

# Programación en J2EE

## Componentes Web JSP.

## JavaBeans

### INDICE:

Programación en J2EE .....	1
Componentes Web JSP. JavaBeans .....	1
1   Objetivos .....	2
2   Arquitectura J2EE: Componente Web JSP .....	3
3   Ciclo de vida de una página JSP .....	5
4   Contenido dinámico .....	6
5   Directivas JSP .....	7
5.1   La directiva page .....	7
5.2   La directiva include JSP .....	8
5.3   Ejemplo desde NetBeans.....	8
6   Scripts JSP.....	11
6.1   Expresiones JSP .....	11
6.2   Scriptlets JSP.....	11
6.3   Declaraciones JSP .....	12
6.4   Ejemplo desde NetBeans.....	12
7   Variables Predefinidas.....	14
7.1   Tipos de variables JSP predefinidas.....	14
7.2   Ejemplo desde NetBeans.....	15
8   Acciones JSP .....	16
8.1   Tipos de acciones JSP .....	16
8.2   Ejemplo desde NetBeans.....	18
9   JavaBeans y Formularios .....	20
9.1   Formularios con JSP.....	20
9.2   Formularios con JSP y JavaBeans.....	20
9.3   Ejemplo desde NetBeans.....	20
10  Ejercicios.....	29
10.1  Ejercicio 1 (Entregado por el profesor).....	29
10.2  Ejercicio 2 (2 puntos) .....	29
10.3  Ejercicio 3 (2 puntos) .....	29
10.4  Ejercicio 4 (3 puntos) .....	30
10.5  Ejercicio 5 (3 puntos) .....	31

# 1 Objetivos

Los objetivos de esta sesión son:

- Estudiar los componentes Web JSP de la arquitectura J2EE
- Comprender la relación estrecha entre JSP y Servlets
- Conocer las posibilidades que nos ofrecen las páginas JSP para la generación de contenido dinámico
- Estudiar los elementos JSP: Directivas, Scripts, Acciones y Variables predefinidas
- Aplicar los nuevos conocimientos adquiridos para completar los formularios HTML vistos en la anterior sesión y darles más sentido que enviar los datos a un mail
- Comprender las ventajas de la separación de generación de contenidos y su presentación
- Formar al alumno en el uso de NetBeans para la creación de proyectos Web, formularios, JavaBeans y páginas JSP.



- Independencia de la plataforma
- Rendimiento optimo
- Separación de la lógica de la aplicación de la presentación de los datos
- Uso de componentes (JavaBeans)

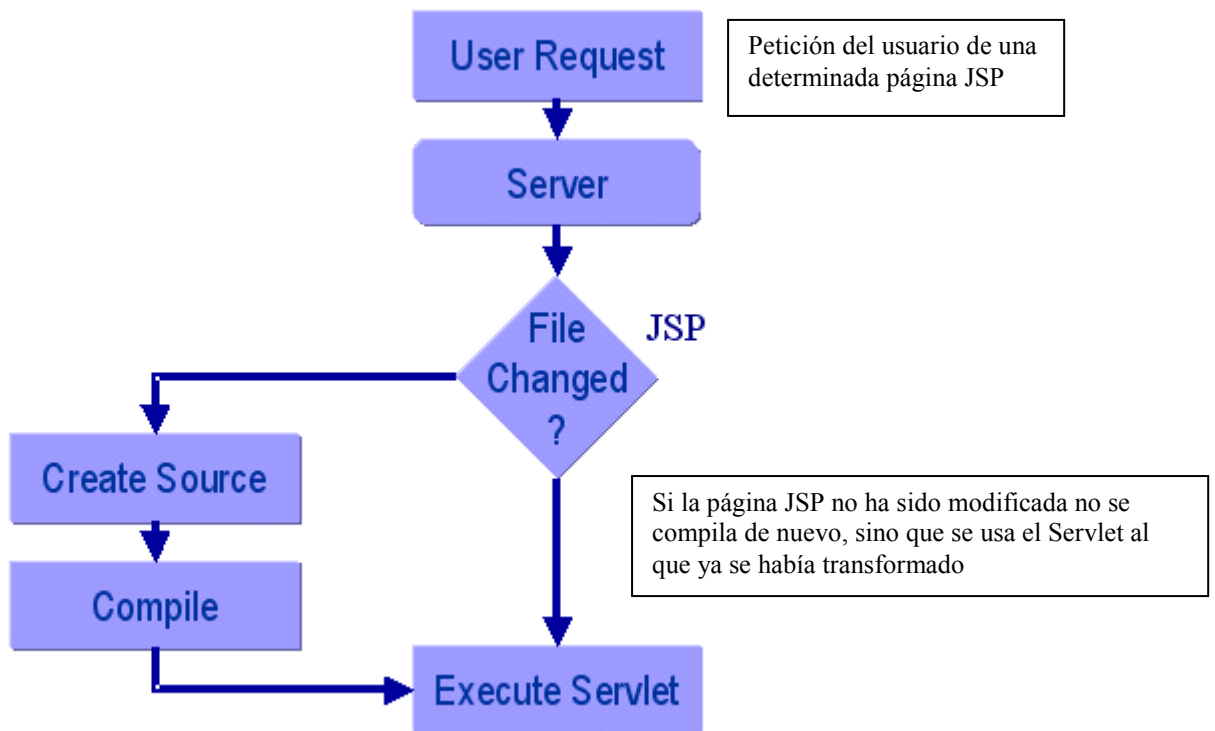
### 3 Ciclo de vida de una página JSP

Generaremos archivos con extensión **.jsp** que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor

La tecnología JSP es una extensión de la tecnología Servlets (la estudiaremos con detenimiento en la siguiente sesión), los cuales son aplicaciones 100% Java que corren en el servidor: Un Servlet es creado e inicializado, se procesan las peticiones recibidas y por ultimo se destruye. El Servlet es cargado una sola vez y esta residente en memoria mientras se procesan las peticiones recibidas y se generan las respuestas a los usuarios.

Cada vez que un cliente solicita al servidor Web una página JSP, este pasa la petición al motor de JSP el cual verifica si la página no se ha ejecutado antes ó fue modificada después de la última compilación, tras lo cual la compila, **convirtiéndola en Servlet**, la ejecuta y devuelve los resultados al cliente en formato HTML.

En la imagen siguiente se puede ver un esquema del proceso de conversión de una página JSP en un Servlet.



## 4 Contenido dinámico

Como se ha indicado anteriormente las páginas JSP generan páginas Web dinámicas combinando una parte estática (visto en la anterior: HTML, javascript, etc.) con una parte dinámica. Esta parte dinámica se puede generar mediante:

- a) Usar código Java directamente a través de implementar los métodos que necesite
- b) Usar código Java indirectamente a través de invocar métodos de clases
- c) Hacer uso de JavaBeans
- d) Desarrollar y usar “custom tags“
- e) Usar custom tags de terceros o JSTL (JSP Standard Tag Library)
- f) Diseñar un framework MVC (Modelo, Vista, Controlador) propio
- g) Usar frameworks MVC de terceros (Struts, JSF, etc.)

En esta sesión veremos los apartados a, b y c. Los apartados f y g los abordaremos en la sexta sesión cuando tengamos conocimientos de las tecnologías involucradas en el patrón MVC. Los apartados d y e no los veremos en este curso por razones temporales. Respecto al apartado g cabe indicar que JSF nos proporciona una solución adecuada al diseño visual de nuestras aplicaciones. Ahora bien, no estudiamos JSF en este primer curso de J2EE porque es recomendable previamente comprender la base de la generación dinámica de páginas Web, ya que podríamos querer aplicar los conocimientos adquiridos a arquitecturas distintas de Java, tales como PHP, .NET, ASP, etc. En un futuro se ofrecerá un segundo curso de J2EE donde se abordará el uso de frameworks.

El código fuente de una página JSP incluye:

- Directivas  
Dan información global de la página, por ejemplo, importación de elementos, página que maneja los errores, cuando la página forma parte de una sesión, etc.
- Scripts de JSP  
Es código Java embebido en la página.
- Variables predefinidas  
Son objetos definidos de forma implícita en toda página JSP
- Acciones de JSP  
Permite controlar el comportamiento del motor de Servlets

Veamos cada una de estas partes y relacionémoslo con el uso de NetBeans

### **NOTA:**

Los comentarios en JSP se introducen a través de

```
<%-- comentario --%>
```

## 5 Directivas JSP

Una **directiva** JSP afecta a la estructura general de la clase Servlet que se genera automáticamente a partir de la JSP. Normalmente tienen la siguiente forma:

```
<%@ directive attribute="value" %>
```

Sin embargo, también podemos combinar múltiples selecciones de atributos para una sola directiva, de esta forma:

```
<%@ directive attribute1="value1"
      attribute2="value2"
      ...
      attributeN="valueN" %>
```

Hay dos tipos principales de directivas: **page**, que nos permite hacer cosas como importar clases, personalizar la clase de la que extiende el Servlet generado, etc. e **include**, que nos permite insertar un fichero dentro de la clase Servlet en el momento que el fichero JSP es traducido a un Servlet.

