Objetivos de la sesión:

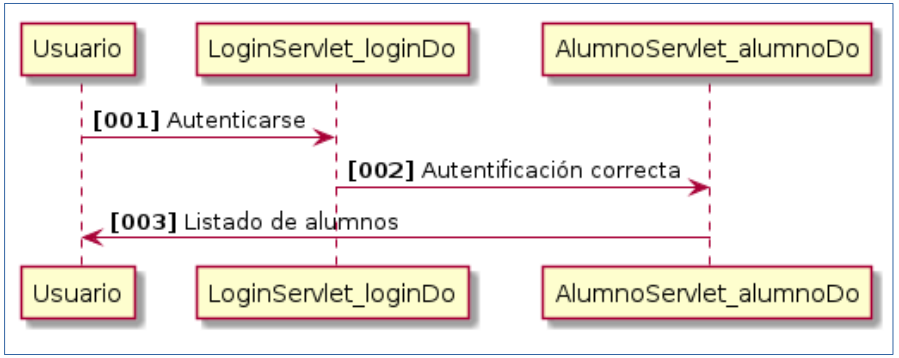
- ***Listar datos*** utilizando ***JSTL*** para mejorar la presentación en el navegador.
- Entender la diferencia entre ***atributos*** y ***variables de sesión***
- Modificar la webapp para realizar ***altas y bajas de datos en el listado*** .

Hasta ahora hemos visto como logearnos correctamente en nuestra aplicación, pero si tuviéramos una lista de alumnos en nuestra aplicación ...

¿Como podemos modificar nuestra aplicación para mostrarlos?

Listar datos:

· Para que nuestra aplicación pueda mostrar una lista de datos (por ejemplo alumnos) en las siguientes diapositivas realizaremos los siguientes ***pasos***:



1º Crear la clase Alumno

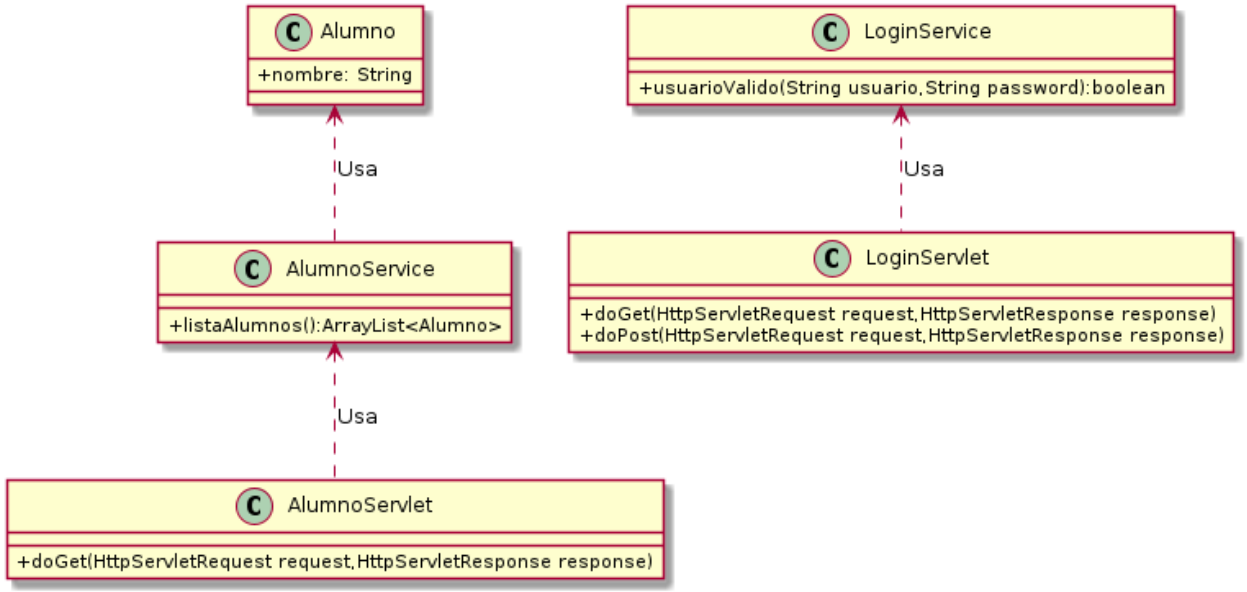
2º Crear un servicio AlumnoService

3º Modificar LoginServlet para pasar la lista de Alumnos como un atributo a la página bienvenida.jsp

4º Refactorizar

5º Redirigir al nuevo servlet AlumnoServlet y alumno.jsp

6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos



Listar datos:

1º Crear la clase Alumno

Hay que crear una clase "Alumno" con un atributo privado "nombre" de tipo String en el mismo package donde esta LoginServlet. Adicionalmente deberemos añadir getters, setters, el constructor y el método toString().

