Sesión 6. Introducción al desarrollo de Java Server Pages (JSP)

Desarrollo del HolaMundo!

Vamos a tratar de realizar las mismas tareas que se desarrollaron durante la práctica de Servlets, pero en esta ocasión mediante el empleo de JSP. Así, comenzaremos por desarrollar una página JSP que retorne el saludo **HolaMundo** cuando sea consultada. La página recibirá como parámetro el nombre del usuario, por lo que en lugar de invocarla directamente, necesitaremos hacerlo desde una página HTML que contenga un formulario.

Para desarrollar la JSP:

1. Debemos crear un proyecto eclipse de tipo "Dynamic Web Project". Vete a "File/New/ Dynamic Web Proyect" y crea el proyecto tew-jspv0_0 de la misma forma que lo hiciste en la sesión de Servlets.
2. Para crear una página JSP vete a la opción "File/New/JSP File" y crea una página **index.jsp**. Al seleccionar el formato de JSP selecciona el template "New JSP File (html)":

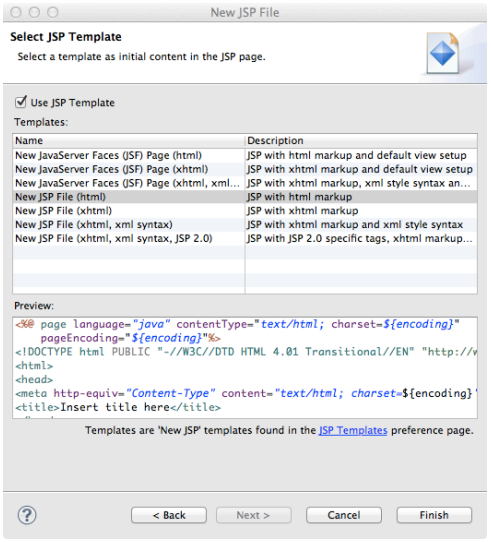


Fig. 1

3. En las páginas JSP, lo que se escribe directamente es el HTML, mientras que lo que escribimos entre marcas es el código java. En esta primera versión del HolaMundo, carente de lógica de negocio alguna, simplemente escribiremos código HTML. Nótese que la JSP no hace distinción entre método GET y método POST.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<!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=US-ASCII">
<title>Hola Mundo!!</title>
</head>
<body>
```

```
<h1>¡¡¡Bienvenido a mi primera página web!!</h1>
</body>
</html>
```

Ya tenemos la JSP implementada. En el caso de los servlets, lo que debíamos hacer ahora era darlo de alta en el descriptor de despliegue de la aplicación (web.xml) o anotarlo con @WebServlet. Las JSP no requieren ser dadas de alta en el descriptor para ser accesibles.

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

   http://localhost:8080/tew-jspv0_0/

5. ¡¡¡¡IMPORTANTE!!!!. La última versión liberada y compilada de JBoss[1] presenta ciertos problemas para mostrar los cambios cuando se realizan re-despliegues en caliente, tanto en servlets como en JSPs. Ahora nos toca solucionar el problema de los JSPs:

   a. Añadir lo que aparece en negrita al archivo: jboss\configuration\standalone\standalone.xml

   ```
   <subsystem xmlns="urn:jboss:domain:web:1.1" ……>
     <configuration>
     <jsp-configuration development="true"/>
     </configuration>
   ```

   b. Salvar el archivo **jboss-as-web-7.1.1.Final-RECOMPILE.jar** suministrado en el CV en el directorio jboss/modules/org/jboss/as/web/main

   c. Ahora abrir el archivo jboss/modules/org/jboss/as/web/main/module.xml, y editarlo sustituyendo en la configuración el archivo erróneo por el archivo corregido que te hemos suministrado.

   ```
   …..
   <!-- <resource-root path="jboss-as-web-7.1.1.Final.jar"/> --> <resource-root
   path="jboss-as-web-7.1.1.Final-RECOMPILE.jar"/>
   …..
   ```

   d. Rearranca el servidor JBoss para que esto tenga efecto.

6. Observa que la primera vez que se accede a la JSP después de cada cambio, el navegador tarda unos instantes es poder mostrarla. ¿A qué se debe? Se está produciendo la compilación de la página en el servidor. Esto ocurre **sólo** la primera vez que se accede. Las siguientes peticiones son servidas por un servlet que el contenedor habrá generado a partir de la JSP.

7. ¿Dónde está ese servlet? En el directorio **jboss/standalone/tmp/work/jboss.web/default-host** el contenedor de servlets despliega las aplicaciones web que recibe y, entre otras cosas, compila las JSPs a servlets. Busca el servlet generado por el contenedor a partir de la página **index.jsp del proyecto tew-jspv0_0** y examina su código fuente.

8. El siguiente paso consiste en personalizar el saludo para que la JSP pueda recibir el nombre del usuario como parámetro de entrada y emplearlo a la hora de generar la salida. Para ello, en

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

primer lugar modificamos el código de la página para que, en caso de que exista el parámetro nombre, lo recupere de la *request*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; utf-8">
<title>Hola Mundo!!</title>
</head>
<body>
        <%
         if (request.getParameter("NombreUsuario") != null) {
        %>
        Hola
        <%=request.getParameter("NombreUsuario")%>!
        <br>
        <%
                }
        %>
        Bienvenido a mi primera página web!
</body>

</html>
```