### 5.1 La directiva page

La directiva page nos permite definir uno o más de los siguientes atributos (veremos únicamente aquellos que nos interesan en el curso):

- import="**package.class**"

Esto nos permite especificar los paquetes que deberían ser importados. Por ejemplo:

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

El atributo import es el único que puede aparecer múltiples veces.

- session="true|false".

Un valor de false indica que no se usarán sesiones, y los intentos de acceder a la variable session resultarán en errores en el momento en que la página JSP sea traducida a un Servlet.

- extends="**package.class**"

Esto indica la clase de la que extenderá el Servlet que se va a generar. Debemos usarla con extrema precaución, ya que el servidor J2EE podría no aceptarlo correctamente.

- info="**message**"

Define una descripción que puede usarse para ser recuperado mediante el método getServletInfo.

- errorPage="**url**"

Especifica una página JSP que se debería procesar si se lanzará cualquier excepción que no fuera capturada en la página actual.

- isErrorPage="true|false"

Indica si la página actual actúa o no como página de error de otra página JSP. El valor por defecto es false.

## 5.2 La directiva include JSP

Esta directiva nos permite incluir ficheros en el momento en que la página JSP es traducida a un Servlet. La directiva tiene el siguiente aspecto:

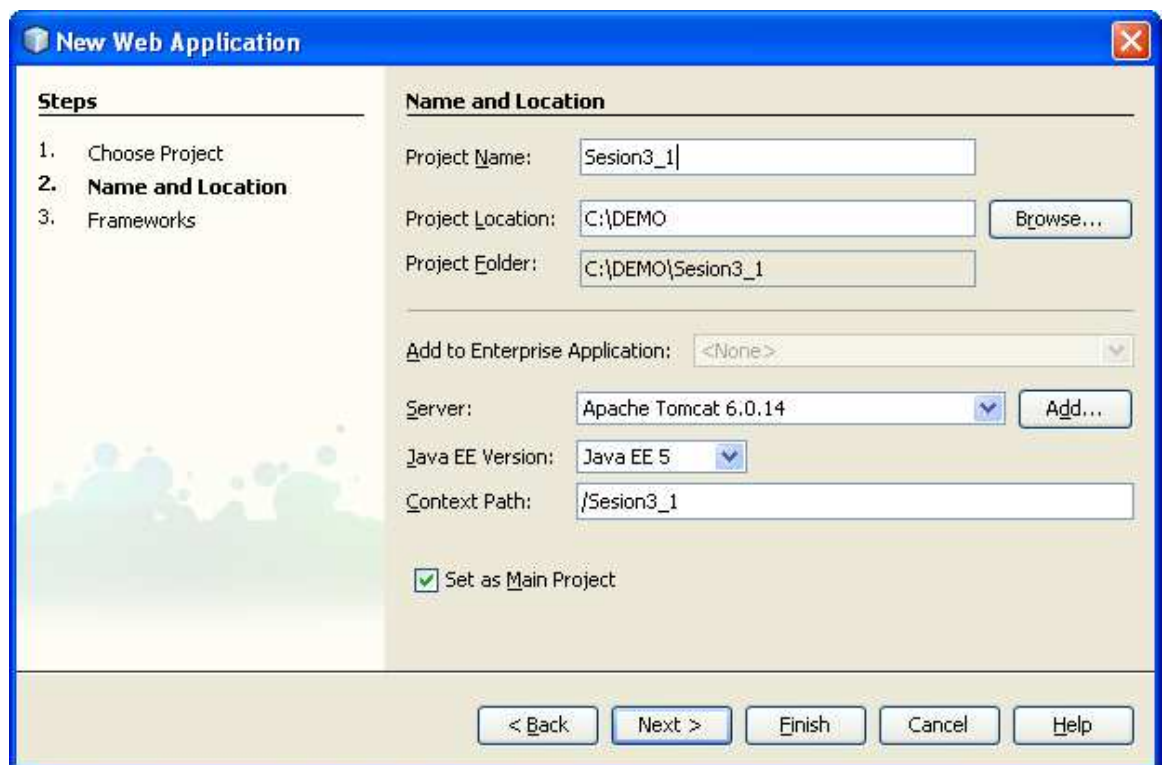
```
<%@ include file="url relativa" %>
```

La URL especificada se interpreta como relativa a la página JSP a la que se refiere. Los contenidos del fichero incluido son analizados como texto normal JSP, y así se pueden incluir HTML estático, elementos de script JSP, directivas y acciones.

Hagamos notar que la directiva `include` inserta los ficheros en el momento en que la página es traducida, si la página incluida cambia, tendríamos que re-traducir todas las páginas JSP que la refieren. Esto repercutiría en la eficiencia. En otras palabras, el uso de esta directiva es adecuada siempre el contenido del fichero incluido no cambie frecuentemente. En este otro caso sería adecuado usar la acción **jsp:include** como veremos más adelante.

## 5.3 Ejemplo desde NetBeans

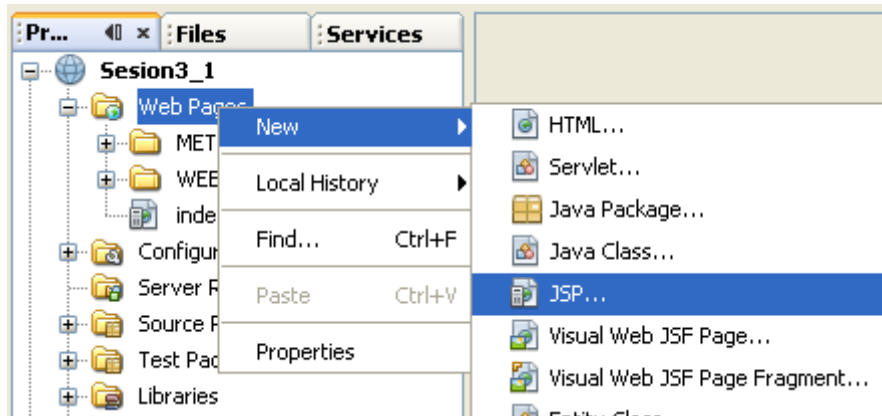
De igual modo que hicimos en la anterior sesión creemos un nuevo proyecto Web llamado Sesion3\_1 desde NetBeans.



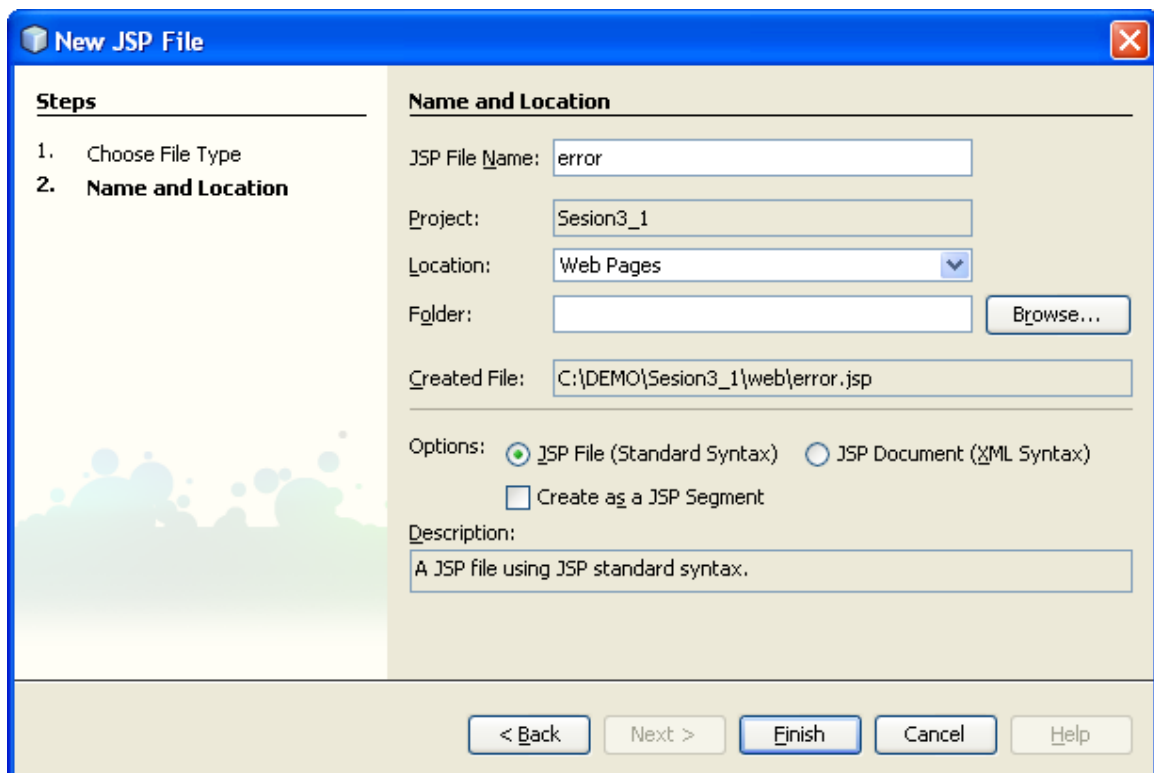


Para este primer ejemplo crearemos una página que produzca un error y otra que muestre el error producido. Para ello añadiremos dos páginas JSP, una llamada error.jsp y la otra pruebaerror.jsp.

