



# Desarrollo de Aplicaciones Informáticas

CICLO FORMATIVO DE GRADO SUPERIOR

FORMACIÓN PROFESIONAL A DISTANCIA

Unidad 9

**Programación WEB - JSP Aplicaciones**

MÓDULO

**Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE**



FORMACIÓN PROFESIONAL

Principado de Asturias

**Título del Ciclo:** DESARROLLO DE APLICACIONES INFORMATICAS

**Título del Módulo:** DESARROLLO DE APLICACIONES EN ENTORNOS DE CUARTA GENERACIÓN Y CON HERRAMIENTAS CASE

**Unidad 9:** **Programación WEB - JSP Aplicaciones**

**Dirección:** Dirección General de Políticas Educativas, Ordenación Académica y Formación Profesional  
Servicio de Formación Profesional Inicial y Aprendizaje Permanente

**Dirección de la obra:**

Alfonso Careaga Herrera  
Antonio Reguera García  
Arturo García Fernández  
Ascensión Solís Fernández  
Juan Carlos Quirós Quirós  
Luís M<sup>a</sup> Palacio Junquera  
Yolanda Álvarez Granda

**Coordinador de los contenidos:**

Juan Manuel Fernández Gutiérrez

**Autores:**

Juan Manuel Fernández Gutiérrez

**Colección:**

Materiales didácticos de aula

**Serie:**

Formación profesional específica

**Edita:**

Consejería de Educación y Ciencia

**ISBN:**

978-84-692-3443-2

**Deposito Legal:**

AS-3564-2009

**Copyright:**

**2009.** Consejería de Educación y Ciencia

Esta publicación tiene fines exclusivamente educativos. Queda prohibida la venta, así como la reproducción total o parcial de sus contenidos sin autorización expresa de los autores y del Copyright

---

### ***Introducción.***

En esta unidad desarrollaremos aplicaciones WEB sin acceso a datos. Manejaremos la gestión de errores, los JavaBeans, las sesiones y comentaremos la utilidad de las Cookies.

Al final de la misma se pretende que se pueda presentar una página web con una funcionalidad básica.

### ***Objetivos***

- Conocer los diferentes modelos de gestión de errores
- Utilizar JavaBeans
- Entender la importancia de las sesiones
- Conocer la función de las cookies

## Contenidos Generales

1. MANEJAR FORMULARIOS .....	3
1.1. <i>request.getParameterNames()</i> y <i>request.getParameterValues()</i> .....	6
1.2. <i>Acción estándar jsp:forward</i> .....	10
2. JAVABEANS .....	12
2.1. <i>Crear un JavaBean</i> .....	13
2.2. <i>Utilizando el JavaBean</i> .....	13
2.3. <i>Utilizando el setProperty y getProperty</i> .....	14
3. APLICACIÓN FORMULARIO. ....	18
4. GESTIÓN DE ERRORES .....	21
5. SESIONES .....	23
5.1. <i>Trabajando con sesiones</i> .....	24
5.2. <i>Guardar y recuperar objetos en las sesiones</i> .....	26
5.3. <i>Caso práctico.</i> .....	27
6. COOKIES .....	31
6.1. <i>Crear cookies.</i> .....	32
6.2. <i>Recuperar cookies.</i> .....	35

## 1. Manejar formularios

Una de las partes más comunes de una aplicación es el formulario HTML en el que los usuarios introducen información. Recordemos que un formulario comienza con la etiqueta de apertura `<form>` y termina con la de cierre `</form>`. Se utilizan tres parámetros para definir su funcionamiento:

- *name*: da nombre al formulario.
- *method*: especifica el método para enviar los datos al servidor, si el método es GET los envía añadiéndolos a la URL, de esta forma son visibles al usuario, si es POST la información se añade a la cabecera del protocolo http, en este caso no son visibles al usuario.
- *action*: es el destino del formulario cuando se pulsa el botón enviar. En nuestro caso será una página JSP la que recoja los datos.

La información que llega a la página JSP se recoge a través de alguno de los métodos del objeto implícito **request**.

Lo normal es recoger la información de un parámetro mediante una variable que contiene el valor, en este caso se utilizará el método *getParameter()* especificando el nombre de la variable.

Es importante recordar que todos los datos independientemente del tipo que sean, son recogidos y evaluados como *String*, luego aquellos que siendo numéricos los vamos a utilizar para operar es necesario convertirlos.

La siguiente línea recoge el valor del campo *importe* y lo visualiza

```
<%  
String importe = request.getParameter("importe");  
out.println("Importe: " + importe);  
%>
```

Si realizamos la siguiente operación

```
Out.println("Nuevo Importe: " + (importe+20));
```

No sumaría, sino que concatenaría el importe con el literal 20

Para que sume hay que convertirlo a entero

```
<%
int importeNumerico = Integer.parseInt(importe);
out.println("Nuevo Importe: " + (importeNumerico+20));
%>
```

Vamos practicar haciendo las comprobaciones con el eclipse. Utilizaremos el mismo proyecto *ejemplos* y crearemos dos ficheros un formulario (fichero HTML) y un JSP.

El fichero formulario cuyo nombre es *enviar* tiene el aspecto que se muestra en la

imagen, como se puede apreciar únicamente pide un nombre y un importe y lo envía a la página *jsp* denominada *recoge01* (asignado con *action*). Obsérvese que el *method* es *post*, luego el importe no es visible para el usuario a través de la URL

#### Formulario *enviar.html*

Posicionarse sobre *WebContent* del proyecto *ejemplos*

*Pulsar botón derecho* → *New* → *HTML* *dar el nombre de enviar* y *Finish*

el código es el siguiente:

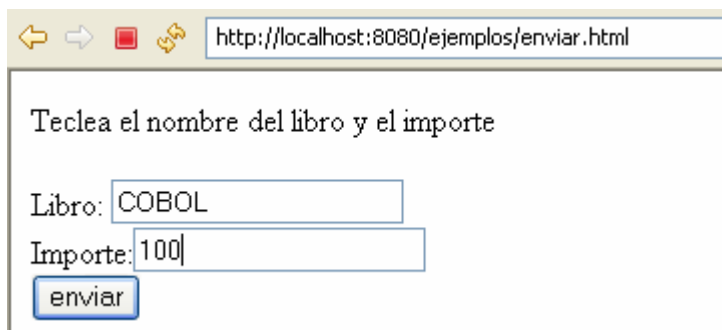
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="recoge01.jsp" method="get">
<p>Teclea el nombre del libro y el importe </p>
  Libro: <input type="text" name="nombre">
  Importe:<input type="text" name="importe">
  <input type="submit" value="enviar">
</form>
</body>
</html>
```

#### Página *recoge01.jsp*

Volvemos a posicionarnos sobre el *WebContent* de ejemplo y realizamos la misma operación esta vez seleccionando el fichero tipo JSP que llamaremos *recoge01*

El código del fichero es el siguiente:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Página JSP que recoge el importe</title>
</head>
<body>
<%
    String libro = request.getParameter("nombre");
    String importe = request.getParameter("importe");
    out.println(libro + " : " + importe);
    out.println(" Concatena: " + (importe + 25));
    int importeNumerico = Integer.parseInt(importe);
    out.println(" Suma " + (importeNumerico + 25));
    %>
</body>
</html>
```