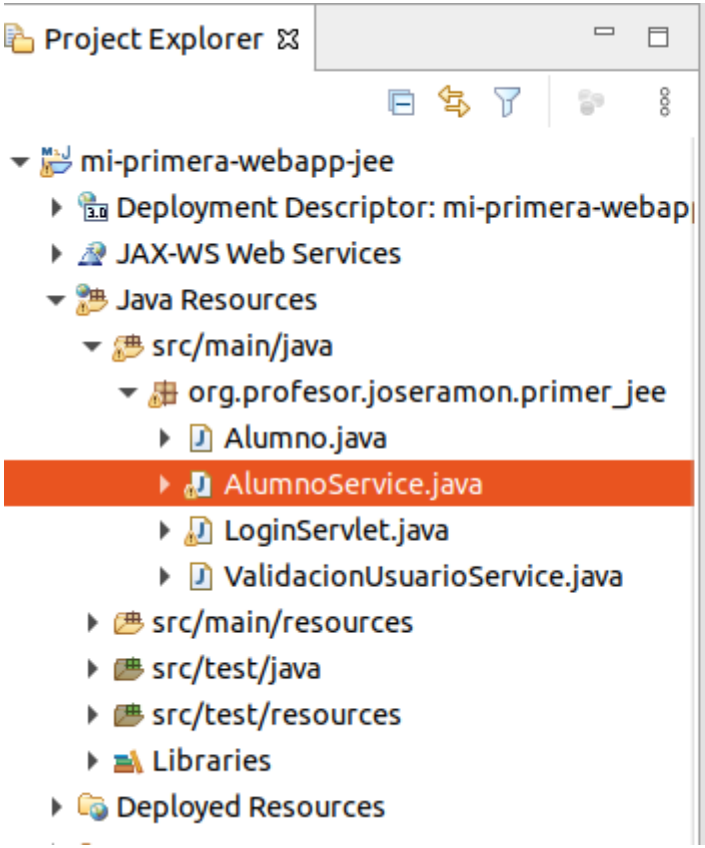
Para los alumnos que no esten familiarizados con el Eclipse, teniendo el fichero Alumno.java abierto si pulsamos dentro con el botón derecho tenemos las opciones:

"Source\Generate Getters and Setters…"
"Source\Generate constructors using fields…" y
"Source\Generate toString()…" .

Listar datos:

2º Crear un servicio AlumnoService

Hay que crear una clase "AlumnoService" y contendrá a una lista estática de alumnos predefinida y un método listaAlumnos():



AlumnoService.java

```java
package org.profesor.joseramon.primer_jee;
import java.util.ArrayList;
import java.util.List;
import org.profesor.joseramon.primer_jee.Alumno;
public class AlumnoService {
    private static List<Alumno> alumnos= new ArrayList<Alumno>();
    //En una situación real aquí se realizaria un acceso a la BD para obtener
    //los alumnos, pero en esta unidad didáctica aun no lo haremos.

    //1º Para evitar tener la lista de alumnos en blanco al arrancar la app
    //debemos crear un bloque estático para inicializar el ArrayList al principio
    //con 3 valores: "Jose", "Pedro" y "Juan"
    static {
        alumnos.add(new Alumno("Jose"));
        alumnos.add(new Alumno("Pedro"));
        alumnos.add(new Alumno("Juan"));
        alumnos.add(new Alumno("Maria"));
    }

    //2º Crear un método listaAlumnos() que devuelva la lista de alumnos

}
```

Listar datos:

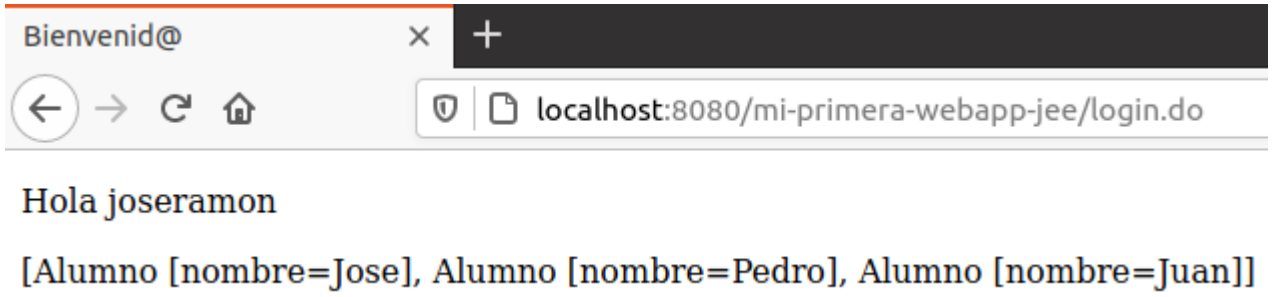
3º Modificar LoginServlet para pasar la lista de Alumnos como un atributo a la página bienvenida.jsp

Hay que modificar LoginServlet para que bienvenida.jsp pueda mostrar los alumnos y modificar bienvenida.jsp como se muestra en la imagen.