9. Dado que la JSP 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.

http://<servidor>:[<puerto>]/directorio/../recurso?parametro1=valor&parametro2=...

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

http://localhost:8080/tew-jspv0_0/index.jsp?NombreUsuario=Fulanito

Manejo de la sesión con JSPs

En la práctica anterior, ya vimos como manejar el objeto HttpSession con servlets, para almacenar el estado de la sesión del usuario actual. Para ponerlo en práctica con JSPs, vamos a implementar el ejercicio que se propuso al final de la sesión 2 pero con JSPs.

Se trata de construir una mini aplicación que nos permita, por medio de un combobox dentro de un formulario, añadir productos al carrito de la compra y visualizar el estado de dicho carrito. Concretamente, lo vamos a realizar con una sola JSP que visualizará el formulario, y a continuación, el estado del carrito de la compra.

11. En primer lugar, partiendo de la práctica en su estado actual, vaciamos el fichero **index.jsp** y lo dejamos en blanco para comenzar a implementar desde cero. Lo primero que vamos a incluir es el formulario html que nos muestre un combobox con películas y que asocie un código inventado a cada título, código que viajará en la request cuando el usuario pulse submit.

```
<html>
  <head>
    <title>Mi Tienda!</title>
  </head>
  <body>
    <center><h1>Tienda Virtual</h1></center>
    <br>
```

```
        <form action="index.jsp" method="post">
       <br>
       <table align="center">
          <tr>
            <td align="center">escoja el artículo que desea:</td>
          </tr>
        <tr>
            <td align="center">
              <select name="producto" size="1">
            <option value="010">la trampa</option>
            <option value="345">los pájaros</option>
            <option value="554">matrix reloaded</option>
            </select>
          </td>
        </tr>
          <tr>
            <td align="center">
              <input type="submit" value="añadir al carrito...">
            </td>
          </tr>
       </table>
    </form>
  </body>
</html>
```

12. Si desplegamos la aplicación tal y como la tenemos, visualizaremos un combo dentro del formulario que al pulsar "submit" nos llevará de nuevo a la misma página, pero sin ejecutar aún lógica alguna. El siguiente paso es abrir un scriplet JSP para poder incluir código en la página que se ejecute cada vez que el usuario acceda a ella. Esté código deberá:

    a. Recuperar el objeto almacenado en la sesión bajo la etiqueta de **carrito**, y hacer un cast a HashMap. Vamos a emplear este tipo de estructura para permitir la posibilidad de que el usuario se lleve más de una unidad de cada producto. En caso de que no existiera, lo instanciamos.

```
HashMap<String, Integer> carrito = (HashMap<String,
Integer>)request.getSession().getAttribute("carrito");
if ( carrito == null ) {
  carrito = new HashMap();
}
```

    b. Una vez obtenida la referencia al carrito de la compra, debemos añadir el código del producto que nos ha llegado en la request, en caso de que exista (en la primera carga de la página la petición no viene del formulario, y no habrá parámetro **producto**.

```
String producto = request.getParameter("producto");
if ( producto != null )
{
  Integer cantidad = (Integer) carrito.get(producto);
  if ( cantidad == null )
    cantidad = new Integer ( 1 );
  else
    cantidad = new Integer ( cantidad.intValue() + 1 );

  //Y añadimos el producto al carrito
  carrito.put(producto, cantidad);
}
```