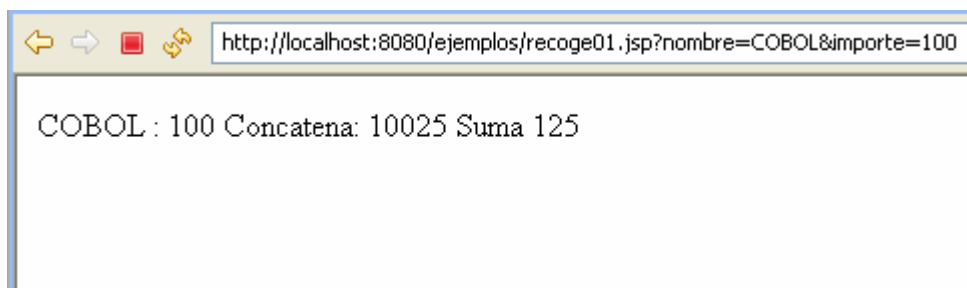
http://localhost:8080/ejemplos/enviar.html

Tecllea el nombre del libro y el importe

Libro:

Importe:

### Página JSP



http://localhost:8080/ejemplos/recoge01.jsp?nombre=COBOL&importe=100

COBOL : 100 Concatena: 10025 Suma 125

Observad que al ser *method* = "get" los datos son visibles en la URL.

Si fuese *method* = "post" la URL sería la siguiente:

<http://localhost:8080/ejemplos/recoge01.jsp>

### 1.1. *request.getParameterNames()* y *request.getParameterValues()*

Con el método *getParameterNames()* recogemos los nombres de los campos del formulario en un array, es útil para saber que campos se quieren recuperar y cuales no. Con el *getParameterValues()* recogemos los valores correspondientes a los campos recuperados con el anterior.

Para comprobar lo anteriormente comentado podemos modificar el fichero HTML *enviar.html* cambiando el destino a la página que llamaremos *recoge02.jsp*

El código de *enviar.html* quedaría como se muestra a continuación. Lo que se pretende es visualizar el nombre de los campos del formulario que en este caso se llaman *nombre* e *importe*, junto con el contenido. Si lanzamos el formulario y tecleamos PASCAL y 1234, el resultado se muestra en la figura que se presenta al final de los ficheros.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="recoge02.jsp" method="post">
<p>Teclea el nombre del libro y el importe </p>
  Libro: <input type="text" name="nombre"><br>
  Importe:<input type="text" name="importe"><br>
  <input type="submit" value ="enviar">
</form>
</body>
</html>
```

El fichero *recoge02.jsp* contiene el siguiente código



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ 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=ISO-8859-1">
<title>ejemplo con getParameterNames y getParametersValues</title>
</head>
<body>
<%
Enumeration variables=request.getParameterNames();
String nom;
String []valor;
while (variables.hasMoreElements()){
    nom=(String)variables.nextElement();
    valor=request.getParameterValues(nom);
    out.println(nom + " : " + valor[0] + "<br>");
}
%>
</body>
</html>
```

Como se ve hay que importar el paquete *java.util* que es donde está la clase *Enumeration*

La página mostrada es la siguiente



Otro ejemplo

El siguiente código

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Formulario Aficiones</title>
</head>
<body>
<form action="valores1.jsp" method="get">
<p> Introduce los siguientes valores </p>
  Libro: <input type="text" name="Libro"><br>
  Autor: <input type="text" name="Autor"><br>
  Editorial: <input type="text" name="Editorial"><br>
  Importe:<input type="text" name="Precio"><br>

  ¿Cuál es tu tema preferido?
<br/><select name="TemaPreferido">
<option>Novela
<option>Poesia
<option>Historia
</select > <br>
  ¿Cuál es tu cine preferido?
<br/>
<input type="radio" name="Cine" value="Accion" checked /> ACCION
<input type="radio" name="Cine" value="Aventura" /> AVENTURA
<input type="radio" name="Cine" value="Comedia" /> COMEDIA

<p>
<br>
  ¿Cuáles son tus aficiones ?
  <input name="Aficion" type="checkbox" value="Deporte" /> Deporte
  <input name="Aficion" type="checkbox" value="Lectura" /> Lectura
  <input name="Aficion" type="checkbox" value="Musica" /> Música
</p>

  <input type="submit" value ="enviar">
</form>
</body>
</html>
```

## Muestra el siguiente formulario

Introduce los siguientes valores

Libro:   
 Autor:   
 Editorial:   
 Importe:   
 ¿Cuál es tu tema preferido?

¿Cuál es tu cine preferido?  
 ACCION  AVENTURA  COMEDIA

¿Cuáles son tus aficiones ?  Deporte  Lectura  Música

El código de la página JSP que muestra los campos y valores es la siguiente

```

<%@ page language="java" import="java.util.*" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Ejemplo con getParameterNames y getParametersValues</title>
</head>
<body>
<%
Enumeration variables = request.getParameterNames();
String variable;
String [] valores;
while (variables.hasMoreElements()){
    variable=(String)variables.nextElement();
    valores=request.getParameterValues(variable);
    if (valores.length >1)
        for ( int i=0;i<valores.length;i++)
            out.println(variable + ":" + valores[i] + "<br>");
    else
        out.println(variable + ":" + valores[0] + "<br>");
}
%>
</body>
</html>
  
```

Muestra lo siguiente

**Nota.**

Editorial:ABC
TemaPreferido:Novela
Autor:ANA
Cine:Accion
Precio:70
Libro:COBOL
Aficion:Deporte
Aficion:Lectura

El eclipse no siempre se comporta de forma lógica, puede suceder que se hagan correcciones en determinado código y el resultado no es el esperado, como sino las tuviese en cuenta. Para intentar corregirlo además de guardar hay que refrescar con F5, o desde el menú File → Refresh. Si aún así no ejecuta podéis copiar la URL del fichero que se lanza en el Explorer y ejecutarlo

desde ahí.

### 1.2.Acción estándar *jsp:forward*

Esta acción permite que la petición sea dirigida a otra página JSP. A otro servlet o a otro recurso. Es muy útil cuando se quiere separar la aplicación en diferentes vistas.

Veamos un ejemplo en el que se utiliza un formulario HTML para pedir el nombre y la contraseña que son enviados a una página *jsp* que los analiza.

Seguimos con el proyecto *ejemplos*, en él creamos un fichero de tipo HTML siguiendo la secuencia de costumbre. Se llamará *credenciales*

El código de formulario es el siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Nombre y contraseña para validar</title>
</head>
<body>
<h1> Ejemplo de forward. Introducir el nombre y la contraseña</h1>
  <form action="compforw.jsp" method="post">
    Nombre: <input type="text" name="nombre"><br>
    clave: <input type="password" name = "passw"><br>
    <input type="submit" name="enviar">
  </form>
</body>
</html>
```