Por ejemplo, para crear error.jsp indicaremos a NetBeans crear una nueva página JSP



y le indicaremos el nombre



Haremos lo mismo para pruebaerror.jsp

La ayudas de NetBeans en cuanto a páginas JSP se reducen a:

- Uso de JavaBeans (lo veremos más adelante)
- NetBeans te ofrece las diferentes opciones posibles con solo pulsar espacio en el elemento JSP correspondiente. Por ejemplo para la directiva **page** nos ofrece:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html:
    contentType
    <errorPage
    extends
    import
    <info
    <isELIgnored
    isErrorPage
    <isThreadSafe
</html>
language
```

Pasemos a introducir el código de **error.jsp**. En la cabecera de la página pondremos la directiva siguiente:

```
<%@page isErrorPage="true" %>
```

En el cuerpo introduciremos el mensaje que queremos que el usuario vea cuando se produzca un error:


```
<body>
Sentimos comunicarle que se ha producido un error al
intentar cargar la página, contacte con su administrador
para más información.
</body>
```

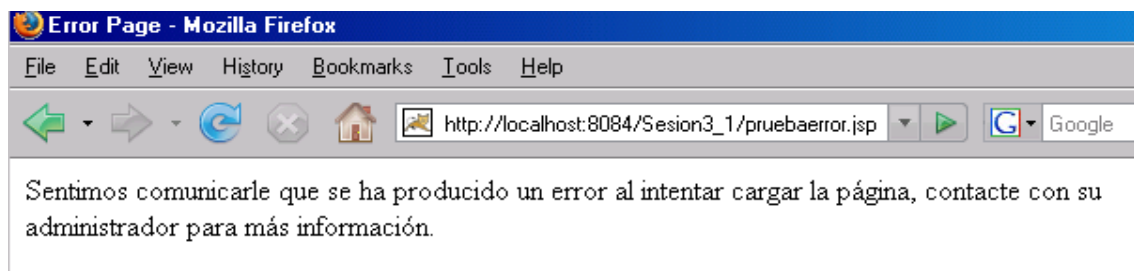
Pasemos ahora a introducir el código en **pruebaerror.jsp**. En la cabecera de la página pondremos la directiva siguiente:

```
<%@page errorPage="error.jsp"%>
```

En el cuerpo introduciremos un script JSP(los veremos a continuación) que produzca una excepción, en este caso la división por cero.

```
<body>
<%= 0/0 %>
</body>
```

Una vez creadas las dos páginas JSP pulsaremos el botón  para desplegar la aplicación. Veremos que se lanza el servidor Tomcat y que se abre el navegador Web. En este pondremos la dirección [http://localhost:8084/Sesion3\\_1/pruebaerror.jsp](http://localhost:8084/Sesion3_1/pruebaerror.jsp) y comprobaremos que la pantalla es automáticamente redirigida a error.jsp:



## 6 Scripts JSP

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

1. Expresiones de la forma `<%= expresión %>` que son evaluadas e insertadas en la salida.
2. Scriptlets de la forma `<% código %>` que se insertan dentro del método `Service` del Servlet, que es el método que se ejecuta una vez traducida la JSP a Servlet.
3. Declaraciones de la forma `<%! código %>` que se insertan en el cuerpo de la clase del Servlet, fuera de cualquier método existente.

### 6.1 Expresiones JSP

Una expresión JSP se usa para insertar valores Java directamente en la salida, es decir mostrarlos directamente al usuario. Tiene la siguiente forma:

```
<%= expresión Java %>
```

Esta expresión Java es evaluada, convertida a un string, e insertada en la página. Esta evaluación se realiza durante la ejecución (cuando se solicita la página) y así tiene total acceso a la información sobre la solicitud. Por ejemplo, esto muestra la fecha y hora en que se solicitó la página:

```
Fecha actual: <%= new java.util.GregorianCalendar().getTime() %>
```

### 6.2 Scriptlets JSP

Los scriptlets JSP nos permiten insertar código arbitrario dentro del método `service` del Servlet que será construido al generar la página. Tienen la siguiente forma:

```
<% Código Java %>
```

Hay que resaltar que el código dentro de un scriptlet se insertará exactamente como está escrito dentro del método `service` del Servlet, y cualquier HTML estático anterior o posterior al scriptlet se convierte en sentencias `println`. Esto significa que los scriptlets no necesitan completar las sentencias Java, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets. Por ejemplo, el siguiente fragmento JSP, contiene una mezcla de HTML y scriptlets:

```
<% if (Math.random() < 0.5) { %>
Ten un <B>bonito</B> día!
<% } else { %>
Ten un <B>buen</B> día!
<% } %>
```

que se convertirá en algo como esto dentro del método `service`:

```
if (Math.random() < 0.5) {
    out.println("Ten un <B>bonito</B> día!");
} else {
    out.println("Ten un <B>buen</B> día!");
}
```

### 6.3 Declaraciones JSP

Una declaración JSP nos permite definir métodos o campos que serán insertados dentro del cuerpo principal de la clase Servlet (fuera del método service que procesa la petición). Tienen la siguiente forma:

```
<%! código java%>
```

Como las declaraciones no generan ninguna salida, normalmente se usan en conjunción con lo anterior. Por ejemplo, aquí tenemos un fragmento de JSP que imprime el número de veces que se ha solicitado la página actual desde que el servidor se arrancó (o la clase del Servlet se modificó o se recargó):

```
<%! private int accessCount = 0; %> <!--define un campo -->
Accesos a la página:
<%= ++accessCount %>
```



Como se ve en el ejemplo esa variable pasa a ser del Servlet al que es traducido la página, dado que en el resto de llamadas se llama a esa instancia creada en memoria no se perderá el valor de la variable entre llamada y llamada

### 6.4 Ejemplo desde NetBeans

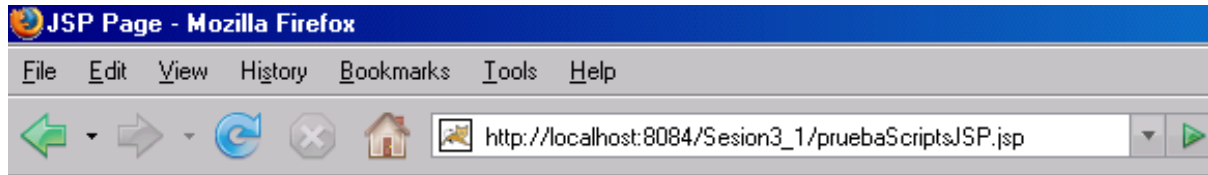
Añadiremos una nueva página JSP llamada pruebaScriptsJSP, al proyecto Sesión3\_1 creado anteriormente, donde insertaremos el siguiente código.

Observad el uso que se hace de la directiva **page** (nos permite usar las clases de java.util en nuestra página JSP) y de Expresiones, Scriptlets y Declaraciones JSP.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="java.util.*"%>
<!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=UTF-8">
<title>JSP Page</title>
</head>
<body>
Contenido dinámico mediante Scripts JSP:
<ul>
<li><b>Expresiones JSP</b><br>
Fecha actual: <%= new Date() %> </li>
<li><b>Scriptlet JSP</b><br>
<% Calendar cal = Calendar.getInstance();
out.println("La fecha actual es: " +
cal.get(Calendar.DATE)+" /"+
(cal.get(Calendar.MONTH)+1) + " / "+ cal.get(Calendar.YEAR));
%></li>
<li><b>Declaración (más expresión).</b><br>
<%! private int accessCount = 0; %>
Accesos a la página:
<%= ++accessCount %></li>
</ul>
</body>
</html>
```

Recordemos que NetBeans permite añadir y realizar modificaciones en páginas JSP y HTML de una aplicación J2EE una vez ha sido desplegada haciendo únicamente el guardar los cambios en NetBeans  y actualizar en el navegador Web .

El resultado del código anterior es:



Contenido dinámico mediante Scripts JSP:

- ♦ **Expresiones JSP**  
Fecha actual: Sat Sep 01 12:59:20 CEST 2007
- ♦ **Scriptlet JSP**  
La fecha actual es: 1 / 9 / 2007
- ♦ **Declaración (más expresión).**  
Accesos a la página: 3

## 7 Variables Predefinidas

Para simplificar el código en expresiones y scriptlets JSP, tenemos ocho variables definidas automáticamente, también llamadas *objetos implícitos*. Simplifican el código dado que también se pueden acceder a estos objetos haciendo una serie de casting, llamando a métodos concretos, etc. Las variables disponibles son: `request`, `response`, `out`, `session`, `application`, `config`, `pageContext`, y `page`.