13. A continuación, una vez añadido el producto al carrito, colocamos el carrito en la sesión. Esto sólo sería necesario si es la primera vez que se accede a la página, puesto que en caso

contrario, lo que hemos obtenido es una referencia a la HashMap que sigue siendo referenciada por el objeto HttpSession.

```
//Añadimos el carrito a la sesión
request.getSession().setAttribute("carrito",carrito);
```

14. Antes de continuar, y debido a que dentro del scriptlet estamos haciendo uso de un objeto de la java.util (java.util.HashMap), debemos importar la clase. ¿Podemos añadir una sentencia **import** directamente en el código del scriptlet? La respuesta es NO. El código que introducimos entre <% y %> formará parte de la implementación del método **service** del servlet una vez compilada la JSP, y dentro de un método no podemos hacer un import. La forma de hacerlo es mediante la directiva **page** de las JSPs. En la primera línea del index.jsp añadimos lo siguiente:

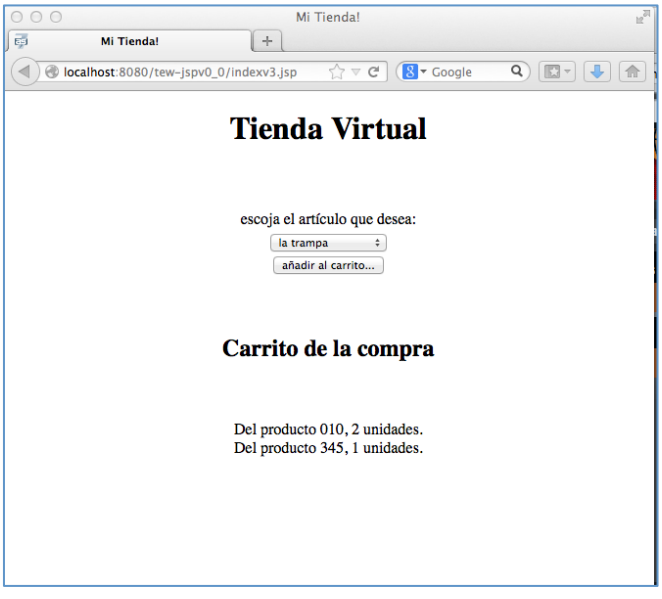
```
<%@ page language="java" import="java.util.*"%>
```

15. Ya sólo nos falta visualizar el estado del carrito de la compra. Vamos a hacer que aparezca a continuación del formulario. Justo antes del tag </BODY> añadimos:

```
<br><br>
<center><H2>Carrito de la compra</h2></center>
<br>
<center>
<%
    Set productos = carrito.keySet();
    Iterator iter = productos.iterator();
    while ( iter.hasNext() )
    {
      String elemento = (String)iter.next();
%>
     <br>Del producto <%=elemento%>,
       <%=(Integer)carrito.get(elemento)%> unidades.
<%
    }
%>
</center>
```

16. Nótese que para visualizar el código del producto y la cantidad de unidades nos estamos sirviendo de una expresión JSP: <%=...%>

17. Desplegamos la aplicación y probamos su funcionamiento.



18. El aspecto final de la página index.jsp será:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@ page language="java" import="java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Mi Tienda!</title>
</head>
<body>
    <center>
            <h1>Tienda Virtual</h1>
    </center>
    <br>

    <form action="index.jsp" method="post">
            <br>
            <table align="center">
                    <tr>
                            <td align="center">escoja el artículo que desea:</td>
                    </tr>
                    <tr>
                            <td align="center"><select name="producto" size="1">
                                    <option value="010">la trampa</option>
                                    <option value="345">los pájaros</option>
                                    <option value="554">matrix reloaded</option>
                            </select></td>
                    </tr>
                    <tr>
                            <td align="center"><input type="submit"
                                    value="añadir al carrito..."></td>
                    </tr>
            </table>
    </form>