El código *jsp* que comprueba las credenciales no tiene nada de HTML. Incluso veis que se ha quitado las líneas que genera eclipse por defecto.

### Código del fichero *compforw.jsp*

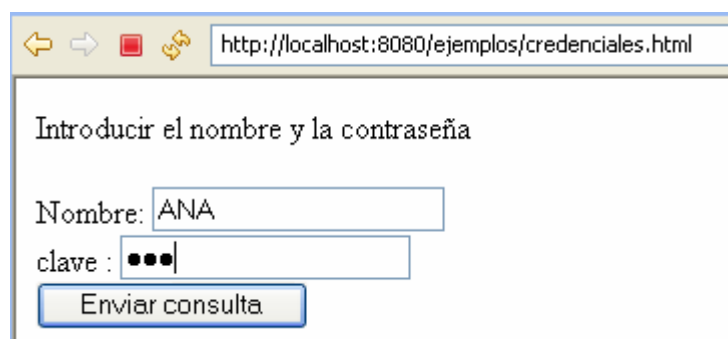
```
<%  
if ((request.getParameter("nombre").equals("ANA")) &&  
    (request.getParameter("passw").equals("XXX"))) {  
%>  
<jsp:forward page="saludaforw.jsp"/>  
<% }else { %>  
<%@ include file="credenciales.html"%>  
<% } %>
```

La línea `((request.getParameter("nombre").equals("ANA"))` recupera el nombre que se teclea en el formulario y al mismo tiempo pregunta si es ANA evaluando true o false. Lo mismo hace la línea de la contraseña (`passw`). Si ambas se evalúan true se redirige a la segunda página *jsp* (*saludaforw.jsp*) para saludar. Si una o ambas evalúan false se recarga la página inicial.

### Código de la página *jsp* que saluda (fichero *saludaforw.jsp*)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
<title>saludos forward</title>  
</head>  
<body>  
<% out.println("BIENVENIDO"); %>  
</body>  
</html>
```

### Página HTML (formulario)



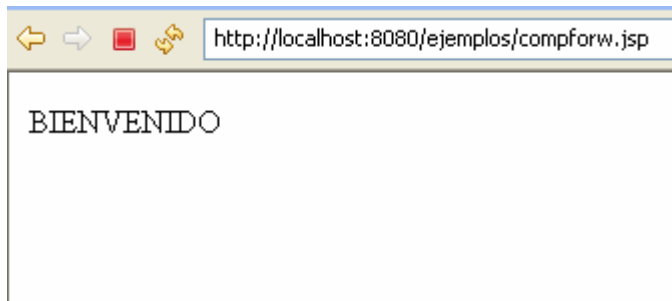
Introducir el nombre y la contraseña

Nombre: ANA

clave : ●●●

Enviar consulta

## Página JSP



## 2. JavaBeans

Se puede definir un JavaBeans como un objeto de java con funcionalidad propia y que puede necesitar algunos datos de entrada con los que después de ejecutar la función encomendada devuelva el resultado requerido.

Se utilizan para reducir al máximo el código Java insertado en una página JSP. En lugar de insertarlo directamente en el fichero JSP se incluye en un objeto y este se llama desde la página JSP.

Permite separar la lógica de ejecución de la presentación. A los atributos de los JavaBeans (llamados propiedades) se accede a través de los métodos `setNombreAtributo` y `getNombreAtributo`.

Si se usa un JavaBeans en una página habrá que definir la clase correspondiente, creando los métodos `set` y `get` para los atributos definidos. El método `get` es llamado de lectura ya que sólo es posible recuperar valores del JavaBeans, mientras que `set` es llamado de escritura porque envía los valores al JavaBean.

La ventaja de utilizar JavaBeans es la de separar el interface de la implementación.

En el JavaBeans los métodos tienen que ser públicos ya que tienen que ser accesibles desde fuera de la clase a través de la página `jsp` mediante las etiquetas `<jsp:setProperty>` y `<jsp:getProperty>`. Sin embargo las variables de instancia se recomienda que sean privadas ya que no deben ser modificadas desde la página `jsp` directamente, se deben utilizar los métodos para acceder a ellas, al tratarse de variables que sólo se utilizan en el JavaBean de forma interna.

## 2.1. Crear un JavaBean

Vamos crear un sencillo JavaBean para sumar dos valores, que serán datos de entrada, y como resultado devolver la suma.

Volvemos al proyecto *ejemplos* y nos posicionamos sobre *Java Resources: src* → **botón derecho** → **new** → **Package** le damos el nombre de *Beans* posicionando el ratón sobre este paquete **botón derecho** → **new** → **class** le damos el nombre de *Sumar* y el resto lo dejamos como está (todo por defecto).

El fichero *Sumar.java* contiene el siguiente código.

```
package beans;

public class Sumar {

    private int num1 = 0;
    private int num2 = 0;

    public void setOpe1 (int num1) {
        this.num1=num1;
    }
    public void setOpe2 (int num2) {
        this.num2=num2;
    }
    public int getResultado() {
        return (num1 + num2);
    }
}
```

Se usa **this** para romper la ambigüedad, dado que el atributo y el parámetro se llaman igual.

## 2.2. Utilizando el JavaBean

Para acceder a la clase *Sumar* desde la página *jsp* utilizamos la acción estándar *jsp:useBean*

En nuestro caso concreto

```
<jsp:useBean id="sumarBean" class="Sumar" scope="request"/>
```

En *id* se define el nombre asignado al JavaBean

En *class* el nombre de la clase del JavaBean

En *scope* el ámbito, en nuestro caso el ámbito de funcionamiento se restringe al momento de petición de la página.

### 2.3. Utilizando el *setProperty* y *getProperty*

Con *jsp:setProperty* se especifica el nombre de la instancia del JavaBean al que se hace referencia, el nombre de la propiedad a la que queremos enviar el dato y el valor a enviar. La etiqueta *jsp:getProperty* recupera el valor devuelto por el JavaBean.