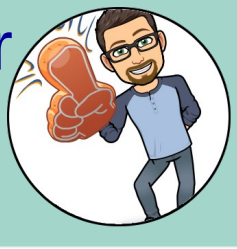
Ayuda: Hará falta implementar el método toString() en Alumno para que se muestre "Alumno [nombre=" + nombre + "]".

bienvenida.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>Bienvenid@</title>
8  </head>
9  <body>
10 <p>Hola ${nombre}</p>
11 <p>${alumnos}</p>
12 </body>
13 </html>
```

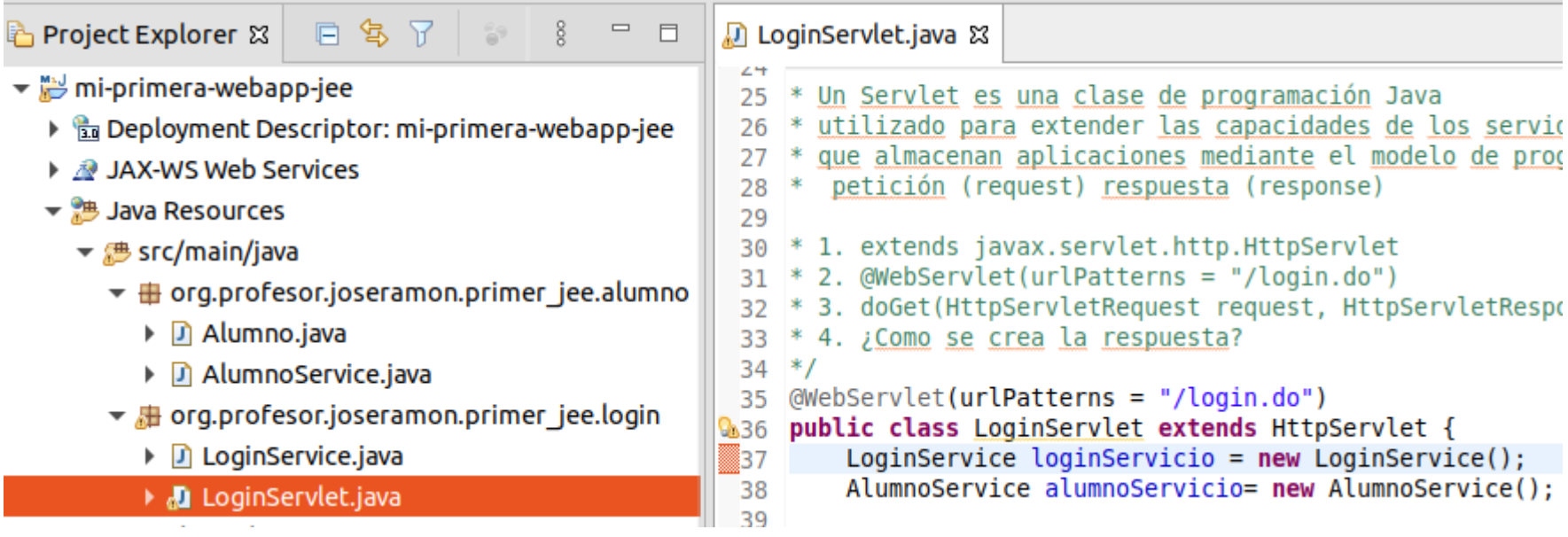


¿ Por que bienvenida.jsp muestra la lista de alumnos simplemente pasandole la lista de alumnos con ${alumnos} ?



Listar datos:

4º Refactorizar : En una aplicación real cada cierto tiempo debemos refactorizar para mejorar la legibilidad de nuestro código. En nuestro paquete principal de clases java empiezan a acumularse las clases, por lo que vamos a modificar LoginServlet para que esté dentro del subpaquete "login". También modificaremos ValidacionUsuarioService para que se llame LoginService y esté también en el paquete login y también moveremos Alumno y AlumnoService a un nuevo subpaquete "alumno". Por último cambiaremos los nombres a los 2 servicios disponibles en LoginServet como en la imagen:



Listar datos:

5º Redirigir al nuevo servlet AlumnoServlet y alumno.jsp

Es más lógico que cuando el login sea correcto LoginServlet nos envie a alumno.jsp (donde se mostrarán los alumnos) en vez de ir a bienvenida.jsp.

Para conseguirlo debemos crear un nuevo servlet AlumnoServlet en el paquete alumno cuya url será "/alumno.do". Para rellenar AlumnoServlet podemos copiar y pegar instrucciones de LoginServlet y dejar solo el servicio de alumnos y el doGet que añadirá el atributo "alumnos" y redirigirá a alumno.jsp.

Adicionalmente deberemos de crear alumno.jsp (podemos copiar información de bienvenida.jsp) .

Hasta ahora hemos reenviado de un servlet a una página JSP, pero …

alumno.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>Alumnos</title>
8  </head>
9  <body>
10 <p>${alumnos}</p>
11 </body>
12 </html>
```

