

Java Server Pages (JSP)

Programación Web-Dinámico



Introducción a JSP

- Java incrustado en páginas web (en y entre tags HTML).
- Cuando se utiliza apropiadamente, es una forma sencilla para generar páginas web dinámicas.
- Cuando no se utiliza apropiadamente, se puede llegar a tener código terriblemente desordenado y complejo.
- Recomendación: Tratar de mantener el código Java incrustado lo más sencillo posible. Pueden utilizarse clases externas (helper classes) – Java Beans.

Ciclo de Vida

- Una página JSP es traducida a un Servlet.
- El Servlet es compilado (en Tomcat, la compilación sucede la primera vez que la página es solicitada). La primera solicitud de una página puede ser muy lenta!
- Después de la compilación, la página JSP es tan rápida como un Servlet (porque ahora es un Servlet).

Ejemplo: Hola Mundo

Archivo **testDate.jsp**

```
<html>
  <head> <title> Hola JSP </title> </head>
  <body>
    <p> Hola Mundo:
      <%=new java.util.Date()%>
    </p>
  </body>
</html>
```

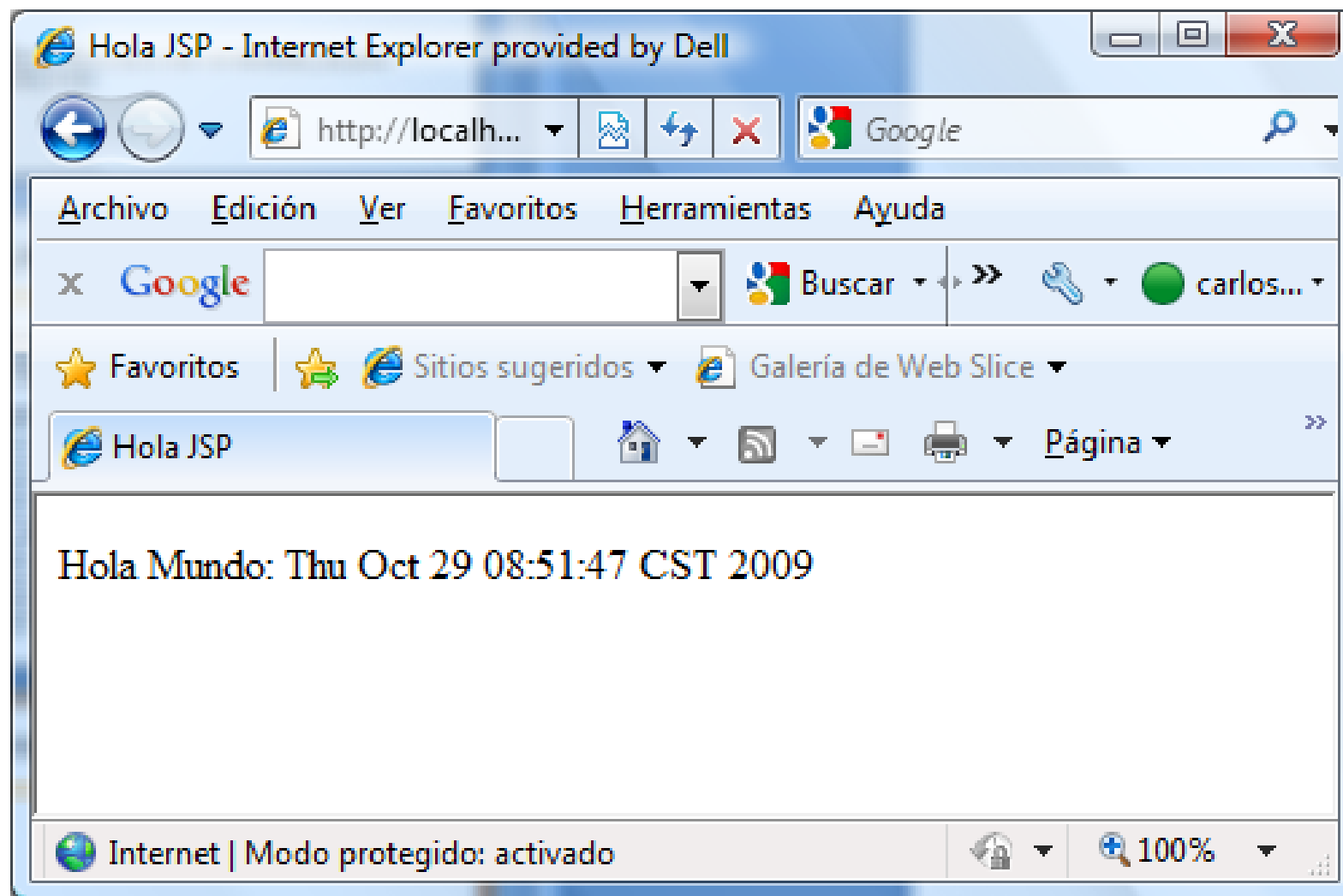
Esta página es traducida al Servlet **testDate_jsp.java**
(ver siguiente slide)