```
<jsp:setProperty      name="id_objeto"      property="nombre_propiedad"|"*"
value="valor"/>
```

Esta acción se puede aplicar a una propiedad (*property="nombre"*) o a todas aquellas propiedades cuyo nombre coincide con parámetros de la petición request (*property="\*"*).

En el siguiente código se muestra como se envían dos valores y se recupera la suma.

Desde el eclipse en el proyecto *ejemplo* se crea el fichero *mostrarsuma.jsp* con el siguiente código.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:useBean id="sumarBean" class="beans.Sumar" scope="request" />
<jsp:setProperty name="sumarBean" property="opel" value="4" />
<jsp:setProperty name="sumarBean" property="ope2" value="5" />
<jsp:getProperty name="sumarBean" property="resultado" />
</body>
</html>
```

#### Si ejecutamos el fichero anterior tiene que mostrar 9 como resultado

En el ejemplo anterior veis que el JavaBean se instancia como *sumarBean*, también se especifica el nombre de la clase junto con el nombre del paquete donde se encuentra



*beans.Sumar* y por último el ámbito de funcionamiento, en este caso se establece para el momento de la petición de la página *request*.

Par enviar datos al JavaBean se utiliza las *setProperty*, en las que se especifica el nombre de la instancia la propiedad y el valor en caso de que se envíen valores. Para recibir se utiliza *getProperty* especificando nombre de la instancia y la propiedad, este formato nos muestra directamente el valor en la pantalla.

Dado que el JavaBean es una clase que está instanciada y sus métodos son públicos podemos hacer referencia a los mismos de forma tradicional como se muestra en el siguiente listado.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:useBean id="sumarBean" class="beans.Sumar" scope="request"/>
<%
sumarBean.setOpe1(8);
sumarBean.setOpe2(9);
int suma=sumarBean.getResultado();
out.println("resultado " + suma);
%>
</body>
</html>
```

Si ejecutamos el fichero anterior mostrará 17. En este caso para mostrar el resultado es necesario utilizar *out.println*

## **OTRA SOLUCIÓN utilizando formulario para introducir los valores**

### ***Fichero HTML***

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>formulario para introducir dos valores y enviarlos a una página
jsp que los suma</title>
</head>
```

```
<body>
<form action="mostrarsuma3.jsp" method="post">
<p>Teclea los valores a sumar </p>
  Valor: <input type="text" name="opel"><br>
  Valor: <input type="text" name="ope2"><br>
  <input type="submit" value ="enviar valores">
</form>
</body>
</html>
```

### Fichero JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:useBean id="sumarBean" class="beans.Sumar" scope="request" />
<jsp:setProperty name="sumarBean" property="*" />
<jsp:getProperty name="sumarBean" property="resultado" />
</body>
</html>
```

Obsérvese que se puede utilizar *property="\*\*"* dado que hacemos coincidir los nombres de los campos del formulario (en minúscula) con los del *Bean* (setNombrecampoformulario)

### ACTIVIDAD

Hacer una aplicación que realice las siguientes operaciones:

- Introducir dos valores desde un formulario
- Página *jsp* que los suma y muestra el resultado.

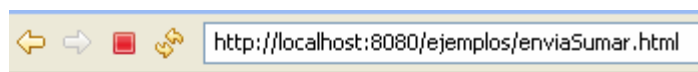
### SOLUCION

Fichero HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>formulario para introducir dos valores y enviarlos a una página
```

```
jsp que los suma</title>
</head>
<body>
<form action="sumaValores.jsp" method="post">
<p>Teclea los valores a sumar </p>
  Valor: <input type="text" name="dato1"><br>
  Valor: <input type="text" name="dato2"><br>
  <input type="submit" value="enviar valores">
</form>
</body>
</html>
```

### Formulario



Teclea los valores a sumar

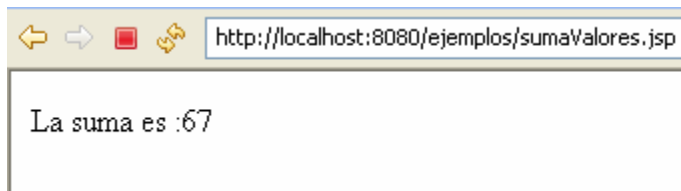
Valor:

Valor:

### Fichero JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <%
String valor1 = request.getParameter("dato1");
String valor2 = request.getParameter("dato2");
int valor1Numerico = Integer.parseInt(valor1);
int valor2Numerico = Integer.parseInt(valor2);
out.println("La suma es :" + (valor1Numerico + valor2Numerico));
  %>
</body>
</html>
```

### Página que muestra el resultado



### 3. Aplicación formulario.

Vamos crear una aplicación en la que intervienen:

- Un fichero **.html** para visualizar una imagen tipo *gif*.
- Dos fichero **.jsp**, uno para pedir el nombre y otro para visualizar el saludo
- Un fichero **.java**, bean con los métodos set y get para los nombres de los elementos del formulario

#### Fichero HTML

El siguiente código muestra el gif denominado Merlin.

El nombre del fichero es **foto.html**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table border="0" width="400" cellspacing="0" cellpadding="0">
<tr>
<td height="150" width="150"> &nbsp; </td> <td width="250"> &nbsp; </td>
</tr>
<tr>
<td width="250"> &nbsp; </td>
<td align="right" width="350">
 </td>
</tr> </table> <br>
</body>
</html>
```

#### Fichero JAVA

Es la clase (JavaBean) que define las propiedades y los métodos set y get para los nombres de los elementos del formulario

El nombre del fichero es **NombreUser.java**

```
package beans;

public class NombreUser {
    private String usuario=null;
    public void setUsu( String usuario ) {
        this.usuario = usuario;
    }
    public String getUsu() {
        return usuario;
    }
}
```

### Fichero JSP.

El siguiente código muestra la página principal en la que se pide un nombre

El nombre del fichero es **holaUser.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ include file="foto.html" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Página principal</title>
</head>
<body bgcolor="#ffffff" background="background.gif">
<table border="0" width="700">
<tr>
<td width="150"> &nbsp; </td>
<td width="550">
<h1>Mi nombre es Merlin ¿el tuyo?</h1> </td>
</tr>
<tr>
<td width="150"> &nbsp; </td>
<td width="550">
<form method="get">
<input type="text" name="usu" size="25"> <br>
<input type="submit" value="Enviar">
</form>
```