### 7.1 Tipos de variables JSP predefinidas

#### 7.1.1 request

Este es el `HttpServletRequest` asociado con la petición, y fundamentalmente nos permite mirar los parámetros de la petición (mediante `getParameter`), además de otra serie de métodos.

#### 7.1.2 response

Este es el `HttpServletResponse` asociado con la respuesta al cliente.

#### 7.1.3 out

Este es el `PrintWriter` usado para enviar la salida al cliente. Observemos que `out` se usa casi exclusivamente en scriptlets ya que las expresiones JSP insertan valores directamente en la salida.

#### 7.1.4 session

Este es el objeto `HttpSession` asociado con la petición. Los ámbitos de los objetos los estudiaremos más detenidamente en el siguiente tema. Si usáramos el atributo `session` de la directiva `page` para desactivar las sesiones, los intentos de referenciar la variable `session` causarán un error en el momento de traducir la página JSP a un Servlet.

#### 7.1.5 application

Este es el `ServletContext` obtenido mediante `getServletConfig().getContext()`. Este objeto es común para toda la aplicación Web y, entre otras cosas, nos permite almacenar información que será accesible desde todas las páginas de la aplicación Web, independientemente de `session`.

#### 7.1.6 config

Este es el objeto `ServletConfig` asociado al Servlet al que se traduce la página. Permite acceder a parámetros de inicialización del Servlet y a su contexto.

#### 7.1.7 pageContext

Es un objeto de la clase `PageContext`. Entre otras cosas, nos permite almacenar información en diferentes ámbitos (`Page`, `Request`, `Session`, `Application`) como veremos en el próximo tema.

### 7.1.8 page

Esto es sólo un sinónimo de `this`.

## 7.2 Ejemplo desde NetBeans

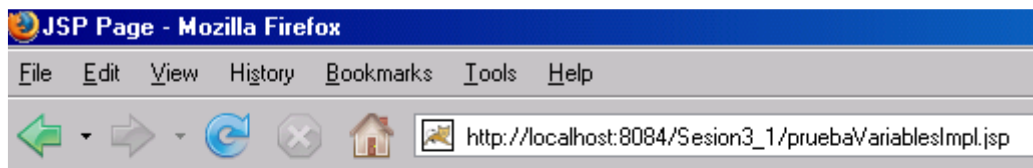
Añadiremos una nueva página JSP llamada **pruebaVariablesImpl.jsp**, al proyecto `Sesion3_1` creado anteriormente, donde insertaremos el siguiente código, el cual nos permite obtener información del objeto asociado a la petición del cliente Web al servidor JSP:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="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=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    La IP de tu ordenador es: <%= request.getRemoteHost () %>
    <BR>
    El nombre del servidor es:<%= request.getServerName () %>
    <BR>
    El puerto del servidor: <%=request.getServerPort () %>
    <BR>
    La IP del servidor es:<%=request.getRemoteAddr () %>
    <BR>
    El protocolo que estas usando es: <%=request.getProtocol () %>
  </body>
</html>
```

El resultado del código anterior es:



La IP de tu ordenador es: 127.0.0.1  
 El nombre del servidor es:localhost  
 El puerto del servidor: 8084  
 La IP del servidor es:127.0.0.1  
 El protocolo que estas usando es: HTTP/1.1

## 8 Acciones JSP

Las **acciones** JSP acciones tienen la forma `<jsp:accion [parámetros]/>` y permiten controlar el comportamiento del motor de Servlets. Permiten insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, o generar HTML para el plug-in Java.

### 8.1 Tipos de acciones JSP

#### 8.1.1 jsp:include

Esta acción nos permite insertar ficheros en una página que está siendo generada. La sintaxis se parece a esto:

```
<jsp:include page="url_relativa" />
```

Al contrario que la directiva `include`, que inserta el fichero en el momento de la conversión de la página JSP a un Servlet, esta acción inserta el fichero en el momento en que la página es solicitada. Esto se paga en eficiencia, pero proporciona una significativa flexibilidad, dado que modificar las páginas incluidas no afectará al rendimiento que siempre será constante.

#### 8.1.2 jsp:forward

Esta acción nos permite reenviar la petición a otra página. Tiene un sólo atributo, `page`, que debería consistir en una URL relativa. Este podría ser un valor estático, o podría ser calculado en el momento de la petición, como en estos dos ejemplos:

```
<jsp:forward page="/utils/pagina2.jsp" />
<jsp:forward page="<%= someJavaExpression %>" />
```

#### 8.1.3 jsp:plugin

Esta acción nos permite insertar un un applet usando el Plug-in Java (se traducirá a un elemento `OBJECT` o `EMBED` dependiendo del navegador).

#### NOTA:

Antes de ver el uso de un JavaBean desde JSP, aclaremos que un JavaBean es un modelo de componente creado por Sun Microsystems para la construcción de aplicaciones en Java. Según su especificación se definen como "componentes de software **reutilizables** que se puedan manipular visualmente en una herramienta de construcción".

En resumen un JavaBean no es más que una clase que:

- \* implementa la interfaz serializable
- \* tiene un constructor sin argumentos
- \* sus propiedades son accesibles mediante métodos `get` y `set`
- \* contiene determinados métodos de manejo de eventos.

Nosotros haremos uso de **JavaBeans “simplificados”** (aunque los llamemos JavaBeans), esto es, cumplirán con los apartados b) y c).



### 8.1.4 jsp:useBean

Esta acción nos permite cargar y utilizar un JavaBean en la página JSP. La sintaxis más simple para especificar que se debería usar un JavaBean es:

```
<jsp:useBean id="name" class="package.class"
scope=="page|request|session|application" type="..." />
```

Esto significa "instanciar un objeto de la clase especificada por **class** si no existe uno con el mismo nombre y ámbito, y únelo a una variable con el nombre especificado por **id**".

Los atributos **id** y **class** son obligatorios. Los atributos **scope** y **type** son opcionales.

Mediante especificar el atributo **scope** se puede hacer que ese JavaBean se asocie con más de una sola página. Esto lo veremos en la siguiente sesión.

Mediante especificar el atributo **type** se especifica el tipo de la variable a la que se referirá el objeto. Este debe corresponder con el nombre de la clase o una superclase o un interface que implemente la clase. Es decir permite realizar un casting del objeto.

Ahora, una vez que tenemos un JavaBean, podemos:

- modificar sus propiedades mediante `jsp:setProperty`, o usando un scriptlet y llamando a un método explícitamente sobre el objeto con el nombre de la variable especificada anteriormente mediante el atributo **id**.
- Del mismo modo para leer sus propiedades usaremos la acción `jsp:getProperty`, o usaremos un scriptlet que llame al método **getXXX** correspondiente

### 8.1.5 jsp:setProperty

Usamos `jsp:setProperty` para establecer valores de propiedades de los beans que se han referenciado anteriormente. Lo usaremos de esta forma:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="someProperty"
value="someValue" />
```

Esto significa "usa la instancia referenciada como **myName** y establece su propiedad llamada **someProperty** con el valor de **someValue**".

Un caso especial es la orden

```
<jsp:setProperty name="myName" property="*" />
```

la cual significa que todos los parámetros del formulario cuyos nombres correspondan con nombres de propiedades del Bean serán almacenados.

### 8.1.6 jsp:getProperty

Este elemento recupera el valor de una propiedad del bean, lo convierte a un String, e inserta el valor en la salida de la JSP. Se usa del siguiente modo:

```
<jsp:useBean id="myName" ... />
```

```
...
<jsp:getProperty name=" myName " property="someProperty" />
```

Esto significa “usa la instancia referenciada como **myName**, recupera su propiedad llamada **someProperty** y muéstrala”.

## 8.2 Ejemplo desde NetBeans

A continuación veremos un ejemplo de uso de `jsp:forward`. En el siguiente apartado veremos ejemplo de cómo usar `jsp:useBean` en formularios.

