Tema 6
JSP y JSTL

Qué es JSP

JSP (**Jakarta Server Pages**) es una tecnología para crear páginas web dinámicas, incluyendo en ellas código Java. Se corresponden con la vista de una arquitectura MVC (los servlets no son cómodos para generar la salida)
Son simples archivos con extensión .**jsp,** localizados en la carpeta web/webapp del proyecto.

Ejemplo de jsp sencillo

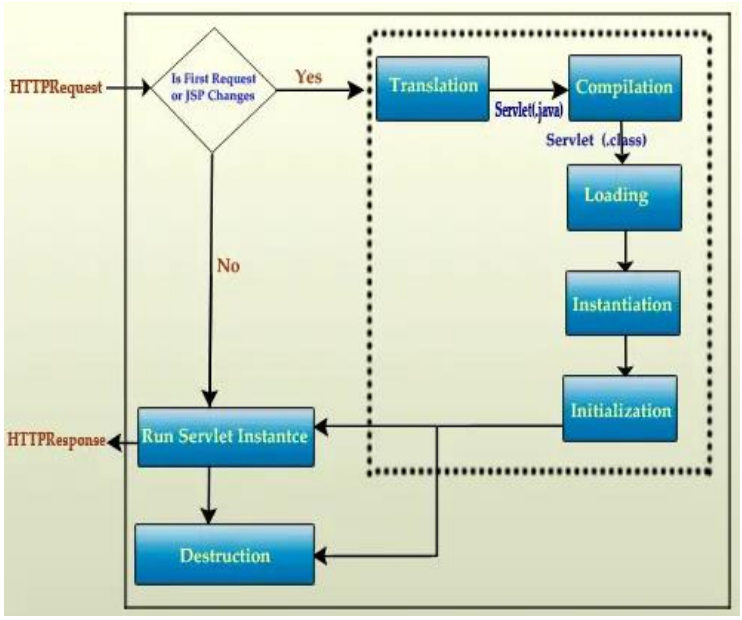
primera.jsp

```
<html>
<head><title>Mi primera página JSP</title></head>
<body>
      <h2> Hoy es: <%= new java.util.Date() %> </h2>
      <h2> Te conectas desde: <%= request.getRemoteAddr()  %> </h2>
</body>
</html>
```

Aunque JSP y servlets parecen tecnologías distintas, el servidor traduce cada JSP a 1 servlet y las siguientes peticiones de la página JSP se realizan a dicho servlet.
Aunque JSPs y servlets sean internamente similares, su estructura los hace propicios para una tarea:

- Los JSP se utilizan para dibujar la salida HTML.
- Los servlets se usan para tareas de control que generen poca salida: control del flujo de la app, redireccionamiento, control de parámetros, control de atributos, interacción con BDs, etc

Petición de una pagina JSP (traducción de JSP a Servlet)



- La 1ª vez que se solicita una página JSP, el servidor genera el servlet equivalente, lo compila y lo ejecuta.
- En las siguientes solicitudes al mismo jsp, la atención será atendida por dicho servlet.
- El servlet asociado a un jsp implementa la interface **HttpJspPage**, que es subinterface de jakarta.servlet.Servlet. Esta interface tiene los métodos ***jspInit, jspService*** *y* ***jspDestroy*** (parejos a *init*, *service* y *destroy* en 1 servlet)
- *El método* ***_jspService,*** equivalente al ***service (doPost,doGet)*** de los servlets, habitualmente, hará varios ***out.write(….)***

La anterior página ***primera.jsp*** podría generar un servlet con estructura similar al siguiente:

primera.jsp

```
<html>
<head>
<title>Mi primera página JSP</title>
</head>
<body>
      <h2> Hoy es: <%= new java.util.Date() %> </h2>
     <h2> Te conectas desde: <%= request.getRemoteAddr()  %> </h2>
</body>
</html>
```

Servlet generado

```
public final class primera_jsp extends org.apache.jasper.runtime.HttpJspBase….
{
          ……..
          …….
         public void _jspService(HttpServletRequest request,   HttpServletResponse  response)
                              throws java.io.IOException, ServletException {
              ……
              ……..
              JspWriter out = null;
              response.setContentType("text/html;ISO-8859-1");
              out.println("<html>");
              out.println("<head>");
              out.println("<title>Mi primera pagina JSP</title>");
              out.println("</head>");
              out.write("<body>\n");
              out.write("\t<h2> Hoy es: ");
              out.print( new java.util.Date() );
              out.write(" </h2>\n");
              out.write("       <h2> Te conectas desde: ");
              out.print( request.getRemoteAddr()  );
              out.write(" </h2>\n");
              out.write("</body>\n");   out.println("</html>");
              ……
               …….
        }
}
```