```
</td>
</tr>
</table>
<%if ( request.getParameter("usu") != null ) { %>
<% include file="respuestaHola.jsp" %>
<%}%>
</body>
</html>
```

## Fichero JSP

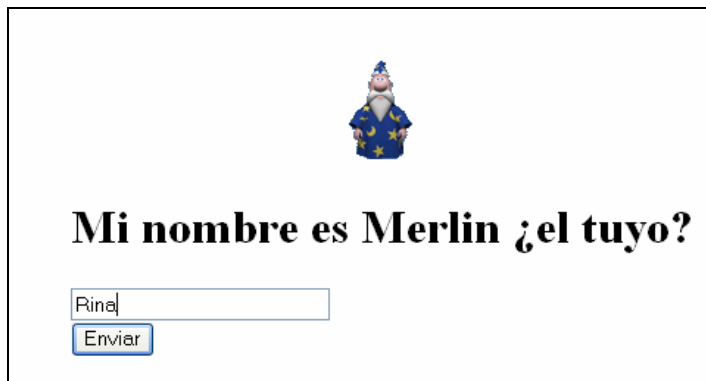
En este fichero se muestra el saludo

El nombre del fichero es **respuestaHola.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table border="0" width="700"> <tr>
<td width="150"> &nbsp; </td>
<td width="550">
<jsp:useBean id="miclase" scope="page" class="beans.NombreUser" />
<jsp:setProperty name="miclase" property="*" />
<h1>Hola, <jsp:getProperty name="miclase" property="usu" />! </h1>
</td> </tr> </table>
</body>
</html>
```

## Vistas de las páginas

Página que pide el nombre:



A screenshot of a web page. At the top center is a small cartoon wizard character with a blue hat and a blue robe with yellow stars. Below the character, the text "Mi nombre es Merlin ¿el tuyo?" is displayed in a bold, black, serif font. Underneath the text is a text input field containing the name "Rina". Below the input field is a blue button with the text "Enviar" in white.

Página que muestra el saludo



A screenshot of a web page, similar to the one above. It features the same wizard character at the top center. Below the character, the text "Mi nombre es Merlin ¿el tuyo?" is displayed in a bold, black, serif font. Underneath the text is an empty text input field. Below the input field is a blue button with the text "Enviar" in white. At the bottom of the page, the text "Hola, Rina!" is displayed in a bold, black, serif font.

#### 4. Gestión de errores

Cuando se desarrolla una aplicación se producen errores. Los de sintaxis, aquellos que se producen porque se olvida poner algún punto, alguna coma o se escribe algo mal, son los llamados errores de compilación, de los mismos nos avisa el sistema más o menos explícitamente de forma que es fácil encontrarlos y corregirlos.

El problema surge cuando los errores se producen en tiempo de ejecución, son errores de lógica y si no están controlados provoca que la aplicación se pare inesperadamente y el usuario que la maneja no sabe que acción realizar. Errores de este tipo se producen, por ejemplo, cuando en una expresión aritmética intervienen valores no numéricos, o intentar una división por cero, etc.

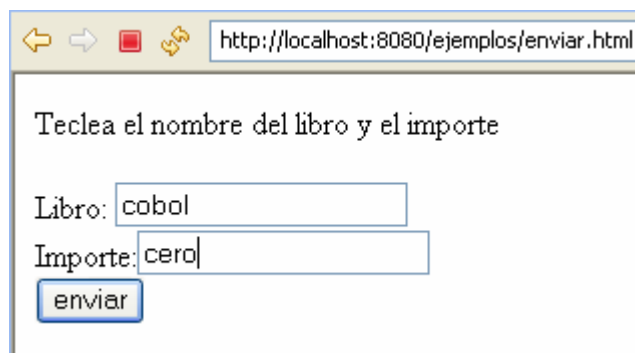
Los errores en tiempo de ejecución hay que controlarlos, se pueden utilizar diferentes técnicas una de ellas es utilizar las sentencias *try-catch*

Vamos a modificar el código del ejemplo en el que se enviaba un nombre y un importe para calcular una suma. Se controlará que los datos que se introducen sean correctos.

```
<%  
String libro;  
int importeNumerico;  
try {  
    libro = request.getParameter("nombre");  
    importeNumerico =  
    Integer.parseInt(request.getParameter("importe"));  
    out.println(" Suma      " + (importeNumerico + 25));  
    }  
    catch (Exception e) {  
        out.println("Se produjo un error " + e.getMessage());  
    }  
%>
```

Al establecer un bloque *try* se consigue que una excepción no interrumpa la ejecución, los errores que se puedan producir en ese bloque se capturan con el manejador de excepciones *catch*

Si introducimos los siguientes datos



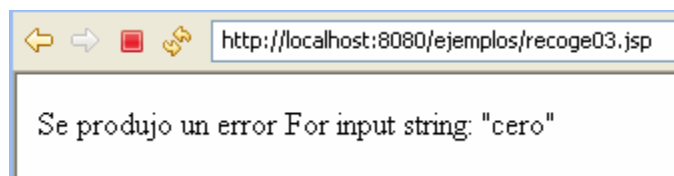
http://localhost:8080/ejemplos/enviar.html

Teclea el nombre del libro y el importe

Libro:

Importe:

Se visualiza el siguiente mensaje de error:



http://localhost:8080/ejemplos/recoge03.jsp

Se produjo un error For input string: "cero"



Se puede utilizar la directiva *page* para capturar cualquier tipo de error, con esa directiva se bifurca a la página que trata el error, en esa página utilizamos el atributo

*isErrorPage = "true"*

ver el siguiente código:

Página donde se produce el error y que llama a la página *error.jsp*

```
<%@ page errorPage="error.jsp"%>
<html>
<head>
<title>ejemplo con errorPage</title>
</head>
<body>
  <%
    int i=10;
    i = i / 0;
  %>
</body>
```

Página *error.jsp* donde se muestra el error

```
<%@ page isErrorPage="true" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>captura los errores</title>
</head>
<body>
<h2>Su aplicación ha generado un error</h2>
<h3>Por favor mire el código</h3>
<b>Excepción:</b><br>
<%= exception.toString() %>
</body>
</html>
```

## 5. Sesiones

Una sesión es una serie de comunicaciones entre un cliente y un servidor, al mismo tiempo utilizando la sesión es posible hacer un seguimiento del usuario a través de la aplicación.