En este ejemplo usaremos `jsp:forward` reenviando la respuesta hacia una página HTML u otra en función de un número aleatorio. Para esto crearemos una página JSP en el proyecto `Sesion3_1` llamada **pruebaAccion.jsp** con el siguiente código:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

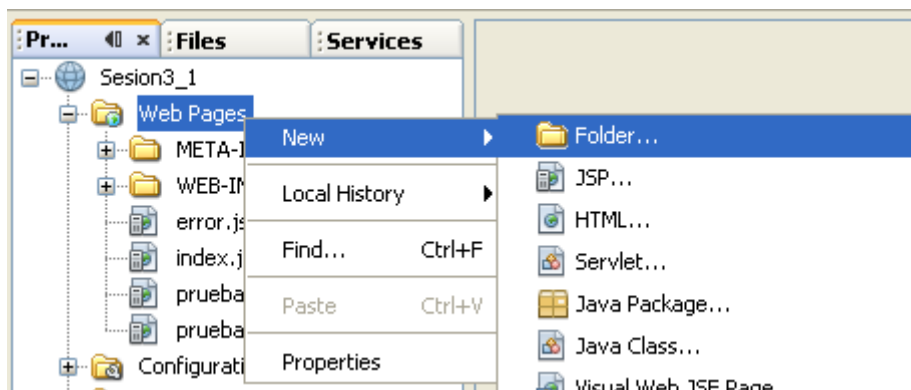
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>

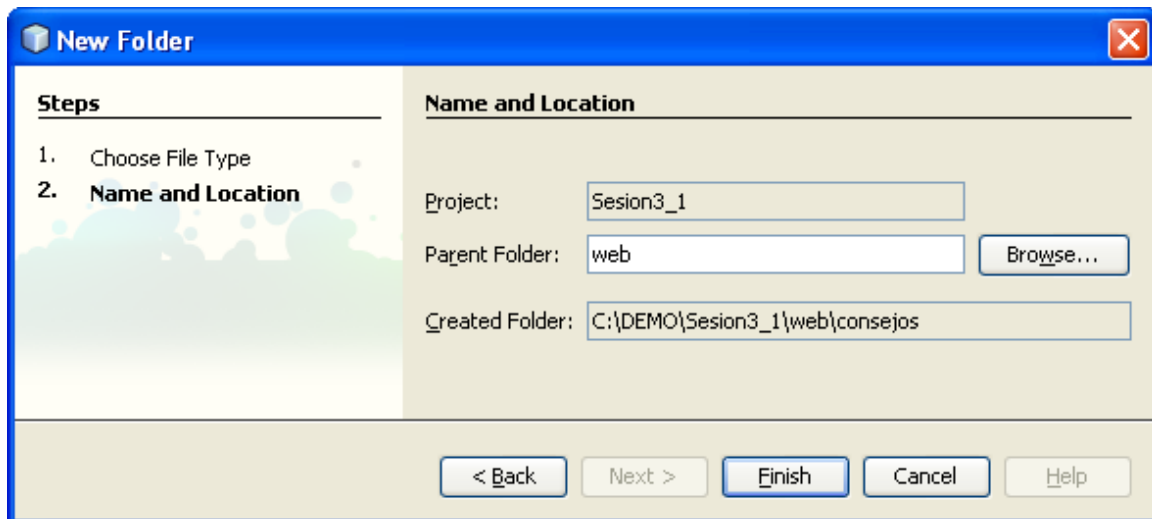
    <% if (Math.random() < 0.5) { %>
    <jsp:forward page="/consejos/consejo1.html" ></jsp:forward>
    <% } else { %>
    <jsp:forward page="/consejos/consejo2.html" ></jsp:forward>
    <% } %>

  </body>
</html>
```

Como vemos en el código se hace uso de `consejo1.html` y `consejo2.html` que se encuentran en la carpeta de `consejos`. Para esto hemos de crear una carpeta llamada `consejos` dentro de las carpetas `Páginas Web`. Esto es:



y rellenaremos el nombre de la carpeta



De este modo, tendremos una carpeta llamada consejos en la que añadiremos dos documentos HTML como los que se muestran

```

consejo1.html x
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">

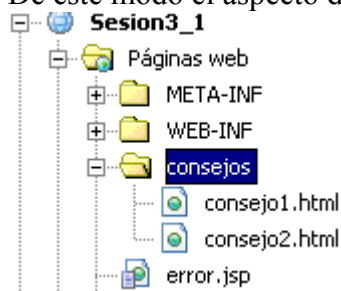
<html>
<head>
  <meta http-equiv="content-type"
  content="text/html; charset=UTF-8">
  <title></title>
</head>
<body>
  ¡Las gracias y el por favor no
  cuestan nada y tienen gran valor!
</body>
</html>
  
```

```

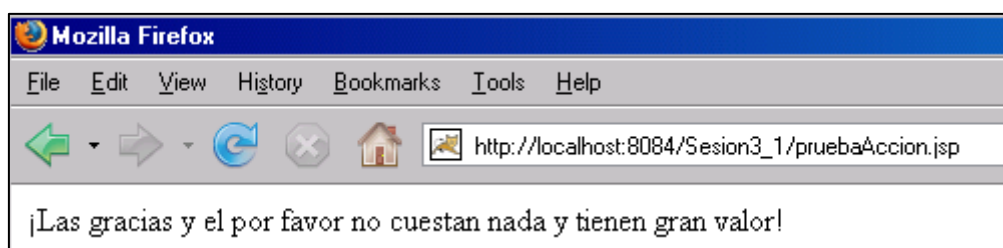
consejo2.html x
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">

<html>
<head>
  <meta http-equiv="content-type"
  content="text/html; charset=UTF-8">
  <title></title>
</head>
<body>
  ¡Es de bien nacido ser agradecido!
</body>
</html>
  
```

De este modo el aspecto de nuestro proyecto es



El resultado del código anterior es una página en la que cada vez que actualice muestra un consejo u otro:



## 9 JavaBeans y Formularios

### 9.1 Formularios con JSP

Usando JSP, los datos de un formulario (la información que el usuario introduce en él) se almacenan en un objeto **request** que es enviado desde el navegador Web del cliente hasta el contenedor Web.

La petición es procesada y el resultado se envía a través de un objeto **response** de vuelta al navegador.

Tanto request como response están disponibles implícitamente como variables predefinidas.

Mediante el método **request.getParameter** se recuperan los datos desde el formulario en variables creadas usando etiquetas de introducción de datos (vistos en la anterior sesión, eran input, textarea, select, radiobutton). En el caso de que se recuperen varios valores (checkbox) se usará el método **request.getParameterValues**.

### 9.2 Formularios con JSP y JavaBeans

Cuanto más código esté implicado en un formulario, más importante es no mezclar la lógica de negocio con la presentación final. De este modo:

- el código de la página JSP se debe limitar a la **presentación** final: formatea los datos de los JavaBeans para su visualización en el navegador
- Los JavaBeans contienen los datos que debe mostrar la página. Estos JavaBeans se han completado en la capa de **lógica de Negocio**.

En otras palabras, una página JSP utiliza un componente JavaBean fijando (**jsp:setProperty**) y obteniendo (**jsp:getProperty**) las propiedades que proporciona.

Entre los beneficios en la utilización de JavaBeans para mejorar las páginas JSP destacaremos:

- Componentes Reutilizables: diferentes páginas JSP pueden reutilizar los mismos JavaBeans.
- Separación de la lógica de negocio de la lógica de presentación: podemos modificar la forma de mostrar los datos sin que afecte a la lógica del negocio, y viceversa.

### 9.3 Ejemplo desde NetBeans

#### 9.3.1 Formularios sin JavaBeans

Para demostrar como manejar formularios HTML usando JSP sin JavaBeans, crearemos un formulario de ejemplo con dos campos: uno para el nombre y otro para el email. Este formulario HTML lo definiremos en una página JSP llamada process.jsp la cual dependiendo de si los valores del formulario son **null** (primera vez que se accede) imprime el formulario o muestra los valores que se rellenaron.

Crearemos una página llamada **process.jsp** en el proyecto Sesión3\_1 creado anteriormente y la rellenaremos con el siguiente código. Obsérvese que se usan elementos de ScriptJSP y básicamente lo que hacemos es en función de si se ha rellenado o no los campos del formulario se muestra:

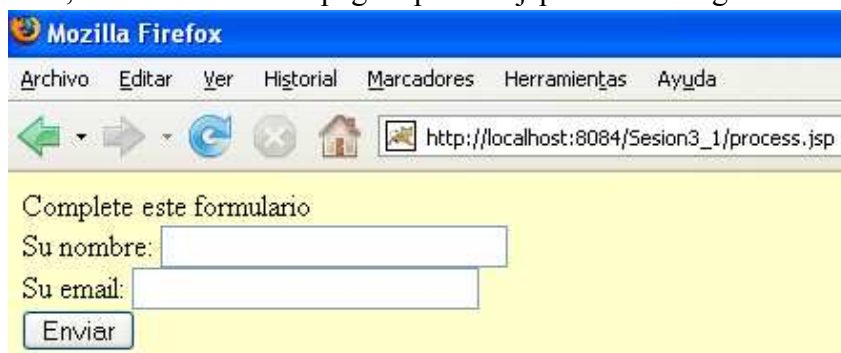
- o bien el formulario (con el action apuntando a la propia página)
- o bien los campos que se rellenaron