Elementos de JSP

En un **jsp** podremos insertar varios tipos de contenido dinámico java (no HTML):

1. **Directivas**
2. **Scripts java**
3. **Acciones <jsp>**
4. **Scripts java con el formato EL (Jakarta Expression Language)**
5. **Etiquetas JSTL**

1. Directivas de página <%@ %>

No producen contenido. Son instrucciones al contenedor sobre la compilación e influyen en la estructura del servlet generado. Su sintaxis es:

<%@ directiva atributo="valor" atributo1=”…” atributo2**=”…” %>**

Las directivas más conocidas son: **<%@page %> <%@include %> <%@tablib %>**

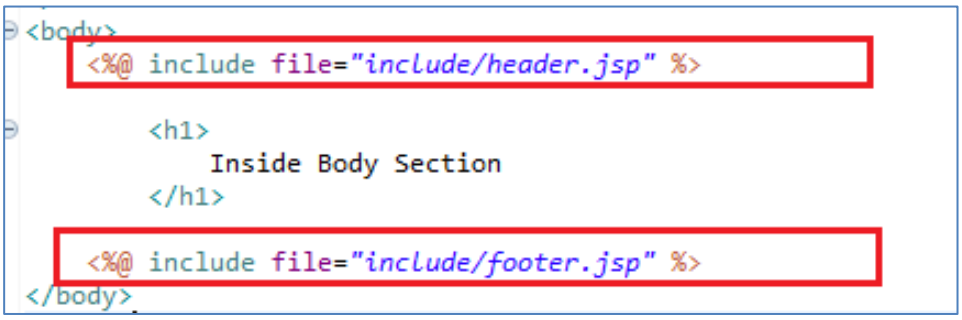
Directiva **<%@page %>**

Permite definir las características de la página: juego de caracteres a emplear, clases a importar, cómo gestionar los errores, etc.

Directiva **page**		
Atributo	**Significado**	**Ejemplo**
import	Equivale a la sentencia import de Java.	<%@ **page import**="java.util.Date, …." %>
contentType	Genera una cabecera HTTP Content-Type	<%@ **page contentType**="text/html" %>
pageEncoding	Define el juego de caracteres que usa el jsp	<%@**page contentType**="text/html" **pageEncoding**="UTF-8" %>
isThreadSafe	Si es false, implementará el interface SingleThreadModel : un único hilo atiende todas las peticiones (hasta no procesar 1 petición, no se admitirá otra). Suele dejarse el valor por defecto, que es true (permitir varios hilos simultáneos)	
session	Si es false, no se crea un objeto session . Suele dejarse el valor por defecto: true	
info	Define una cadena informativa sobre el jsp, que puede obtenerse a través del método ***getServletInfo***	<%@ **page info**="carro de la compra" %>
errorPage	Qué JSP debe procesar los errores no capturados en la página actual. Si no se define, el contenedor web se encarga de mostrar los mensajes de error.	<%@ **page errorPage**="error.jsp" %>
isErrorPage	Si es true, esta página actúa como página de error para otro JSP. Por defecto es false.	

Directiva **<%@include %>**

Sirve para incluir código en la página antes de que se realice la compilación del JSP.
Equivale a reemplazar la directiva con el contenido del fichero incluido.



Dado que los fragmentos de código incluidos pueden tener efecto sobre la página actual, se suele utilizar esta directiva para definir constantes, cabeceras y pies de página o contenido principalmente estático

El contenedor JSP no detectará de manera automática los cambios en los ficheros incluidos de esta manera, una vez haya compilado y generado el servlet.