testDate_jsp.java (fragmento de código)

Este fragmento de código (tomado de **testDate_jsp.java**) muestra la parte que produce la salida. Este archivo es creado en el directorio **Tomcat/work**:

```
out = pageContext.getOut();
_jspx_out = out;
out.write("<html>\n");
out.write("<head><title>Hola JSP</title></head>\n");
out.write("<body>\n");
out.write("<p> Hola Mundo:\n");
out.print(new java.util.Date());
out.write("\n");
out.write("</p>\n");
out.write("</body>\n");
out.write("</html>\n");
out.write("\n");
```

Salida de testDate.jsp



Construcciones básicas

- Hasta el momento hemos visto literales:
Por ejemplo: `<html>`
Esto es copiado directamente a la salida (al browser).
- También hemos visto expresiones:
Por ejemplo: `<%=new java.util.Date()%>`
Regresan un valor que es incluido en la salida.
- Pero también tenemos:
Directivas, declaraciones y scriptlets.

Directivas

- Son instrucciones para el compilador.
- Ejemplos:

Para incluir otra página (en tiempo de compilación)

```
<%@ include file="header.jsp"%>
```

Para importar algunos paquetes de Java
(separados por comas si se desea añadir más)

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


Declaraciones

- Son usadas para declarar variables y métodos.
- Van en la sección de declaraciones del Servlet.
- Una vez que se declaran pueden ser usadas en la JSP.
- Ejemplos:

```
<%! int count = 0 %>
```

```
<%! double sqr(double x) {  
    return x * x; } %>
```

Scriptlets

- Son secciones de código Java incrustadas en la página.
- A diferencia de las expresiones, los scriptlets no regresan un valor.
- Pueden usarse para escribir directamente a la página.
Por ejemplo: `response.getWriter()`
- Se ejecutan cada vez que la página es solicitada