La sesión se crea cuando el usuario se conecta la primera vez y finaliza cuando abandona el sitio web, bien porque alcanzó el tiempo previamente establecido o porque se cerró el servidor.

Un ejemplo de utilización de sesiones es en las aplicaciones de comercio electrónico, en este tipo de aplicaciones vamos navegando por diferentes páginas para elegir los productos que interesan, sin el uso de las sesiones no se podría guardar la información ya que al ir cambiando de página se perderían los productos que se van comprando.

Mediante el uso de sesiones es posible guardar objetos en la sesión de forma que mientras dure la misma esos objetos asociados estarán siempre disponibles. Para poder utilizar una sesión el atributo *session* de la directiva *page* tiene que tener el valor *true*.

Las acciones que se pueden hacer sobre las sesiones son realizadas mediante el interface *HttpSession*, incluido en el paquete *javax.servlet.http*, y los métodos que implementa.

### **5.1. Trabajando con sesiones**

Para trabajar con sesiones lo primero que se debe hacer es obtener la sesión que se ha creado de forma automática cuando el usuario accedió por primera vez al servidor. Se utiliza el método *getSession* para obtener el interface de tipo *HttpSession*. Una vez creado el objeto podemos acceder a información asociada a la sesión. Por ejemplo, con *getId* se obtiene el identificador asociado a la sesión, con *getCreationTime* se conoce la fecha y la hora de creación, el método *getLastAccessedTime* fecha y hora del último acceso al servidor.

El siguiente código muestra:

- El identificador de la sesión
- Cuándo se creó
- Cuánto tiempo se lleva conectado, o se cuánto tiempo se lleva navegando por la página web.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" session="true" import="java.util.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

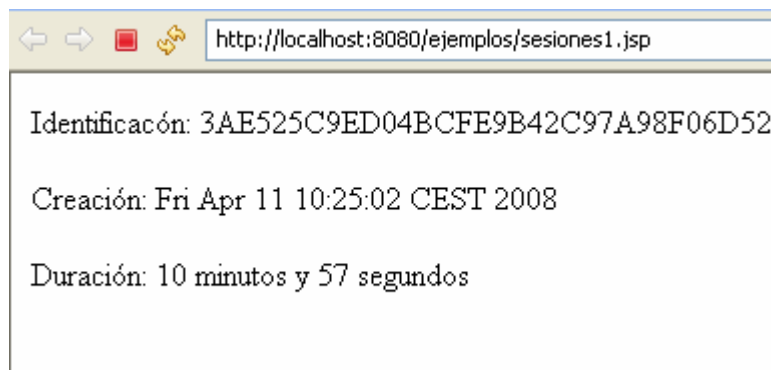
```

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>trabando con sesiones</title>
</head>
<body>
<%
//obtener la sesión
HttpSession sesion1 =request.getSession();
//muestra la identificación
out.println("Identificación: " + sesion1.getId() + "<br> <br>");
//obtener y mostrar la fecha de creación
Date fecha = new Date(sesion1.getCreationTime());
out.println("Creación: " + fecha + "<br> <br>");
//obtener y mostrar el tiempo de conexión
Long duracion = sesion1.getLastAccessedTime() -
                sesion1.getCreationTime();
Date verDuracion = new Date(duracion);
out.println("Duración: " + verDuracion.getMinutes() + " minutos y " +
verDuracion.getSeconds() + " segundos");
%>
</body>
</html>

```

Fijaros que en la directiva *page* se incluye `session="true"` para trabajar con la sesión y `import="java.util.*"` para trabajar con las fechas.

El resultado se muestra en la siguiente vista



Se puede prescindir del interface *sesion1* y trabajar directamente con la variable predefinida *session*. El código anterior quedaría de la siguiente forma.

```

<%
out.println("Identificación: " + session.getId() + "<br> <br>");
//obtener y mostrar la fecha de creación
Date fecha = new Date(session.getCreationTime());

```

```
out.println("Creación: " + fecha + "<br> <br>");  
//obtener y mostrar el tiempo de conexión  
Long duracion = session.getLastAccessedTime() -  
session.getCreationTime();  
Date verDuracion = new Date(duracion);  
out.println("Duración: " + verDuracion.getMinutes() + " minutos y " +  
verDuracion.getSeconds() + " segundos");  
%>
```

## 5.2. Guardar y recuperar objetos en las sesiones

Es importante tener la posibilidad de guardar objetos en una sesión para no perder la información al pasar de página. Por ejemplo, supongamos que se desea ver en la cabecera de cada página el nombre de usuario que se pidió en la primera, si no se almacena al pasar de página se perdería.

El método para guardar un objeto es **setAttribute()** con dos argumentos, el primero es la variable donde se guarda el objeto para recuperarlo posteriormente y el segundo es el dato que se desea guardar

```
Sesion1.setAttribute("nombre", "ANA");
```

El segundo valor debe ser un objeto, si se desea guardar, por ejemplo (int), un entero primero hay que convertirlo

```
Integer edad = new Integer(30);  
Sesion1.setAttribute("edad", edad);
```

En el primer ejemplo no hizo falta convertirlo ya que se trata de un objeto tipo string, no así en el caso de int, long, etc.

Para recuperar los objetos guardados se utiliza el método **getAttribute()**

```
Sesion1.getAttribute("nombre");
```

El objeto que devuelve no establece de qué tipo se trata. Si se conoce previamente la naturaleza hay que convertirlo mediante una operación llamada *casting*. Para realizar esta

conversión se añade al nombre de la sesión la naturaleza, como se muestra en el siguiente ejemplo.

```
(String)sesion1.getAttribute("nombre");  
Integer edad = (Integer)sesion1.getAttribute("edad");
```

Si no existe un objeto almacenado con el nombre del identificador especificado devuelve nulo que el contenedor JSP traduce en un error. Por lo tanto se debe controlar esta posible eventualidad

### 5.3. Caso práctico.

Un supuesto de utilización de sesiones es aquel en el que para acceder a una página exige que previamente se valide con el nombre del usuario y la contraseña. Si son incorrectas no permite acceder a la página, visualiza un mensaje de error, si se intenta entrar directamente en la página sin identificarse se le redirige a la página de identificación.