¿Como podemos redirigir desde el servlet LoginServlet hasta AlumnoServlet cuando el login sea correcto?

Listar datos:

... continuación 5º Redirigir al nuevo servlet AlumnoServlet y alumno.jsp

Vamos a redirigir el servlet LoginServlet hasta AlumnoServlet utilizando "sendRedirect()":

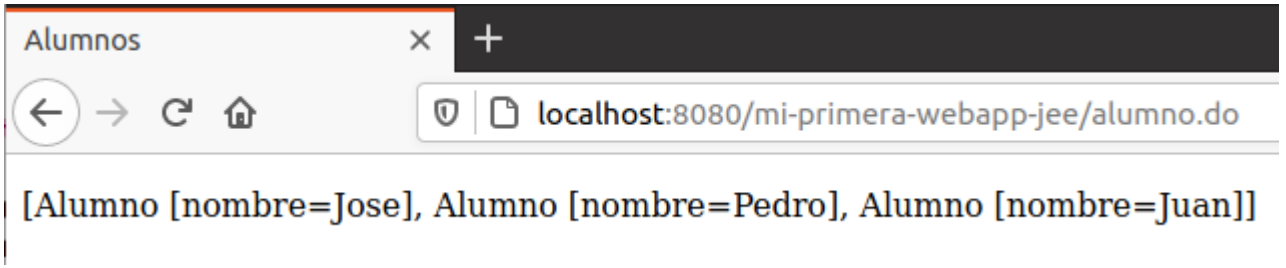
LoginServlet.java

```
36 public class LoginServlet extends HttpServlet {
37     LoginService loginServicio = new LoginService();
38     AlumnoService alumnoServicio= new AlumnoService();
39
40     @Override
41     protected void doGet(HttpServletRequest request,
42             HttpServletResponse response) throws IOException, ServletException {
43         request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
44     }
45     @Override
46     protected void doPost(HttpServletRequest request,
47             HttpServletResponse response) throws IOException, ServletException {
48
49         String nombre =request.getParameter("nombre");
50         String password =request.getParameter("password");
51
52         if (loginServicio.usuarioValido(nombre, password)) {
53             //validación correcta: Redirigir al Servlet de alumno
54             response.sendRedirect("alumno.do");
55         } else {
56             //validación incorrecta
57             request.setAttribute("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
58             request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
59         }
60     }
61 }
```

Listar datos:

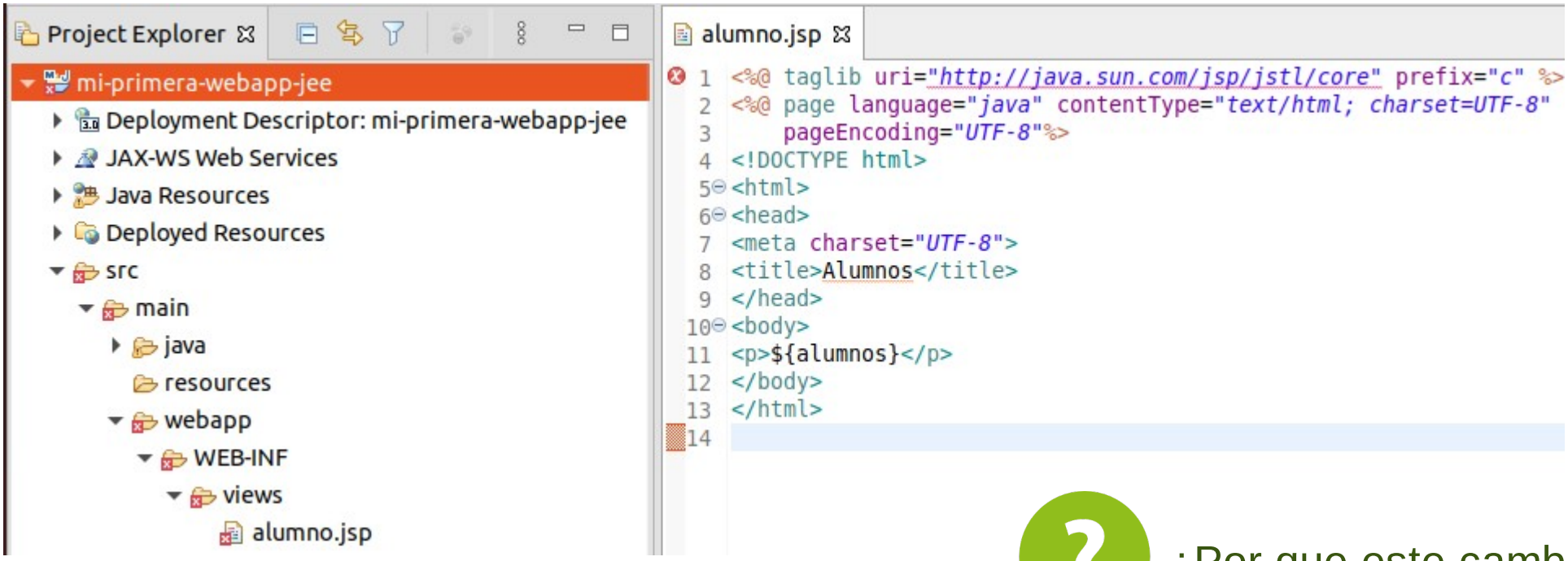
... continuación 5º Redirigir al nuevo servlet AlumnoServlet y alumno.jsp

Ahora debemos comprobar desde el navegador que cuando introducimos un login correcto nos redirige a alumno.do y nos muestra el listado:



Listar datos:

6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos: JSTL es el acrónimo de ***Java Server Pages Standard Tags Library***. JSTL es una librería de tags que nos ayudará a codificar nuestras páginas JSP. Para utilizar JSTL tenemos que añadir la librería de tags (JSTL) a la página JSP como se puede observar en la primera linea del fichero JSP. Sin realmente utilizar los tags de JSTL comprobemos que cambia al ejecutar la aplicación web:



¿Por que este cambio?

Listar datos:

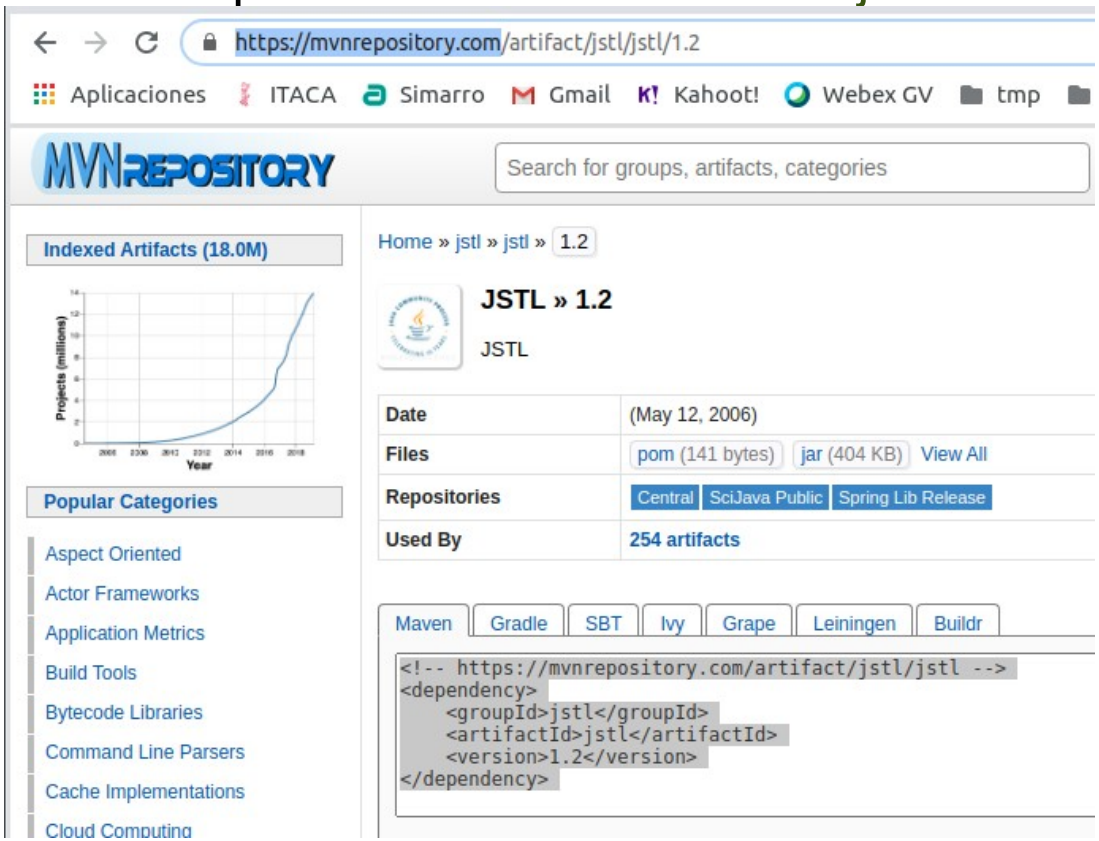
... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