    <!-- LÓGICA DE NEGOCIO -->
    <%
    //Comprobamos si existe el objeto "carrito" en sesión. Si no existe, lo creamos vacío.
Será de tipo HashMap
    @SuppressWarnings("unchecked")
    HashMap<String, Integer> carrito = (HashMap<String,
Integer>)request.getSession().getAttribute("carrito");
    if ( carrito == null ) {
    carrito = new HashMap<String, Integer>();
    }
    //Añadimos el producto recibido al carrito de la compra (en caso de que no sea nulo!)
    String producto = request.getParameter("producto");
    if ( producto != null )
    {
      Integer cantidad = (Integer) carrito.get(producto);
      if ( cantidad == null )
        cantidad = new Integer ( 1 );
      else
        cantidad = new Integer ( cantidad.intValue() + 1 );
      //Y añadimos el producto al carrito
      carrito.put(producto, cantidad);
    }
    //Añadimos el carrito a la sesión
    request.getSession().setAttribute("carrito",carrito);
```

```
    %>
    <br>
    <br>
    <center>
            <H2>Carrito de la compra</h2>
    </center>
    <br>
    <center>
            <%
    Set<String> productos = carrito.keySet();
    Iterator<String> iter = productos.iterator();
    while ( iter.hasNext() )
    {
      String elemento = (String)iter.next();
%>
            <br>Del producto
            <%=elemento%>,
            <%=(Integer)carrito.get(elemento)%>
            unidades.
            <%
    }
%>
    </center>
</body>
</html>
```

Manejo del Contexto en JSPs

Para probar el funcionamiento del contexto en JSPs vamos a complementar la página index.jsp con un contador que visualice las visitas totales a la página desde que se desplegó en el servidor. El contexto de las JSPs viene representado por la variable **application**. Así, para obtener un elemento del contexto empleamos **application.getAttribute(…)** y para colocarlo, su dual **application.setAttribute(...,...).**