Directiva **<%@taglib uri=”.....” prefix=”.....” %>**

Indica que la página va a utilizar librerías de etiquetas

Usaremos esta directiva para incluir las librerías de etiquetas de jstl

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

2. Scripts Java

Hay tres formas de insertar código Java en una página JSP: expresiones, scriptlets, declaraciones

Expresiones de la forma **<%= *expresión* %>**

La expresión se evalúa, se convierte a *String* y se escribe en la salida (el objeto *out*).
Equivale, por tanto, a **<% *out.write(*** expresión***); %>**

Ejemplos:

- Página generada a las <%= LocalDateTime.now().getHour() + ":" + LocalDateTime.now().getMinute() %>
- El dado ha arrojado un <%= Math.floor(1+Math.random()*6) %>
- *El color de pelo del pato <%= pato1.getNombre( ) %> es <%= pato1.getColor( ) %>*

Scriptlets de la forma **<% código %>:**

Permiten ejecutar código arbitrario Java, cuyo resultado no es necesario enviar a la salida.
Si desde un *scriptlet* se desea escribir en dicha salida, bastará con usar el objeto predefinido **out**

Ejemplo de scriptlet con **if** para hacer que ciertas partes de código HTML aparezcan o no:

```
<%          java.util.Calendar ahora = java.util.Calendar.getInstance();
            int hora = ahora.get(java.util.Calendar.HOUR_OF_DAY);
%>
<b> Hola mundo, <i>
<%          if ((hora>20)||(hora<6)) {                  %>
                    buenas noches
<%          }
            else if ((hora>=6)&&(hora<=12)) {           %>
                    buenos días
<%          }
            else {                                      %>
                    buenas tardes
<%          }                                           %>
</i> </b>
```

Declaraciones de la forma **<%! código %>**

Permiten definir variables o métodos del servlet
Se insertan como atributos del servlet y por tanto, son únicos y su valor es compartido por las distintas request.
Esto nos permite, por ejemplo, crear un contador de accesos a la página:

```
<%!         private int accesos = 0;            %>
<h1> Visitas: <%= ++accesos %> </h1>
```

Así pues, las variables declaradas en un jsp con la notación **<%! %>** tienen ámbito de contexto
Mientras que aquellas declaradas de manera simple (<% int x; %>), tendrán ámbito de página: se crean y destruyen por cada petición de la página.

Comentarios de la forma **<%- - Comentario - -%>**

El intérprete JSP ignorará todo lo contenido entre <%-- y --%>

Es más habitual incluir los comentarios habituales de Java dentro de un scriptlet:

- <% // comentario línea %>
- <% /* comentario varias líneas */ %>

Objetos implícitos de JSP

Son objetos instanciados automáticamente y disponibles en todas las páginas jsp

Objeto	Significado
request	Objeto HttpServletRequest asociado con la petición del cliente
response	Objeto HttpServletResponse asociado con la respuesta al cliente
out	Writer asociado a la respuesta, para enviar la salida al cliente.
session	Representa la sesión asociada a la petición. En JSP, las sesiones se crean automáticamente: este objeto está instanciado en todas las páginas jsp
application	Objeto de tipo ServletContext, es decir, el entorno o app web al que pertenece la página jsp. Es común a todos los servlets de la aplicación web.
config	Objeto de tipo ServletConfig, para leer parámetros de inicialización del fichero xml
page	Referencia al propio servlet (equivale a **this**).
exception	Representa un error producido en la aplicación. Solo es accesible si la página se ha designado como página de error (mediante la directiva page isErrorPage).

3. Acciones <jsp: >

Veremos las 2 más importantes : <jsp:include> y <jsp:forward>

Acción **<jsp:include>**

Esta acción incluye en una página la salida generada por otro recurso. Su sintaxis es:

<**jsp:include page**="*URL relativa*" />

El atributo page es la URL del recurso a incluir.

A diferencia de <%@include %>, el recurso se incluye cada vez que se pide la página principal, por lo que:

- Se utiliza para incluir recursos dinámicos.
- Cambios en la página incluida no obligan a recompilar la "principal".
- La petición se redirige a la página incluida, y la respuesta que genera se incluye en la generada por la principal.
- La página incluida y la página contenedora comparten la misma request, a la que se pueden agregar parámetros adicionales, mediante la etiqueta **<jsp:param**>:

```
<jsp:include page="cabecera.jsp">
        <jsp:param name="color" value="<%= valor%>" />
        <jsp:param name="tamanio" value="16" />
        . . . . . .
</jsp:include>
```

Acción **<jsp:forward>**

Esta acción se utiliza para redirigir la petición hacia otra página JSP, Servlet o HTML. Su sintaxis es:

<jsp:forward page="principal.jsp"/>

La salida generada hasta el momento por la página actual se descarta (se borra el buffer), y todas las líneas que aparezcan a continuación de esta acción no se consideran.