El error nos aparece porque realmente aunque le hemos dicho que utilice la librería de tags JSTL en la página JSP, ***en nuestro proyecto no existe la librería porque no le hemos dicho al Maven que la incorpore***.

Listar datos:

... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

Para solucionarlo agregamos la nueva dependencia en el fichero pom.xml. Si queremos saber que código añadir al fichero pom.xml para utilizar un artefacto y cual es la última versión de dicha dependencia podemos teclear “maven jstl” en el buscador del navegador y nos irá a una página de la web https://mvnrepository.com/ donde podemos ver todas las versiones y el código a añadir en la sección de dependencias al fichero pom.xml si queremos utilizar dicho artefacto.

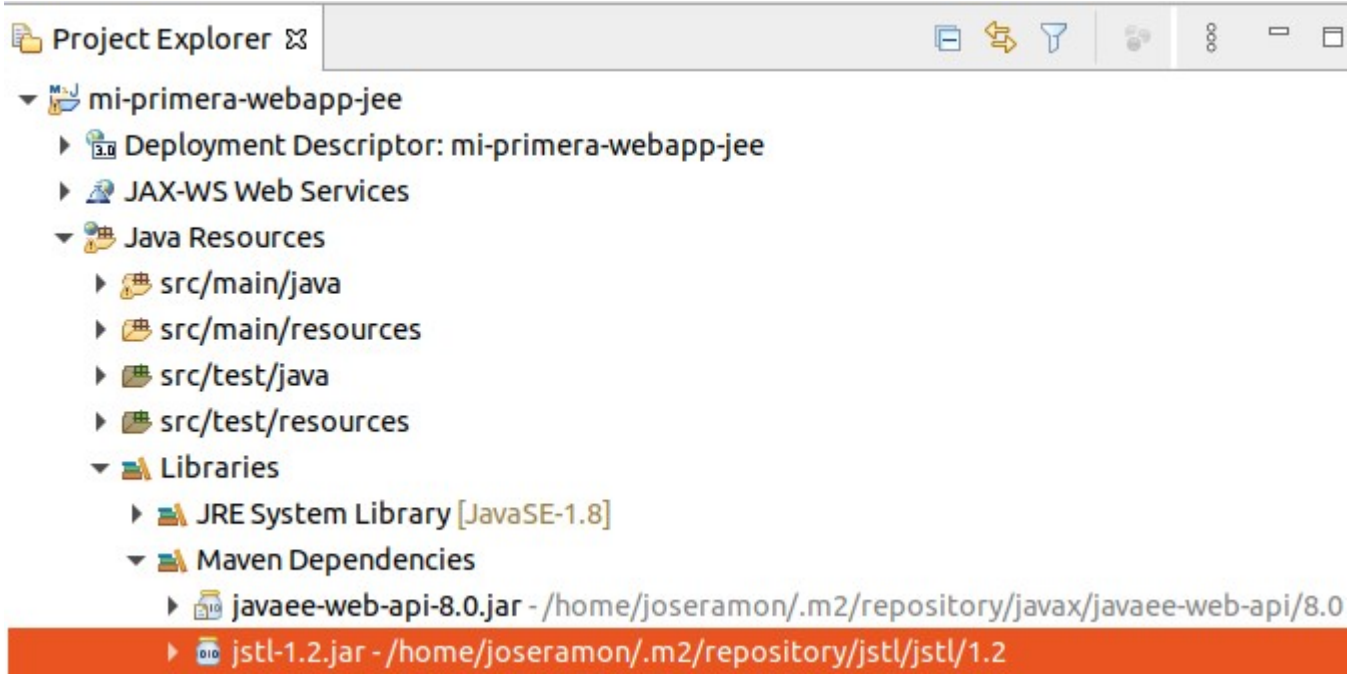


No nos obsesionemos con poner la última versión de la dependencia porque a veces son incompatibles con otras dependencias que ya tenemos en uso.