**Página JSP (identificacion.jsp)** en la que el usuario se tiene que identificarse con nombre y contraseña

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1" session="true"%>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
<title>Página de identificación</title>  
</head>  
<body>  
<%  
    if (request.getParameter("error") != null) {  
        out.println(request.getParameter("error"));  
    }  
%>  
<form action="validacion.jsp" method="post">  
    usuario.....: <input type="text" name="usuario" size="25"><br>  
    contraseña: <input type="password" name="clave" size="10"><br>  
                <input type="submit" value="enviar">  
</form>  
</body>  
</html>
```

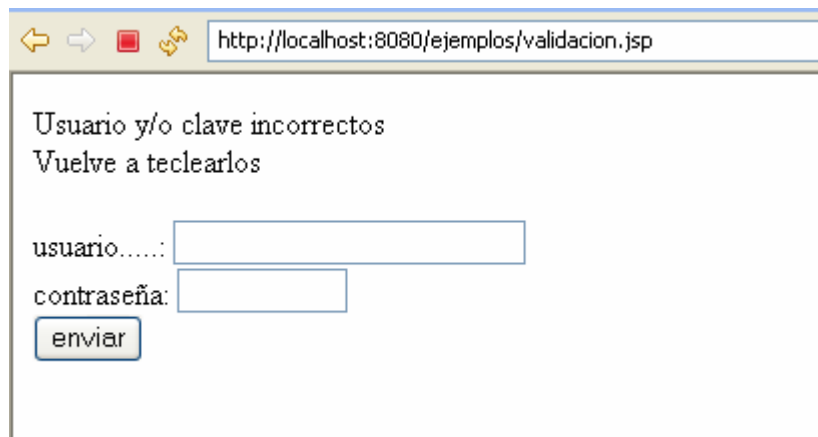
## Página de identificación

**Página de validación (validación.jsp)** que recoge el usuario y contraseña tecleados y comprueba que son los correctos, si es así crea un objeto tipo sesión y guarda el nombre del usuario para mostrarlo en las páginas de la aplicación y pasa, mediante la etiqueta `<jsp:forward>`, el control a la página inicial de las aplicaciones mostrando en la cabecera el usuario que está conectado. En caso de que el usuario o la contraseña fueran incorrectas se redirecciona a la página de inicio (identificación).

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" session="true"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>proceso de validación</title>
</head>
<body>
<%
String usuario=null;
String clave=null;
if (request.getParameter("usuario") !=null)
usuario=request.getParameter("usuario");
if (request.getParameter("clave") !=null)
clave=request.getParameter("clave");
if (usuario.equals("MERLIN")&&clave.equals("XXXX")) {
    HttpSession sesion1=request.getSession();
    sesion1.setAttribute("usuario",usuario);
%>
<jsp:forward page="menu.jsp"></jsp:forward>
<%
}else {
%>
    <jsp:forward page="identificacion.jsp">
    <jsp:param name="error" value="Usuario y/o clave
        incorrectos<br>Vuelve a teclearlos"/>
```

```
</jsp:forward>
<%
}
%>
</body>
</html>
```

Página que se muestra en el caso de que la validación haya fallado

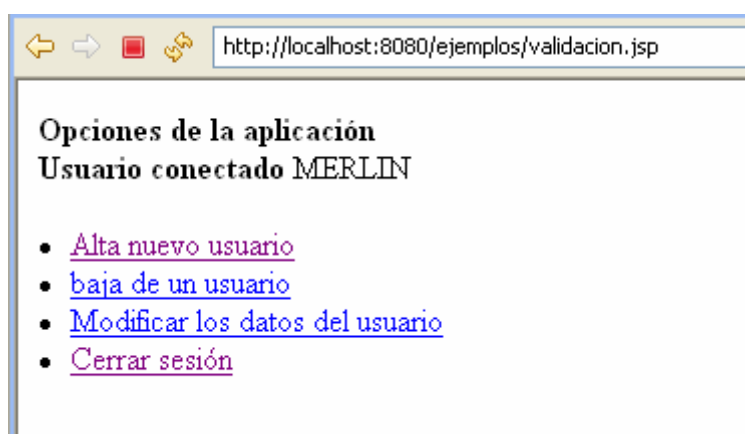


**Página inicial de la aplicación (menú.jsp)**, en esta página se controla que el usuario se identificó previamente, de no ser así se le muestra el mensaje de que debe identificarse y se redirecciona a la página de identificación. Si el usuario se identificó de forma correcta pasa a mostrar las opciones de la aplicación y la posibilidad de cerrar la sesión, en este caso llama al fichero *jsp* que cierra la sesión.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" session="true"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>El programa principal</title>
</head>
<body>
<%
String usuario=null;
HttpSession sesion1=request.getSession();
if (sesion1.getAttribute("usuario")==null) {
%>
<jsp:forward page="identificacion.jsp">
<jsp:param name="error" value="Hay que identificarse"/>
```

```
</jsp:forward>
<%} else {
    usuario=(String)sesion1.getAttribute("usuario");
}
%>
<b>Opciones de la aplicación</b><br>
<b>Usuario conectado</b>
<%=usuario%><p>
<li><a href="altas.jsp">Alta nuevo usuario</a>
<li><a href="bajas.jsp">baja de un usuario</a>
<li><a href="modificar.jsp">Modificar los datos del usuario</a>
<li><a href="cerrarSesion.jsp">Cerrar sesión</a>
</body>
</html>
```

Página que se muestra en el caso de que la validación se haya realizado correctamente



Página que se muestra en el caso de que el usuario se salte la validación, se accede directamente a la URL de la página de aplicación



**Fichero JSP que cierra la sesión (cerrarSesion.jsp)**, este fichero cierra la sesión con el método **invalidate()** y devuelve el control a la

página inicial de identificación.



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" session="true"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>cierra la sesión y vuelve a la identificación</title>
</head>
<body>
<%
HttpSession sesion1=request.getSession();
if (sesion1!=null) sesion1.invalidate();
%>
<jsp:forward page="identificacion.jsp"></jsp:forward>
</body>
</html>
```

## 6. Cookies

Una cookie es un elemento de información de tamaño reducido, un fichero de texto, enviado por el servidor para que el navegador lo guarde de manera que se pueda recuperar su valor en cualquier momento. Una de las características del protocolo HTTP es que se trata de un protocolo "sin memoria". Esto significa que, siempre que el usuario sigue un enlace y la página se carga en el navegador, es como si se hiciera por primera vez, no existiendo ninguna información disponible con respecto a lo que se cargo o dejó de descargar anteriormente. Uno de los objetivos de las cookies es la de reconocer al usuario que se conecta al ordenador. Una de las páginas que recoge la petición de usuario puede comprobar si existe una cookie que ha dejado anteriormente, si es así sabe que el usuario ha visitado ese sitio y por lo tanto puede leer los valores que lo identifiquen.