Al igual que en el caso de <jsp:include>, la página destino comparte request y response con la página original, y se permite añadir parámetros a la petición original para que los reciba la nueva página JSP:

```
<jsp:forward page="principal.jsp">
        <jsp:param name="privilegios" value="root" />
</jsp:forward>
```

4. Jakarta EL: Expression Language

JSP 2.0 introdujo el Expression Language, para simplificar la escritura de código java en las páginas JSP, promoviendo así la ausencia de scripts ~~<% ……%>~~

- Las expresiones del EL tienen la forma **${** expresión **}**

- Estas expresiones pueden ser incluidas como valores de atributos o combinadas con texto html

- En las expresiones (EL), podemos usar:
  - los operadores típicos +, -, *, /, %, >, >=, <, <=, ==, !=, &&, ||, !
  - El operador empty, que nos servirá para comparar a la vez con null y con cadena vacía

- Para acceder a un atributo de un objeto, podemos usar los operadores '.' Y '[]', de la forma:
  - **objeto.atributo** ó **objeto["atributo"]**

Ejemplo 1

```
<p> La suma de 3 y 4 es ${3+4} </p>
<p>La raíz cuadrada de 18 es ${Math.sqrt(18)} </p>
<p> ¿Es 3 mayor que 4? Eso es ${(3 > 4)? "verdad":"mentira"} </p>
```

El EL tiene acceso a los objetos implícitos de los JSP, a los que se añaden varias variables predefinidas :

Variable predefinida	Significado
pageScope	Para recuperar atributos de ámbito página
requestScope	Para recuperar atributos de ámbito request
sessionScope	Para recuperar atributos de ámbito sesión
applicationScope	Para recuperar atributos del contexto
param	Para recuperar parámetros de la petición
paramValues, header, headerValues, cookie ….que utilizaremos en menor medida	

Ejemplo 2

```
<%= request.getParameter("parametro") %>      equivale a    ${param.parametro}
<%= application.getAttribute("atributo") %>   equivale a    ${applicationScope.atributo}
```

- El EL es "null friendly", es decir, maneja los valores null sin propagar excepciones. Así,
  <%= request.getParameter("x") %> muestra null si el parámetro no está definido
  ${param.x}" muestra la cadena vacía si el parámetro x no está definido

- Cuando indiquemos un atributo por su nombre sin indicar "ámbito"/scope, se buscará en el contexto más cercano (pageScope). Si no existe, se buscará en el siguiente contexto más cercano y así sucesivamente hasta alcanzar el contexto más general (applicationScope)
  Por ejemplo, si tenemos 2 atributos con el mismo nombre "reason", uno en el request y otro en la sesión, la expresión ${reason} hará referencia al atributo "reason" ámbito request.

Ejemplo 3

Servlet que deja preparados:

- 1 atributo de contexto de tipo Point (con x e y)
- 1 atributo de sesión de tipo List<String>

```
class MiServlet extends HttpServlet
{

Point p=new Point(-3,4);
this.getServletContext().setAttribute("punto",p);
…….
……..

ArrayList<Strings> lista=
        new ArrayList<String>();
lista.add("Adolfo Ruiz");
lista.add("Ana Saez");
lista.add("Peio Perez");
request,getSession().setAttribute("amigos",lista);
……..
……
response.sendRedirect("el.jsp?error=0");
}
```

jsp que utiliza el **EL** para

- Acceder a los atributos "punto" y "amigos"
- Acceder al valor de 1 parámetro

el.jsp

```
<body>

<p> Punto coordenada x: ${applicationScope.punto.x} </p>
<p> Punto coordenada x: ${punto['y']} </p>
…….
……..
<p>Nº de amigos:  ${sessionScope.amigos.size()}  </p>
<p>1º amigo:    ${sessionScope.amigos.get(0)}   </p>
<p>2º amigo:    ${sessionScope.amigos[1]}   </p>
……..
……..
<p>El parámetro error tiene ${param.error} </p>

</body>
```

5. JSTL

En JSP es posible definir librerías de etiquetas personalizadas. Estas etiquetas son clases Java que heredan de determinadas clases y que se agrupan en librerías mediante un archivo descriptor TLD.
No construiremos librerías de etiquetas, pero sí usaremos la más común: **JSTL: Jakarta Standard Tag Library**

JSTL es un componente dentro de la especificación J2EE . Se trata de un conjunto de con etiquetas simples para escribir páginas JSP de modo más sencillo. Consta de los siguientes librerías/grupos de etiquetas:

- **core**: Con las funciones básicas para escribir, como bucles, condicionales, entrada/salida.
- **fmt**: Comprende la internacionalización y formato de valores como de moneda y fechas.
- **functions**: Con funciones para formateo de cadenas
- **xml**: Para procesamiento de xml.
- **sql**: Comprende el acceso a base de datos.