Listar datos:

... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

Si lo hemos hecho bien veremos que en las ***librerías de Maven*** en nuestro proyecto ya aparece la ***librería jstl-1.2.jar***:



¿Para que vamos a gastar esta librería?

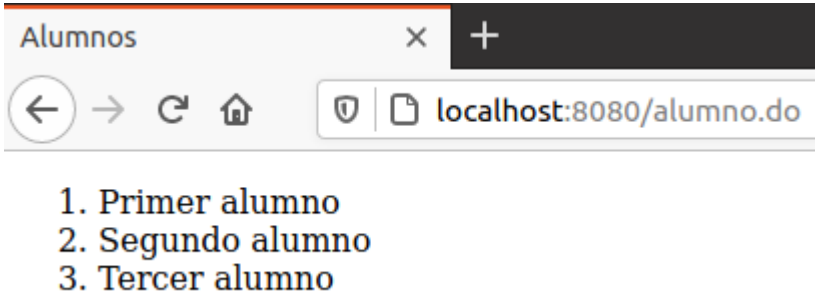
Listar datos:

... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

En HTML si queremos imprimir una lista utilizamos los tags ***<ol>*** y ***<li>*** como podemos ver en el ejemplo:

alumno.jsp

```
1  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2  <%@ page language="java" contentType="text/html; charset=UTF-8"
3      pageEncoding="UTF-8"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>Alumnos</title>
9  </head>
10 <body>
11 <ol>
12     <li>Primer alumno</li>
13     <li>Segundo alumno</li>
14     <li>Tercer alumno</li>
15 </ol>
16 </body>
17 </html>
```



Nuestra intención es utilizar la librería para crear un bucle y poder rellenar dicha lista en base a los alumnos.

¿Como?

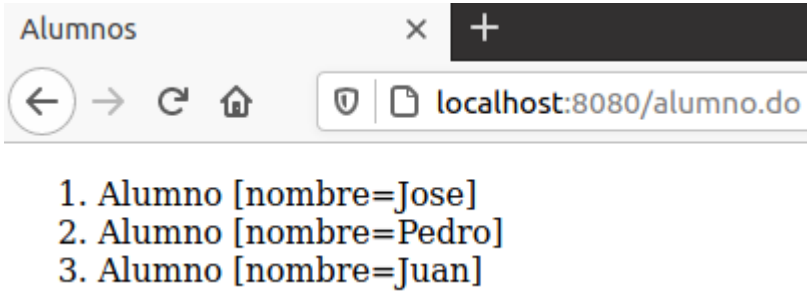
Listar datos:

... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

Para ***crear un bucle*** utilizamos el ***tag forEach*** y como el prefijo de la librería le hemos dicho que es "c" ***(prefix="c")*** al añadir la librería, nuestro tag realmente será ***c:forEach***. El prefijo podemos cambiarlo al declarar la librería si nos interesara. Cambiemos alumno.jsp y comprobemos el resultado :

alumno.jsp

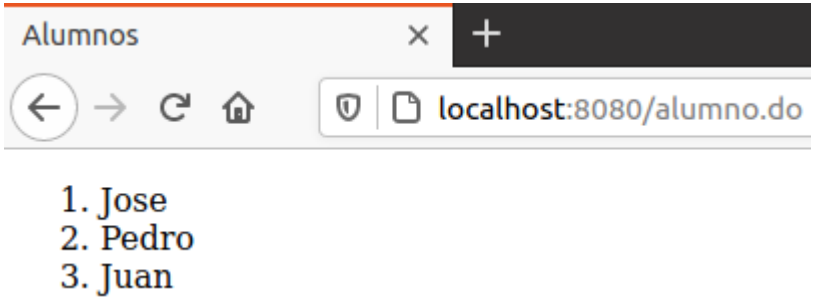
```
1  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2  <%@ page language="java" contentType="text/html; charset=UTF-8"
3      pageEncoding="UTF-8"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>Alumnos</title>
9  </head>
10 <body>
11 <ol>
12     <c:forEach items="${alumnos}" var="alumno">
13         <li>${alumno}</li>
14     </c:forEach>
15 </ol>
16 </body>
17 </html>
```



