
      <OPTION VALUE="345">Los pájaros</OPTION>
      <OPTION VALUE="554">Matrix reloaded</OPTION>
</SELECT>
```

Cuando tenga funcionando el ejercicio con un solo servlet, implemente una segunda versión del ejercicio en la que quede separada la manipulación del carrito de la compra y la visualización del mismo (en HTML) en dos servlets separados que colaboran con la utilización del encadenamiento de servlets visto en el último ejemplo.