JSTL está perfectamente integrado con el **EL.**
Veremos su librería principal**: core**

Librería **core** de JSTL

En las páginas que la usen tendremos que incluir la directiva:

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

Esta librería implementa acciones de propósito general, como mostrar información, crear y modificar variables de distinto ámbito, tratar excepciones, etc. Veamos algunas de sus acciones

<c:out>

Muestra en la salida html el resultado de evaluar su atributo value. Ejemplos:

<c:out value="1+2+3" />	Equivale a	"1+2+3"
<c:out value="${1+2+3}" />	Equivale a	${1+2+3}
<c:out value="${param.nombreParametro}" />	Equivale a	${param.nombreParametro}
<c:out value="${sessionScope.atribDeSesion} " />	Equivale a	${sessionScope.atribDeSesion}

<c:set>

Equivale a setAttribute. Guarda información en una variable y tiene los siguientes atributos

- var: Nombre de la variable
- scope: Ámbito (page, request, session, application)

```
<c:set var="variableDePagina" scope="page" value="valor" />
<c:set var="variableDeSesion" scope="session" value="valor variable" />
<c:set var="variableDeAplicacion" scope="application" value="otro valor" />
<br>${variableDePagina}
<br>${variableDeSesion}
<br>${variableDeAplicacion}
```

<c:if> *(No válido para if-else)*

Procesa el cuerpo de la etiqueta si la condición es cierta. La condición se indica en el atributo *test*

```
<c:if test="${applicationScope.totalVisits == 1000000}" >
                    Eres el visitante 1.000.000. Enhorabuena!
</c:if>


<c:if test="${not empty param.error}" >
        <div class='alert-danger'>
                        <strong><c:out value="${param.error}"/></strong>
        </div>
</c:if>
```

<c:choose>

Procesa condiciones múltiples. Equivale a la sentencia ***switch-case***. (O a una sentencia if-else si sólo contiene un bloque *<c:when>* + un bloque *<c:otherwise>*)

```
<c:choose>
        <c:when test="${item.tipo == 'book'}">
        ...
        </c:when>
        <c:when test="${item.tipo == 'electronics'}">
        ...
        </c:when>
        <c:when test="${item.tipo == 'toy'}">
        ...
        </c:when>
        <c:otherwise>
        ...
        </c:otherwise>
</c:choose>
```

<c:forEach>

Se utiliza para iterar sobre una colección (lista, mapa, conjunto,….)
Consta de los siguientes atributos:

- **items**: nombre de la colección sobre la que iterar. Pueden ser arrays, listas, mapas, enumeraciones, ….
- **var**: nombre simbólico de la variable donde se guarda el elemento en curso
- **varStatus**: guarda el estado de la iteración. Se trata de un objeto con varias propiedades interesantes acerca de la iteración actual
    - **index**: Indice del elemento en curso (comienza en 0)
    - **count**: Número de iteración (comienza en 1)
    - **first**: Booleano que indica si es la primera iteración
    - **last**: Booleano que indica si es la última iteración

```
<table>
<c:forEach items="${sessionScope.arrLibros}" var="libro">
      <tr>
        <td>${libro.titulo}</td>
        <td>${libro.autor.nombre}</td>
        <td>${libro.autor.nacionalidad}</td>
      </tr>
</c:forEach>
</table>
```

Recorre arrayList de objetos Libro de la sesión

```
<ul>
        <c:forEach items="${mapaPaises}" var="pais" >
                <li> Country name: ${pais.key}  - Capital: ${pais.value}  </li>
        </c:forEach>
</ul>
```

Recorrido de HashMap (Pais → Capital) existente en el ámbito +cercano

```
<c:forEach items="${arrLibros}" var="libro" varStatus="estado">
            <a href="${libro.isbn}"> ${libro.titulo}</a>
             ${ estado.last ? " : '<hr/>'}
</c:forEach>
```