```
<%@page contentType="text/html"%>
<%@page pageEncoding="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=UTF-8">
  </head>
  <body bgcolor="#ffffcc">
    <% if ((request.getParameter("name")==null
      && request.getParameter("email")== null)
      ||
      (request.getParameter("name").equals("")
      && request.getParameter("email").equals(""))
      )
      { %>
        Complete este formulario
        <form method="POST" action="process.jsp">
          Su nombre: <input type="text" name="name" size=26><BR>
          Su email: <input type="text" name="email" size=26><BR>
          <input type="submit" value="Enviar">
        </form>
      <% } else { %> <!-- scriptlet JSP -->
        <% String nombre, mail; %>
        <%
          nombre = request.getParameter("name");
          mail = request.getParameter("email");
        %>
        <b>Usted indicó la siguiente información</b>:
        <BR><B>Nombre</B>: <%= nombre %><!-- expresion JSP -->
        <BR><B>Email</B>: <%= mail %>
      <% } %>
    </body>
  </html>
```

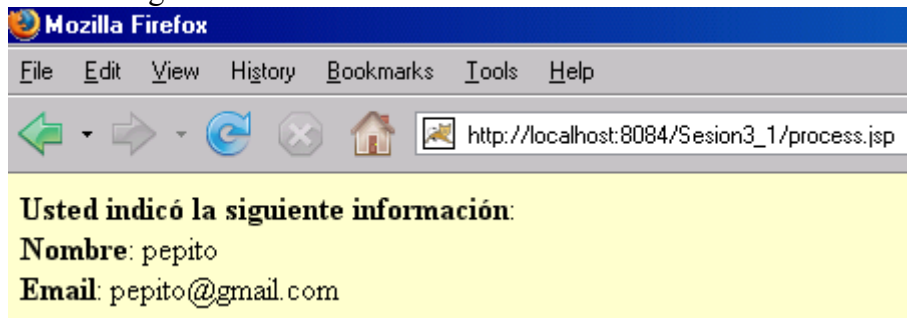
#### NOTA:

Obsérvese que en los formularios que utilizaremos **no aparece** `enctype="text/plain"` como en la sesión anterior. Este atributo se usa en el caso de que ACTION sea mailto. En el resto de casos no lo indicaremos.

De este modo, si solicitáramos la página process.jsp veríamos algo similar a:



Introducimos nuestro nombre y email y pulsamos Enviar para procesar el formulario, y veremos algo similar a:

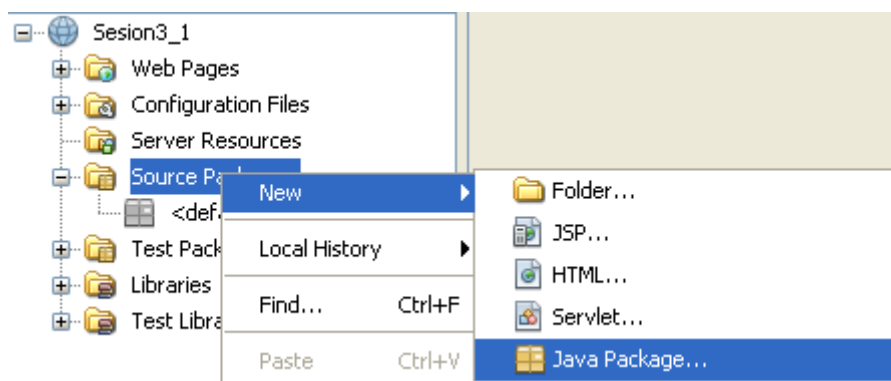


Veamos ahora como mejorar el código anterior usando JavaBeans.

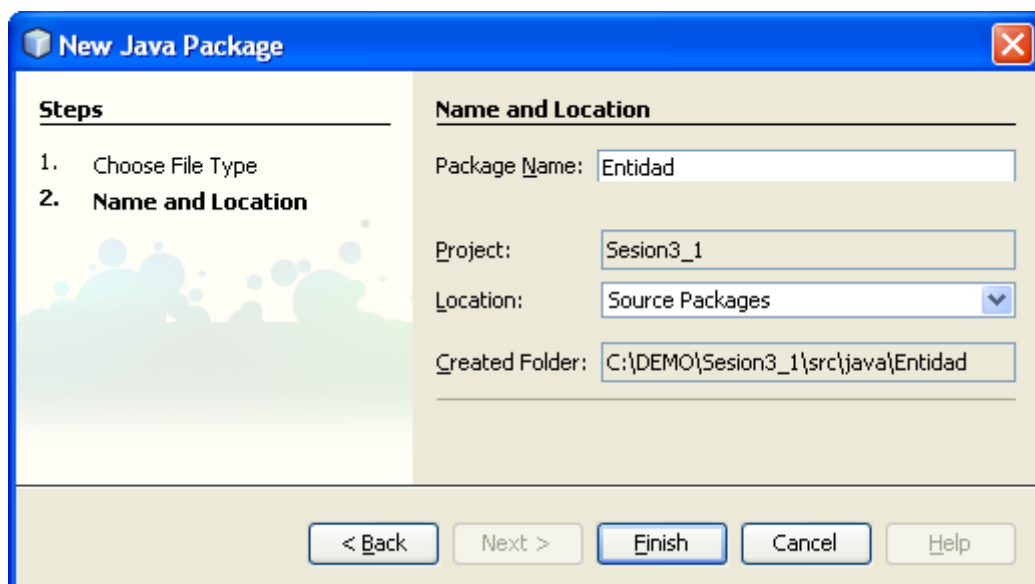
### 9.3.2 Formularios con JavaBeans

Ahora, veamos como modificar el ejemplo anterior process.jsp para usar JavaBeans. En el formulario anterior había dos campos: name y email. En nuestra clase JavaBean, definiremos propiedades con métodos setXXX y getXXX donde XXX es el nombre de la propiedad.

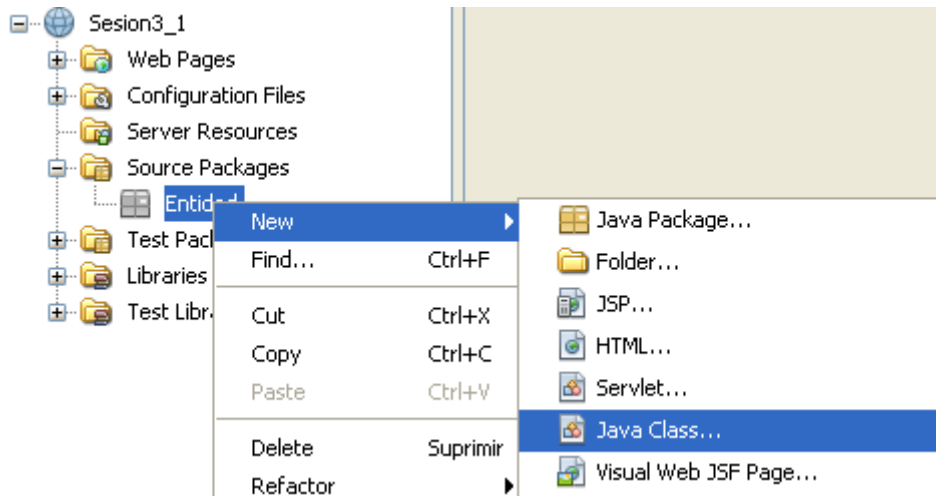
Crearemos primero el package donde irá la clase mediante ir a la carpeta de Paquetes de origen del proyecto Sesion3\_1 y añadir un nuevo Paquete Java (pulsamos botón derecho sobre Paquetes de origen):



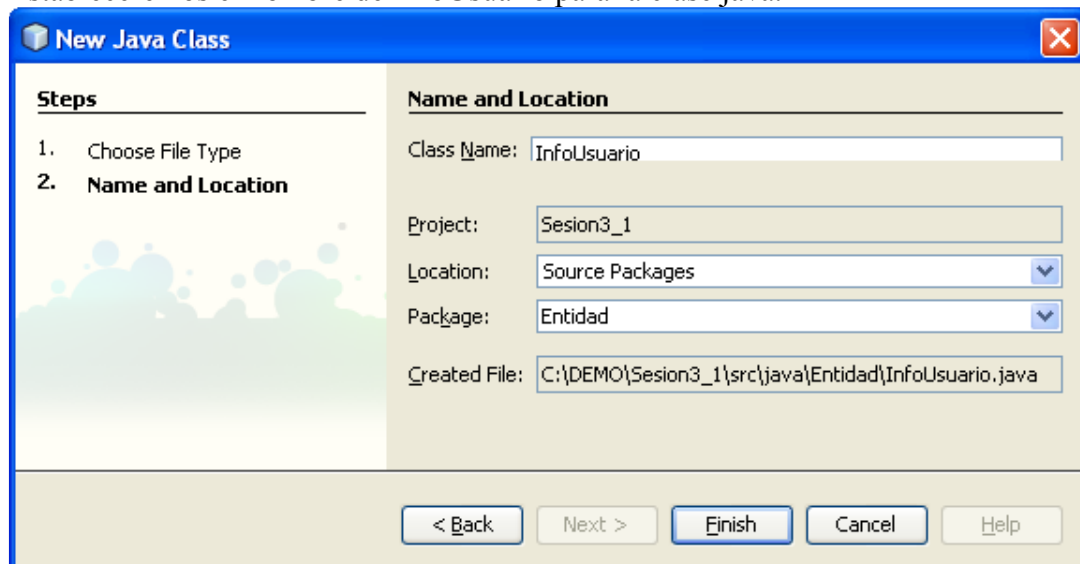
Llamaremos al paquete Entidad (de este modo lo vamos relacionando con las capas que vimos en la primera sesión)



A continuación crearemos una clase Java que llamaremos UsuarioBean. Para esto añadiremos un archivo al paquete entidad recién creado (pulsamos botón derecho sobre Entidad) de tipo Clase Java



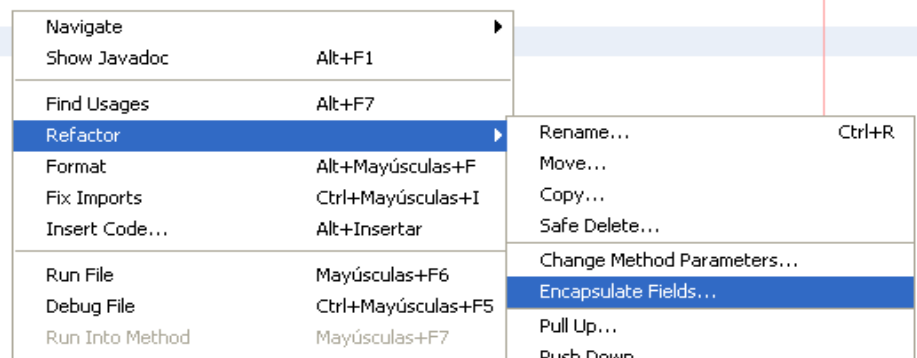
Estableceremos el nombre de InfoUsuario para la clase java.



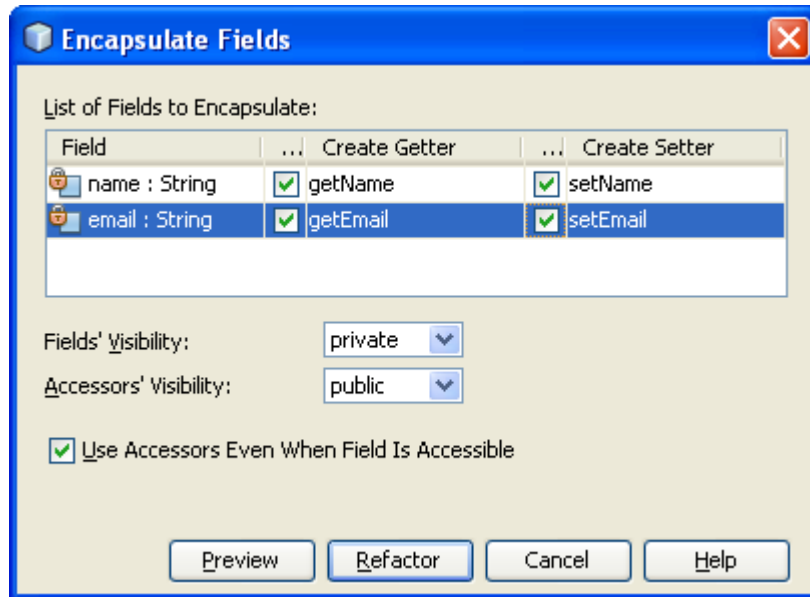
Añadiremos las propiedades name y email, para esto escribiremos los atributos privados en la clase InfoUsuario y usaremos la funcionalidad “Encapsulate Fields” de NetBeans (pulsando botón derecho sobre el código)