Ejemplo

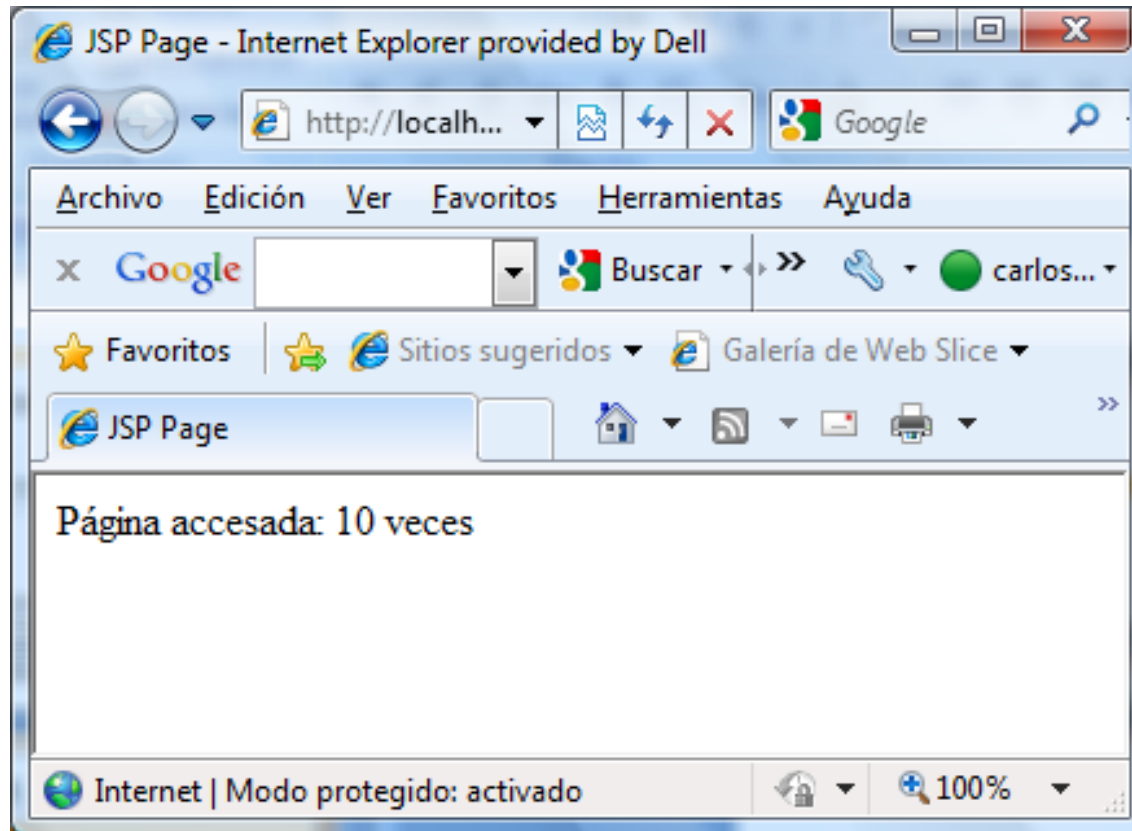
- Este ejemplo demuestra algunos de los conceptos mencionados:

```
<%! int n = 0; %>
Página accesada <%= ++n %> veces
<% if ( (n % 10) == 0 ) {
    n = 0;
}
%>
```

- ¿Qué sucede cuando se solicita esta página?
- ¿Qué sucede si solicitas la página desde sesiones diferentes (desde diferentes instancias de browsers)?
- ¿Qué valores toma ***n*** si refrescas la página?

Ejemplo (continuación)

- ¿Qué sucede al refrescar la página?



Request & Response

- Cada página JSP tiene acceso a dos objetos especiales.
- El objeto **Request** lleva información pasada por la solicitud HTTP (HTTP Request) hecha por el browser. Esto incluye cualquier dato contenido en la forma.
- El objeto **Response** es usado para pasar información de regreso al cliente (browser).
Por ejemplo: `response.getWriter()` proporciona un flujo de salida para escribir directamente en el cliente.

Manejo de Formas con JSP

- Con JSP es sencillo el manejo de las formas HTML.
- Se puede utilizar `request.getParameter()` para obtener los valores enviados dentro de una forma.
- También se puede definir un Java Bean para capturar los valores semi-automáticamente.
- Veremos esto en acción con un ejemplo sencillo.

Formas HTML

- Las formas HTML permiten al usuario pasar información al servidor web. Algunos ejemplos de controles HTML que permiten la captura de información son los siguientes:
 - Campos de texto (una sola línea)
 - Campo Password (una sola línea, texto escondido)
 - Areas de texto (múltiples líneas)
 - Selección (combo box, menú pop-up)
 - Radio-button (1 selección de n posibles opciones)
 - Check-box (m selecciones de n posibles opciones)
 - Botón *Browse* (para búsqueda de archivos)
 - Botón *Submit* (para enviar la forma con los datos)

Generación de Formas HTML

- Las fomas deben estar bien presentadas (por ejemplo, alineadas en una tabla).
- Es necesario que los campos de entrada tengan un nombre (su atributo ***name***). Esto es para que posteriormente puedan ser accedados y extraer sus valores.
- Ejemplos:

```
<input type="text" name="nombre" />
```

```
<input type="text" name="apellidoMaterno" />
```

```
<input type="text" name="apellidoPaterno" />
```


Procesamiento de Formas HTML

- La forma puede especificar si los datos serán enviados usando una solicitud GET o POST. Esto se hace directamente en el atributo ***method*** de la forma. POST es el método más común.
- El servlet que recibirá la solicitud, debe de implementar el método doPost() para procesar la forma. En un JSP, el servlet que es generado automáticamente por el proceso de compilación se encarga de implementar este método.