```
<br>
<%
 Integer contador = (Integer)application.getAttribute("contador");
 if ( contador == null )
 {
        contador = new Integer(0);
 }
 application.setAttribute("contador",new Integer(contador.intValue()+1));
%>
<center><%=contador%> visitas</center>
```

Una vez implementado el contador, probarlo accediendo al ordenador del compañero, o abriendo varios navegadores para acceder a nuestro servidor. Cada navegador abrirá su propia sesión de usuario, pero el contador de visitas sumará el total de todas ellas. Finalmente, examinar el servlet generado por el contenedor a partir de la JSP desplegada, y observar en que se traducen las invocaciones al objeto **application** de la JSP.

Uso de acciones JSP estándar

Como ya sabes de las clases de teoría el uso de scriptlet en las páginas JSP para añadirles lógica no es la única forma. Con el proceso se madurez de esta tecnología se añadió la posibilidad de emplear TAGS xml (llamadas acciones) que representaban comportamiento. Gracias a ello las

páginas no son una mezcla de código Java y Html lo que permite mejor interacción entre los desarrolladores y los diseñadores. Existen tres tipos de acciones:

a) Estándar y JSTL, ambas incluidas en la especificación JEE

b) De terceras partes, a menudo distribuidas como parte de frameworks para permitir una integración sencilla con ellos. Por ejemplo las que acompañan a Struts2 o JSF2

c) Acciones propias que podremos desarrollar nosotros mismos para fines particulares. La especificación establece cómo implementarlas; es sencillo y muy similar a construir un servlet.

La referencia completa de las acciones estándar y de cómo construir las que necesitemos se puede encontrar en la especificación JEE. Aquí, para continuar con el ejercicio usaremos dos de las más populares <jsp:useBean> y <jsp:getProperty>.

La acción <jsp:useBean> permite recuperar un objeto de un contexto o, si no lo encuentra, instanciar uno nuevo y colocarlo en ese contexto. Se le puede especificar el contexto de la request, la sesión, la aplicación o la página (este último es por defecto). Un ejemplo de uso puede ser:

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

En este caso buscará en el contexto por defecto (page) un objeto llamado "name". Si no lo encuentra creará uno nuevo del tipo especificado por "class". <jsp:useBean> tiene más parámetros; revísalos en la documentación.

19. Para ampliar el ejercicio vamos a crear una clase contador específica, la colocamos en el contexto de la aplicación y la recuperaremos con <jsp:useBean>. Vete a "File/New/Class" y crea una clase según el asistente de la Figura (clase tew.beans.Counter):

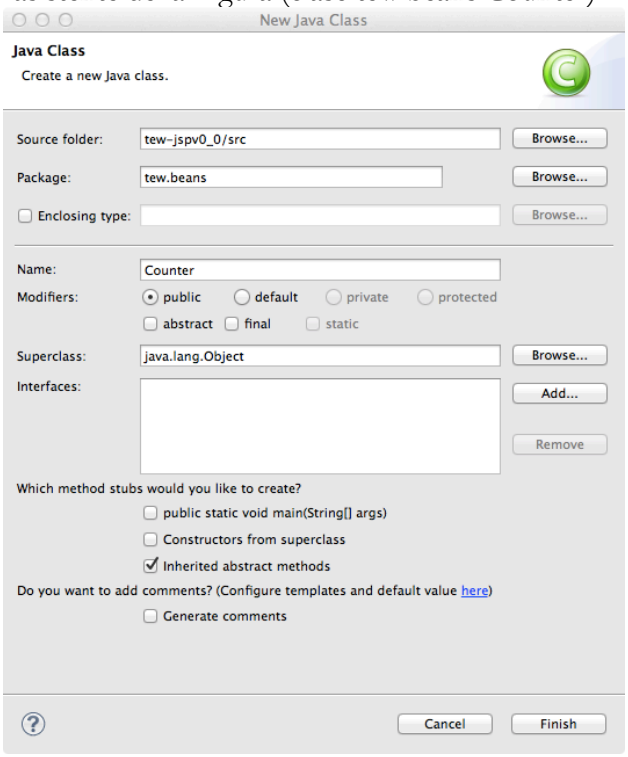


Fig. 2

Se trata de un JavaBean. Tendrá que atenerse a las condiciones de un JavaBean:

a) Constructor por defecto. Sino lo creamos nos lo creará el contenedor.
b) Getters y Setters con el sufijo de la propiedad que se desea leer o escribir capitalizada. Aunque la propiedad no tiene porqué existir como dato miembro con ese nombre. Como en el caso del ejemplo que te mostramos. En este ejemplo la propiedad es IncrementedValue, aunque después internamente hemos creado el dato miembro value.

```
public class Counter {
    private int value;
    public int getIncrementedValue() {
            return ++value;
    }
}
```

20. Usando la acción <jsp.getProperty> podremos obtener el valor del contador sustituyendo el código scriptlet que teníamos hasta ahora para contabilidad las visitas.

```
<jsp:useBean id="counter" class="tew.beans.Counter"
     scope="application"/>
```

```
<center>
 <jsp:getProperty property="incrementedValue" name="counter"/> visitas
</center>
```

Fíjate en la relación entre el nombre que se le asigna al atributo `property` de `<jsp:getProperty` y el del *getter* de la clase contador.

Uso de acciones de la JSTL

21. Ampliando un poco más el ejercicio podríamos sustituir el scriptlet final que renderiza el carrito por etiquetas de la sección "*core*" de la JSTL. Necesitamos sustituir el bucle, algo molesto por el continuo cambio entre Jsp y java, por la etiqueta `forEach`. Para ello necesitamos declarar que se usa la *taglib core* de JSTL.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

y sustituir un código por otro.