```
package Entidad;
```

```
public class InfoUsuario {
    private String name;
    private String email;
}
```



Aparecerá un diálogo que nos permite especificar los métodos accesorios para los atributos del JavaBean (que a partir de este momento se llamarán propiedades del JavaBean). Pulsaremos Refactor.



Si echamos un vistazo al código fuente veremos la siguiente clase, en la cual podemos ver sus dos propiedades, con sus campos y métodos accesorios get y set correspondientes:

```
package Entidad;
```

```
public class InfoUsuario {
    private String name;
    private String email;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

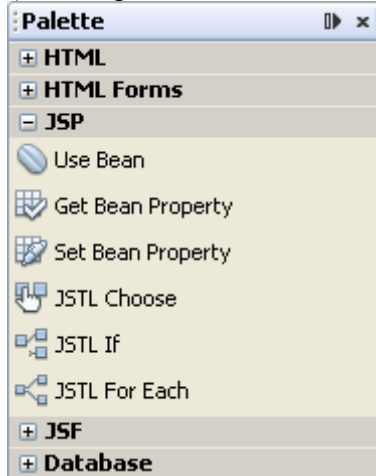
    public void setEmail(String email) {
        this.email = email;
    }
}
```


Para poder usar el anterior componente InfoUsuario en nuestra página JSP, necesitamos instanciar el componente a través de `<jsp:useBean>`. Una vez instanciado usaremos `<jsp:setProperty>` para inicializar sus propiedades. Este será process2\_1.jsp

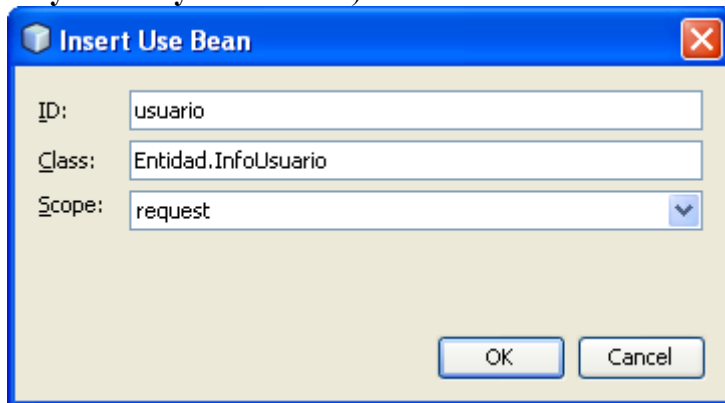


Mandaremos la respuesta a otro JSP, que presentará el contenido del JavaBean a través de `<jsp:getProperty>`. Este será process2\_2.jsp

Pasemos a crear process2\_1.jsp. Una vez creada la página usaremos la paleta de JSP (zona superior derecha de la pantalla de NetBeans) para añadir el JavaBean a la página.




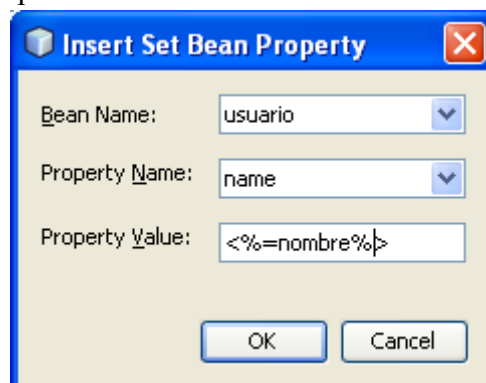
De este modo arrastraremos al principio de la página y después de las directivas el gráfico de usar Bean  que rellenaremos con los valores (prestad atención a las **mayúsculas y minúsculas**)



De este modo al principio de la página process2\_1.jsp veremos

```
<%page contentType="text/html"%>
<%page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<jsp:useBean id="usuario" scope="request" class="Entidad.InfoUsuario" />
```

A la hora de definir las propiedades del Bean arrastraremos de la paleta el gráfico de definir propiedad del Bean  el cual nos mostrará la siguiente pantalla que rellenaremos para establecer el nombre



y haremos lo mismo para establecer el email




De este modo process2\_1.jsp quedará como sigue:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<jsp:useBean id="usuario" scope="request" class="Entidad.InfoUsuario" />


<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <% if (request.getParameter("name")==null
    && request.getParameter("email")== null) { %>
      Complete este formulario
      <form method="POST" action="process2_1.jsp">
        Su nombre: <input type="text" name="name" size=26><BR>
        Su email: <input type="text" name="email" size=26><BR>
        <input type="submit" value="Enviar">
      </form>
    <% } else { %> <!-- scriptlet JSP --%>
      <% String nombre, mail; %>
      <%
        nombre = request.getParameter("name");
        mail = request.getParameter("email");
      %>
      <jsp:setProperty name="usuario" property="name" value="<%=nombre%>" />
      <jsp:setProperty name="usuario" property="email" value="<%=mail%>" />
      <jsp:forward page="/process2_2.jsp" ></jsp:forward>
      <% } %>
    </body>
  </html>
```

Como vemos en este caso cuando capturamos los valores del formulario los almacenamos en las propiedades del JavaBean y no presentamos nada, sino que dejamos que de eso se encargue process2\_2.jsp que tendrá el JavaBean relleno y solo se preocupará de realizar la presentación. Cuando en la siguiente sesión veamos Servlets apreciaremos que la parte de **else** del código de process2\_1.jsp se hace normalmente desde un Servlet, con lo cual

quedará totalmente separada la presentación de los datos de la lógica de negocio que prepara esos datos.