Procesamiento de Formas HTML

- JSP esconde los detalles de GET/POST .
- Para recuperación de datos en JSP utilizamos:
`request.getParameter()`
`<jsp:setProperty>`

Formato de los datos enviados

- Cuando una forma es generada, se puede especificar el formato de los datos. Por default es texto plano.
- También los datos pueden ser en un formato MIME.
- MIME permite transmitir datos binarios (usando codificación base 64).
- Dado que se pueden transmitir datos binarios, archivos de cualquier tipo pueden ser transmitidos a un servidor web.

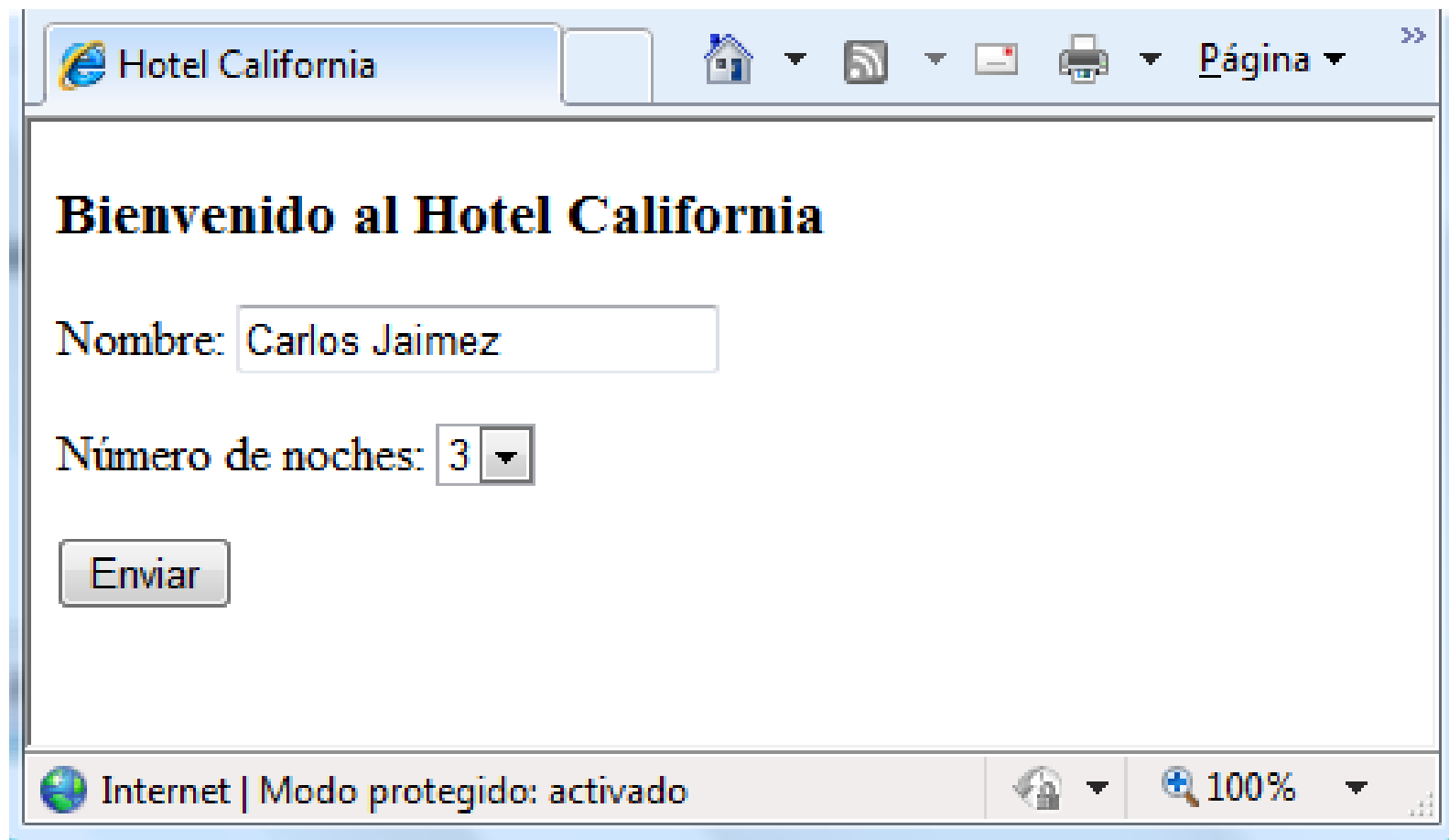
Arquitecturas para procesamiento de los datos