Recorrido de array de Libros, haciendo uso de varStatus para controlar en qué iteración estamos

<c:redirect>

Redirige a la dirección especificada en el atributo url.
El navegador generará una nueva petición (equivale a ***response.sendRedirect()***)

```
<c:if test="${param.clave!='damocles'}">
        <c:redirect url="login.jsp" />
</c:if>
```

<c:catch>

Lo utilizaremos para capturar excepciones sin que se aborte la ejecución de la página al producirse un error. El atributo *var* guarda el nombre de la variable que almacenará información de la excepción (null si no se ha generado la excepción)

```
<c:catch  var="error01">
        <%=  Integer.parseInt(request.getParameter("parametro"))  %>
</c:catch>
<c:if  test="${error01!=null}">
        Se produjo un error: ${error01}
</c:if>
<form>
        <input  type="hidden" name="parametro" value="prueba"  />
        <input  type="submit" name="submit" value="Enviar 'prueba'"  />
</form>
<form>
        <input  type="hidden" name="parametro" value="1234"  />
        <input  type="submit" name="submit" value="Enviar 1234"  />
</form>
<form>
        <input  type="submit" name="submit" value="No enviar parametro"  />
</form>
```

Librería **fn** (functions)

Incluyen funciones para el formateo de Strings y requiere del siguiente taglib:

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

Algunas de sus funciones son:

fn:contains, fn:trim, fn:replace, fn:split, fn:join, fn:substring, fn:toLowerCase, fn:toUpperCase, etc

Ejemplos

```
<c:set var="array_palabras" value="${fn:split(saludo,' ')}" scope="page"/>
<c:out value="${fn:length(array_palabras)} palabras" />

<c:if test="${fn:contains(saludo,'agur')}">
        <p>Es una despedida </p>
</c:if>

 <c:set var="numbers" value="One,Two,Three,Four,Five" />
 <c:set var="splitNumbers" value="${fn:split(numbers,',')}" />
 <c:set var="joinedNumbers" value="${fn:join(splitNumbers,' ')}" />
```

Librería **fmt (format)** de JSTL

Con etiquetas para formatear números, porcentajes, monedas, etc. Requiere del siguiente taglib

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

Ejemplos

```
<c:set var="amount" value="9850.14115" />

<fmt:formatNumber value="${amount}" type="currency" /></p>
<fmt:formatNumber type="number" maxFractionDigits="2" value="${amount}" />
<fmt:formatNumber type="number" pattern="###.###$" value="${amount}" /></p>

<c:set var="porcen" value="0.75"/>
<fmt:formatNumber value="${porcen}" type="percent"></fmt:formatNumber></p>

<fmt:formatDate value="<%= new Date() %>"  type="time"/>

<c:set var="today" value="28-03-2018"/>
<fmt:parseDate value="${today}" var="parsedDate" pattern="dd-MM-yyyy"/>
```

Java Bean. Concepto

Los javabeans son clases Java que representan objetos ordinarios (POJOs), que se comportan como cajas negras y deben cumplir una serie de requisitos

Son otra forma de separar la lógica de presentación de datos de una aplicación y la lógica de negocio, siendo los beans parte de la lógica de negocio.

Requerimientos mínimos de la clase asociada a un bean:

- Debe ser pública
- Debe ser Serializable
- Debe tener, al menos, un constructor público sin argumentos.
- Sus atributos son privados
- Debe tener getters y setters

Ubicación de los JavaBeans: por ser clases Java, deben estar situados en la misma zona que los servlets.

Ejemplo de clase de tipo Bean

```
public class Aficion implements Serializable{
    private String nombre;
    private int riesgo;

    public void setNombre(String nombre){
            this.nombre = nombre ;
    }
    public void setRiesgo(int riesgo){
            this.riesgo=riesgo ;
    }
    public String getNombre( ){
            return nombre ;
    }
    public int getRiesgo( ){
            return riesgo ;
    }
    public String getComentarioAficion( ){
            if (nombre.equals("Tumbismo"))
                    return "Eres el monstruo del sofa";
            else if (nombre.equals("Buceo"))
                    return "Cuidado con los pulpos";
            else if (nombre.equals("Parapente"))
                    return "Cuidado no te estrelles";
            else return "Aficion saludable";
    }
    public String despedida( ){
            return "Adios";
    }
}
```