```
<ul>
<c:forEach var="entry" items="${carrito}">
<li>
<c:out value="Del producto ${entry.key}, ${entry.value} unidades"/>
</li>
</c:forEach>
</ul>
```

Uso de etiquetas de acción propias

Vamos a completar la sesión implementando y utilizando una etiqueta de acción propia. Dicha etiqueta deberá generar contenido HTML que dependerá de un valor numérico que se le pase como argumento:

- El argumento será el valor del contador de visitas a la aplicación.
- Si el argumento es múltiplo de 3, deberá generarse un contenido HTML como este:

  `<center>ENHORABUENA. HAS GANADO UN VIAJE!!!</center>`

- Si el argumento no es múltiplo de 3, deberá generarse un contenido HTML como este:

```
<center>Siga comprando ...</center>
```

Para ello tenemos que dar los siguientes pasos:

1. Desarrollar la clase que implementa la acción, tew.tags.PrizeTag, yendo a la opción "File/New/Class".

```
package tew.tags;
public class PrizeTag extends TagSupport {

        private String number;

        public void setNumber(String newName) { number=newName;}

        public String getNumber() { return number;}

        public int doEndTag() throws JspException {

                try {
                        JspWriter out = pageContext.getOut();
                        int theNumber=Integer.parseInt(number);
                    if (theNumber % 3 == 0)
                      out.println("<center>Enhorabuena ha ganado un viaje!!</center>");
                    else
                      out.println("<center>Siga comprando ...</center>");

                } catch (IOException e) {}
                return EVAL_PAGE;
        }
}
```

   Incluye los import necesarios para incluir las librerías empleadas en el código.

2. Describir la etiqueta en un fichero TLD. Para ello, crear un fichero denominado WEB-INF/tlds/`prize.tld` con el asistente yendo a la opción "File/New/File". Su contenido es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
    "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_2.dtd">
<taglib>
        <tlibversion>1.0</tlibversion>
        <jspversion>1.2</jspversion>
        <shortname>pri</shortname>
        <uri>tew.prize</uri>
        <tag>
                <name>prize</name>
                <tagclass>tew.tags.PrizeTag</tagclass>
                <bodycontent>empty</bodycontent>
                <attribute>
                        <name>number</name>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
```

```
</taglib>
```

3. Registrar el taglib (que realmente sólo contendrá una nueva etiqueta de acción) en el descriptor de despliegue web.xml:

```
  </welcome-file-list>
  <jsp-config>
  <taglib>
        <taglib-uri>tew.prize</taglib-uri>
        <taglib-location>/WEB-INF/tlds/prize.tld</taglib-location>
  </taglib>
  </jsp-config>
```

4. Finalmente quedaría por modificar nuestra página JSP para que utilizase la nueva etiqueta de acción:

```
<%@ taglib uri="tew.prize" prefix="pri" %>
…
<pri:prize number="<%= new Integer(counter.getValue()).toString() %>"/>
```

Fíjate que te dice el compilador que getValue() no está definida. Piensa por qué es y soluciona el problema.

Ejercicios propuestos

1. Eliminar completamente el scriptlet del ejercicio haciendo que un servlet reciba la petición, procese el carrito y pase la ejecución (forward) al JSP. Este recuperará los datos con <jsp:useBean> desde la petición.
   - Para hacer un *forward* desde un servlet a una página JSP:

```
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/index.jsp");

dispatcher.forward(request, response);
```

   - Para referenciar un JavaBean desde un servlet (por ejemplo, el JavaBean que cuenta las visitas utilizado anteriormente):
     i. Como su ámbito (*scope*) es la aplicación (*context*) completa:

```
tew.beans.Counter
    contador=(tew.beans.Counter)getServletContext().getAttribute("counter");
```

     ii. Si su ámbito fuera, por ejemplo, la sesión, habría que acceder de manera similar al objeto sesión para obtener/añadir el JavaBean.

2. Desarrollar una aplicación con JSPs que actúe de calculadora. La única página JSP recibirá tres parámetros. Los dos primeros, deberán ser números enteros (en forma de String, así que habrá que transformarlos en Integer), y el tercero un código que identifique una de las posibles operaciones de la calculadora. En la misma página tendremos el formulario que contendrá dos campos de texto (uno para cada término de la operación), y un combo box que permitirá escoger entre operación SUMA, PRODUCTO, RESTA o DIVISION. Cuando se haga submit, se mostrará el resultado debajo del formulario.