- Helper classes para servlets.
Para generar formas (y otros elementos HTML) .
Para procesar formas.
- JSP + Java Beans (veremos un ejemplo).
- JSP + Tag Library (no cubierto en este curso).
- XForms (no cubierto en este curso, pero vale la pena leer acerca de ellas si están interesados).
- MS InfoPath (no cubierto en este curso – puede generar automáticamente formas a partir de esquemas XML).

Java Beans

- Hay de dos tipos:
 - Simple (cubiertos en este curso).
 - Enterprise Java Beans (EJB: más complejos).
- Java Beans Simple:
 - Definen atributos/propiedades/campos.
 - Definen métodos get/set para leer/escribir sus atributos.
 - Pueden ligarse a datos.
- Veamos el siguiente ejemplo.

Forma para Reservación del Hotel California



The image shows a web browser window with the following elements:

- Address Bar:** "Hotel California" with navigation icons (Home, RSS, Mail, Print) and a "Página" dropdown menu.
- Page Content:**
 - Header:** "Bienvenido al Hotel California" in a large, bold, black serif font.
 - Form Fields:**
 - Nombre:** A text input field containing "Carlos Jaimez".
 - Número de noches:** A dropdown menu with the value "3" selected.
 - Submit Button:** A button labeled "Enviar".
- Status Bar:** "Internet | Modo protegido: activado" with a shield icon, a magnifying glass icon, and "100%" zoom level.

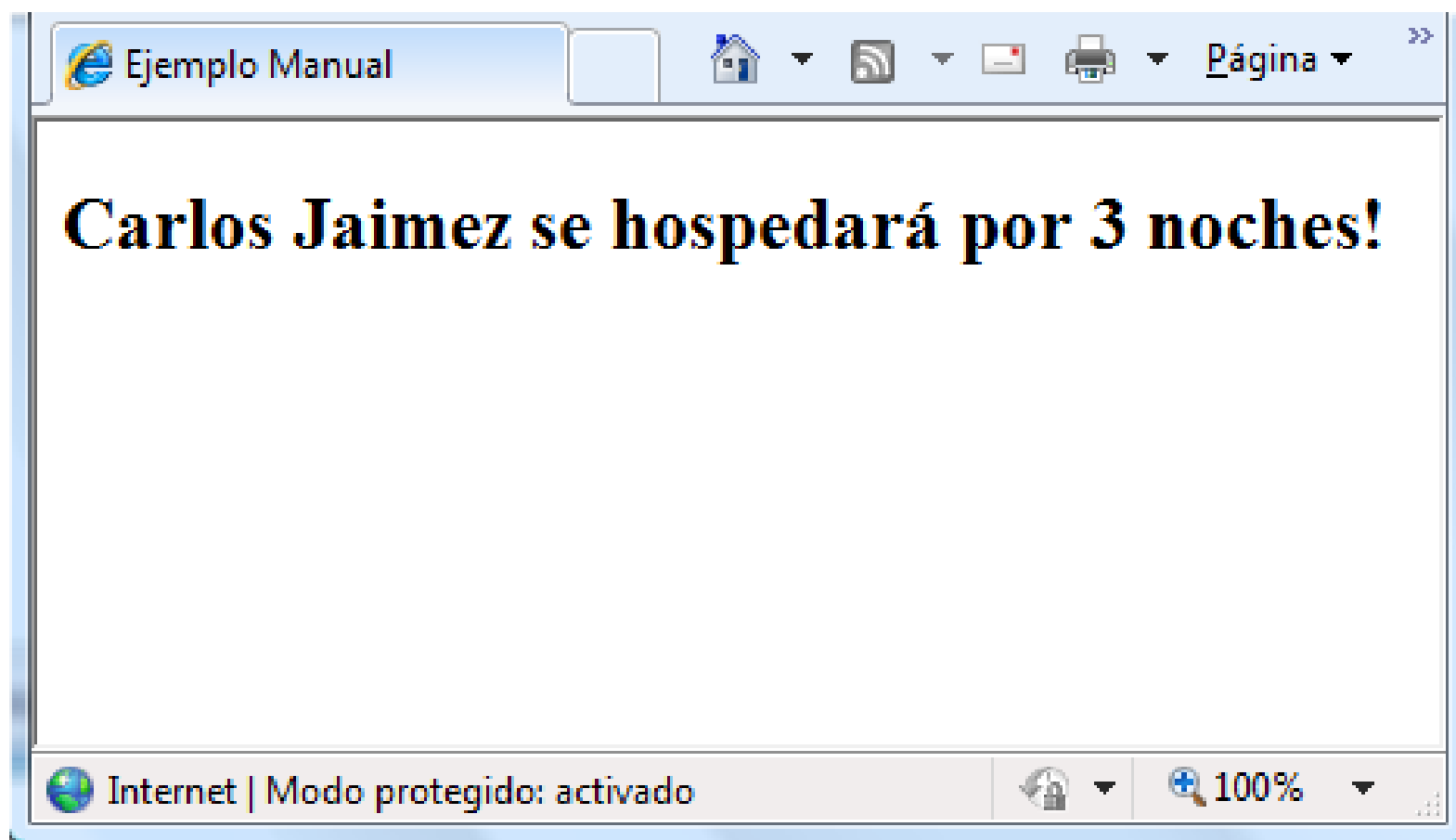
Forma de Reservación (HTML - body)

```
<body>
  <h3>Bienvenido al Hotel California</h3>
  <form method="post" action="bookingFormDisplay.jsp">
    <p>Nombre:
    <input type="text" name="nombre" size="20"></p>
    <p>Número de noches:
    <select size="1" name="numNoches">
      <option selected="">1</option>
      <option>2</option>
      <option>3</option>
    </select></p>
    <p><input type="submit" value="Enviar" name="b1">
    </p>
  </form>
</body>
```