Pasemos a definir process2\_2.jsp. En este caso volveremos a usar  Use Bean en la cabecera y volveremos a rellenarlo con los mismos datos. Esto es



En process2\_2.jsp queremos mostrar las propiedades del JavaBean, para esto usaremos el gráfico de Obtener propiedad de Bean  Get Bean Property de la paleta de JSP que arrastraremos hasta la página. Aparecer la siguiente pantalla que rellenaremos para obtener la propiedad name:



Y también para la propiedad email:

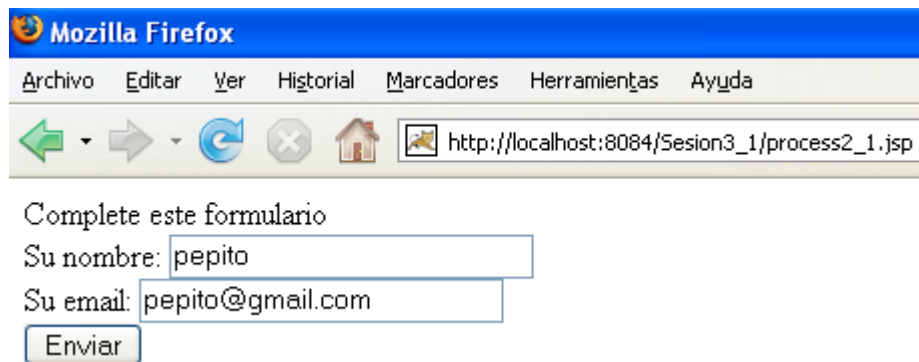


De este modo process2\_2.jsp quedará del siguiente modo:

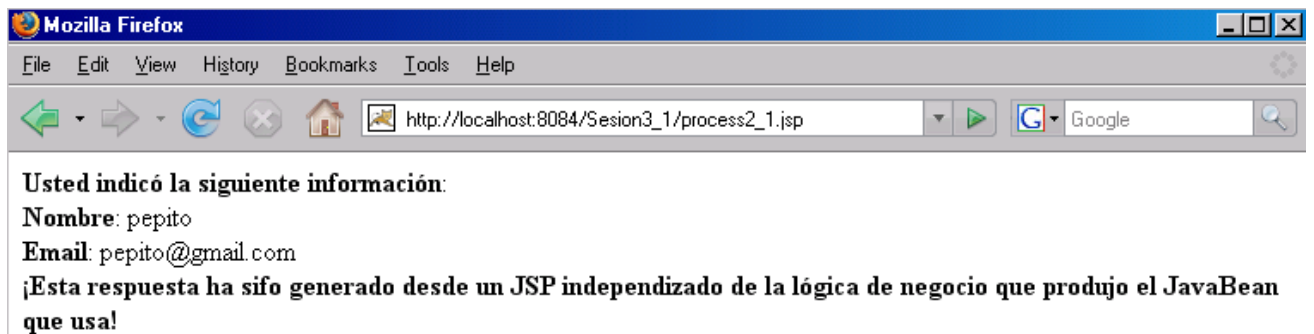
```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<jsp:useBean id="usuario" scope="request" class="Entidad.InfoUsuario" />

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <b>Usted indicó la siguiente información</b>:
    <BR><B>Nombre</B>: <jsp:getProperty name="usuario" property="name" />
    <BR><B>Email</B>: <jsp:getProperty name="usuario" property="email" />
    <BR><b>¡Esta respuesta ha sido generada desde un JSP independizado
    de la lógica de negocio que produjo el JavaBean que usa!</b>
  </body>
</html>
```

Si pasamos a observar los resultados en el navegador indicaremos la siguiente dirección URL [http://localhost:8084/Sesion3\\_1/process2\\_1.jsp](http://localhost:8084/Sesion3_1/process2_1.jsp)



Una vez pulsemos Enviar se mostrará el siguiente resultado

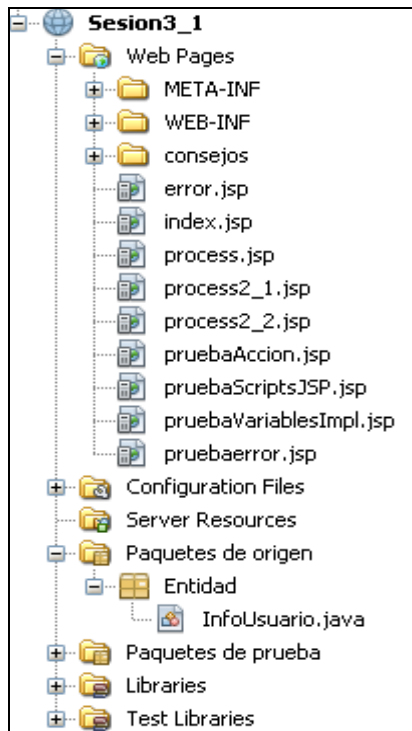


## 10 Ejercicios

Realiza los siguientes ejercicios y envíalos en un comprimido con tu nombre y apellidos al profesor. (Ej.: vperezcabello\_sesion3.zip).

### 10.1 Ejercicio 1 (Entregado por el profesor)

Realiza la aplicación ejemplo Sesión3\_1 explicada a lo largo de la sesión. Para que te sirva de guía estos son los elementos que debe tener este proyecto:



**NOTA:** Este ejercicio es entregado por el profesor, de modo que el alumno tenga disponible el código indicado en la sesión.

Realiza los siguientes ejercicios creando una nueva aplicación Web **Sesión3\_2** desde NetBeans

### 10.2 Ejercicio 2 (2 puntos)

Usando la clase `java.util.Calendar`, haz un JSP llamado `MostrarHora.jsp` que imprima la hora y los minutos

### 10.3 Ejercicio 3 (2 puntos)

Usando `Scripts JSP` crea una página llamada `TablaAuto.jsp` que cree mediante un bucle una tabla como la que se muestra a continuación

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

### 10.4 Ejercicio 4 (3 puntos)

Crea una página JSP llamada `formulario1.jsp`, otra llamada `formulario2.jsp` y un JavaBean adecuado llamado `AlquilerBean`. De modo que `formulario1.jsp` muestre el siguiente formulario (es el mismo que el del ejercicio 8.2 de la sesión anterior) y procese sus datos introduciéndolos en `AlquilerBean` (en package `Entidades`) y `formulario2.jsp` muestre en pantalla las opciones elegidas por el usuario. Esto es:

#### Formulario1.jsp

**VIDEOCLUB ON-LINE**

Nombre Pelicula

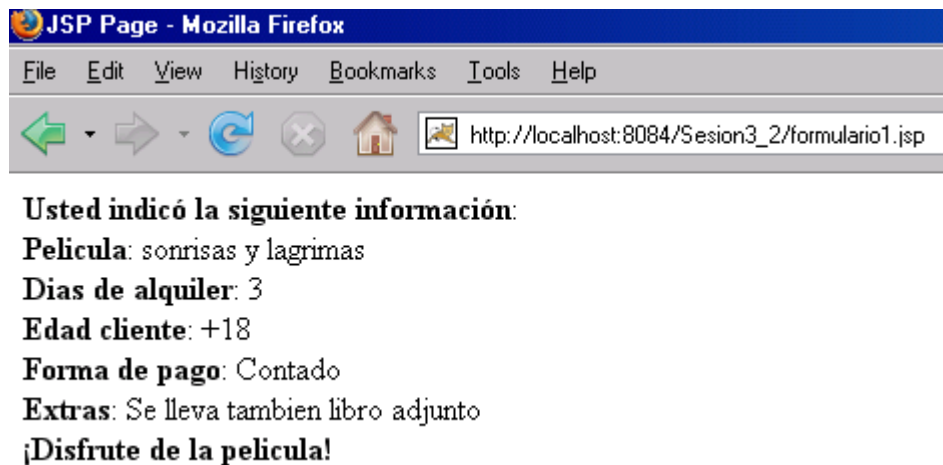
Número Días Alquiler

Edad Cliente ☐ Menor de 7 años  
☐ Menor de 14 años  
☐ Menor de 18 años  
☐ Mayor de 18 años

Forma de Pago

Especificaciones extras

## Formulario2.jsp



### 10.5 Ejercicio 5 (3 puntos)

Muchas Web incluyen una pequeña barra de navegación en cada página. Debido a los problemas con los marcos HTML, esto normalmente se implementa mediante una pequeña tabla que cruza la parte superior de la página o el lado izquierdo, con el HTML repetido para cada página de la Web.

Crea un documento HTML llamado navbar.html con enlaces a las páginas JSP de los ejercicios anteriores, esto es: MostrarHora.jsp, TablaAuto.jsp y formulario1.jsp

Usando la directiva include haz que se muestre esa cabecera en todas las páginas JSP que has realizado en la aplicación Sesion3\_2