Las cookies nacen con la intención de proporcionar un mecanismo para almacenar en el ordenador de cada usuario datos con los que establecer un sistema de comunicación entre las distintas páginas que participan en una aplicación Web. Uno de los casos más típicos es el que se produce cuando se registra en un sitio web. Cuando el mismo usuario visita de nuevo esas páginas, es reconocido de forma automática. Esto es así porque la primera vez que se identifico se creó una cookie en el ordenador local de dicho usuario con los datos necesarios para ser reconocido.

Un ejemplo de utilización son los checkbox de algunas páginas de forma que cuando el usuario realiza un registro o solicita un alta permiten recordar el nombre y el usuario sólo tiene que teclear la clave.

Para trabajar con cookies se utiliza la clase *Cookie* que esta disponible en el paquete *javax.servlet.http*

Todas las cookies tienen un período de caducidad. Cuando este tiempo se cumple, la cookie desaparece. Teniendo en cuenta este comportamiento cabe distinguir cuatro tipos de cookies:

- Cookies de sesión: desaparecen cuando se acaba la sesión, entendiéndose por sesión el período de tiempo que un usuario está en un sitio web de manera continuada con la misma ventana del navegador.
- Cookies permanentes: si la fecha de caducidad de la cookie se corresponde con un futuro lejano, se puede decir que la cookie que es permanente. Esto no es del todo cierto, como se ha observado, ya que los navegadores cuentan con un espacio limitado para almacenar cookies de manera que las más nuevas hacen desaparecer a las más viejas cuando dicho espacio está totalmente ocupado.
- Cookies con fecha de caducidad: a veces los sitios web establecen cookies con una fecha de caducidad concreta. Por ejemplo, se puede utilizar una cookie para recordar a un cliente que existe una oferta. Esta cookie tendrá necesariamente la fecha de caducidad de la propia oferta, más allá de la cual no tienen ningún sentido que siga existiendo.
- Cookies para borrar: borrar una cookie existente es como volver a escribirla con una fecha de caducidad anterior a la fecha actual.

### **6.1. Crear cookies.**

Una cookie almacenada en el ordenador del usuario está compuesta por un nombre y un valor asociado a la misma, además puede tener asociados una serie de atributos que contienen información acerca de su tiempo de vida, alcance, dominio, etc.

Para crear un objeto de tipo cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar.

```
Cookie miCookie = new Cookie("usuario","merlin");
```

También es posible crear cookies con contenido que se genera dinámicamente, en el siguiente ejemplo se muestra una cookie que guarda texto con la fecha actual.

```
Cookie miCookie = null;
Date fecha = new Date();
String texto = "Esta es mi cookie " + fecha;
miCookie = new Cookie("usuario",texto);
```

En el siguiente ejemplo se guardan datos que provienen de un formulario

```
Cookie miCookie = null;
String ciudad = request.getParameter("formCiudad");
miCookie = new Cookie("miCiudad",ciudad);
```

Creada la cookie hay que establecer una serie de atributos para que se pueda utilizar, el primero de los atributos es el que se conoce como tiempo de vida. Por defecto las cookies se mantienen mientras dura la ejecución del navegador, cuando se cierra el navegador las cookies que no tienen establecido un tiempo de vida se destruyen. Por lo tanto si se quiere una cookie dure más tiempo y siga disponible es necesario establecer un valor de tiempo.

En el siguiente ejemplo se utiliza el método **setMaxAge()** para establecer un tiempo de vida de 31 días

```
miCookie.setMaxAge(60*60*24*31);
```

(segundos\*minutos\*horas\*días)

La cookie se destruye después de ese tiempo, si el valor es cero o negativo, se destruye cuando se cierre el navegador.

Otro atributo es el *path* que indica desde donde es visible la cookie. Por ejemplo con `setPath(/)` es visible desde todo el sitio web si se establece `setPath(/datos)` sólo es visible desde dentro del directorio.

Para conocer el valor del *path* se utiliza el método **getPath()**

```
Out.println("La cookie es visible en " + miCookie.getPath());
```

También puede ser necesario establecer el ámbito, si no se especifica la propiedad *domain*, se entiende que la cookie es visible desde el dominio que la creó, si se especifica un nombre de dominio se entiende que es visible en los dominios que contengan el nombre especificado.

```
miCookie.setDomain(".paginasjsp.com");
```

En el ejemplo anterior la cookie está disponible para todos los dominios que contengan el nombre `".paginasjsp.com"`

Para crear el fichero con la cookie se utiliza el método **addCookie()** de la interface `HttpServletResponse`

```
response.addCookie(miCookie);
```

Una vez ejecutada la línea anterior, la cookie ya existe en el disco del usuario que ha accedido a la página JSP

## 6.2. Recuperar cookies

Para recuperar una cookie es necesario recorrerlas todas y quedarnos con la que nos interesa. Para obtener las cookies se crea un array del tipo Cookie y se utiliza el método `getCookies()` para recuperarlas.

En el ejemplo siguiente vemos como se crea una cookie con el nombre de *usuario* valor *merlin* y tiempo para expiración  $60*60*24*2$ , es tiempo en segundos.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" 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=ISO-8859-1">
<title>crear cookies</title>
</head>
<body>
<%
Cookie miCookie = new Cookie("usuario","merlin");
miCookie.setMaxAge(60*60*24*2);
response.addCookie(miCookie);
%>
</body>
</html>
```

Fijaros que la directiva `page` incorpora `import = "java.util.*"%>`

El siguiente código muestra como se recorren las cookies de nuestro puesto visualizando la información correspondiente a la que se creó con el ejemplo anterior

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" 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=ISO-8859-1">
<title>recuperar cookies</title>
</head>
<body>
<%
Cookie[] lasCookies = request.getCookies();
for (int i=0; i<lasCookies.length; i++)
{
```

```
Cookie miCookie = lasCookies[i];
if (miCookie.getName().equals("usuario"))
{
    out.println("Nombre de mi cookie...  :" +
                miCookie.getName() + "<br>");
    out.println("Contenido de la cookie  :" +
                miCookie.getValue() + "<br>");
    out.println("Tiempo de vida.....   :" +
                miCookie.getMaxAge() + "<br>");

    break;
}
%>
</body>
</html>
```

En el ejemplo se va recorriendo el *array* con el *for* y cuando se encuentra la cookie *usuario* se visualiza la información correspondiente al nombre, el valor y el tiempo de vida.

# Desarrollo de Aplicaciones Informáticas

materiales didácticos de aula



UNIÓN EUROPEA  
Fondo Social Europeo



Gobierno del Principado de Asturias  
Consejería de Educación y Ciencia



FORMACIÓN PROFESIONAL  
Principado de Asturias