Accesando los valores enviados (Manualmente)

```
<html>
  <head><title>Ejemplo Manual</title></head>
  <body>
    <h2>
      <%=request.getParameter("nombre")%>
      se hospedar&aacute; por
      <%=request.getParameter("numNoches")%> noches!
    </h2>
  </body>
</html>
```


Página resultante



Accesando los valores enviados

Versión Java Bean (Automáticamente)

```
<jsp:useBean id='hotelBeanObject'  
            scope='page'  
            class='beans.HotelBean'  
/>  
  
<jsp:setProperty name='hotelBeanObject' property='*' />  
  
<html>  
  <head><title>Ejemplo Java Bean</title></head>  
  
  <body>  
    <h2>  
      <%=hotelBeanObject.getNombre() %>  
      se hospedar&aacute; por  
      <%=hotelBeanObject.getNumNoches() %> noches!  
    </h2>  
  </body>  
</html>
```

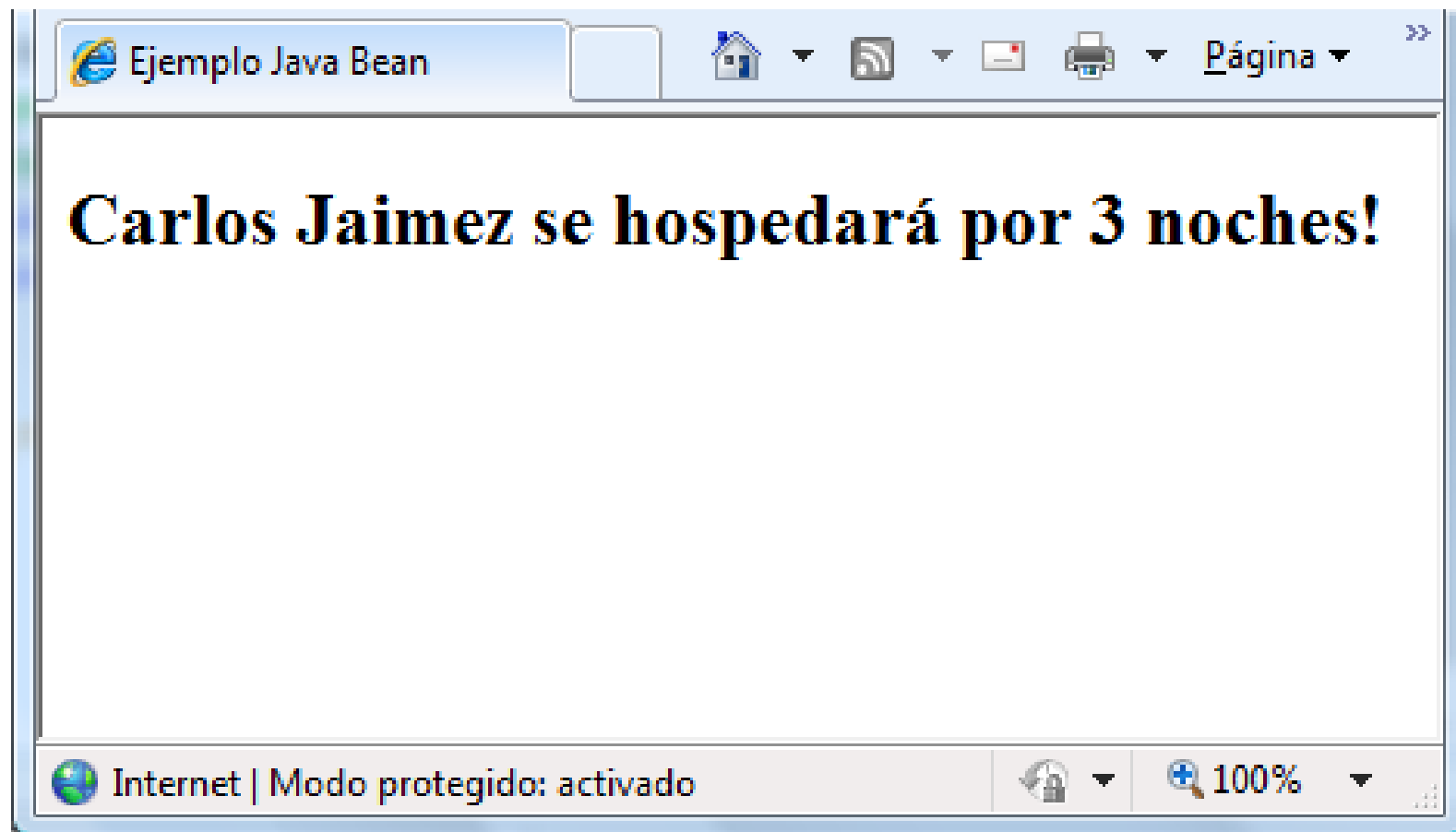
Código del Java Bean

```
package beans;

public class HotelBean {
    String nombre;
    int numNoches;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getNumNoches() {
        return numNoches;
    }
    public void setNumNoches(int numNoches) {
        this.numNoches = numNoches;
    }
}
```

Página resultante



Alcance de un Java Bean

- Observa que en nuestro ejemplo el alcance es especificado de la siguiente manera:

```
scope= 'page '
```

- El alcance tiene cuatro variantes:

Page. El Java Bean existe durante la ejecución de esa página solamente.

Request. Lo mismo que en el alcance de página, pero además el Java Bean sobrevive cualquier solicitud **forward** o **include**.

Alcance de un Java Bean (continuación)

Session. El Java Bean existe para múltiples solicitudes dentro de una misma aplicación web, desde una instancia particular del web browser.

Este tipo de alcance es normalmente utilizado para beans que desean mantenerse por cada usuario de la aplicación. Por ejemplo, los carritos de compras, etc.

Alcance de un Java Bean (continuación)

Application. El Java Bean existe para todas las solicitudes de todos los usuarios, para todas aquellas páginas que lo usen. Beans de este tipo mueren cuando el servidor web es reiniciado.

Este tipo de alcance puede ser utilizado para mantener conexiones a bases de datos, contadores globales, etc.

Java Beans vs Manual

- Para el ejemplo que acabamos de ver, tenemos las siguientes diferencias:
 - La página JSP con Java Bean es más compleja
 - El Java Bean requiere ser definido como una clase externa
- Pregunta de discusión:
 - ¿Cuál sería el(los) criterio(s) para determinar si las versiones con Java Beans son mejores que las versiones manuales?