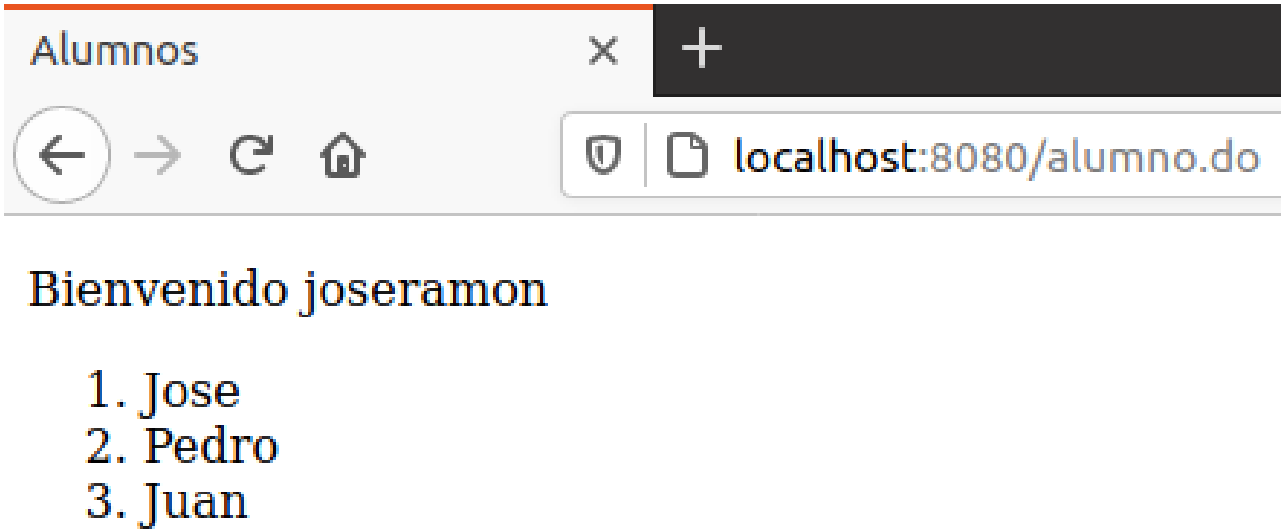
Listar datos:

... continuación 6º Utilizar JSTL para mejorar el aspecto de la lista de alumnos:

Modifica SOLO el fichero alumno.jsp para que se visualice de la siguiente manera :



Ahora que ya tenemos el listado, queremos que en la página de alumnos (alumnos.jsp) aparezca un mensaje saludando al usuario como en la imagen:



¿Como podemos hacerlo? ¿Que fichero/s hay que modificar?

Atributos vs Sesión:

· La primera idea es ***realizar los siguientes cambios:***

→ Modificar LoginServlet para que cuando redirige a AlumnoServlet se le pase como atributo el nombre del usuario

→ Modificar AlumnoServlet para que cuando abra alumno.jsp pasarle el atributo

→ Modificar alumno.jsp para que muestre el saludo al usuario.

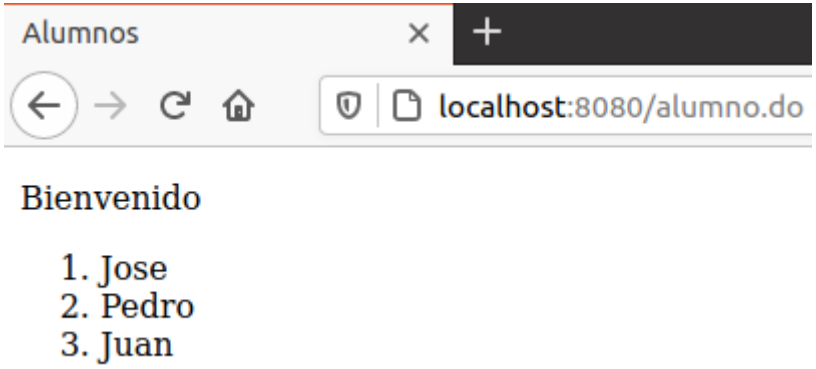
Para continuar la explicación es necesario primero realizar los cambios anteriores.

Atributos vs Sesión:

Sin embargo, una vez hechos los cambios el resultado es el siguiente:

alumno.jsp

```
1  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2  <%@ page language="java" contentType="text/html; charset=UTF-8"
3      pageEncoding="UTF-8"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>Alumnos</title>
9  </head>
10 <body>
11 <p>Bienvenido ${nombre}</p>
12 <ol>
13     <c:forEach items="${alumnos}" var="alumno">
14         <li>${alumno.getNombre()}</li>
15     </c:forEach>
16 </ol>
17 </body>
18 </html>
```

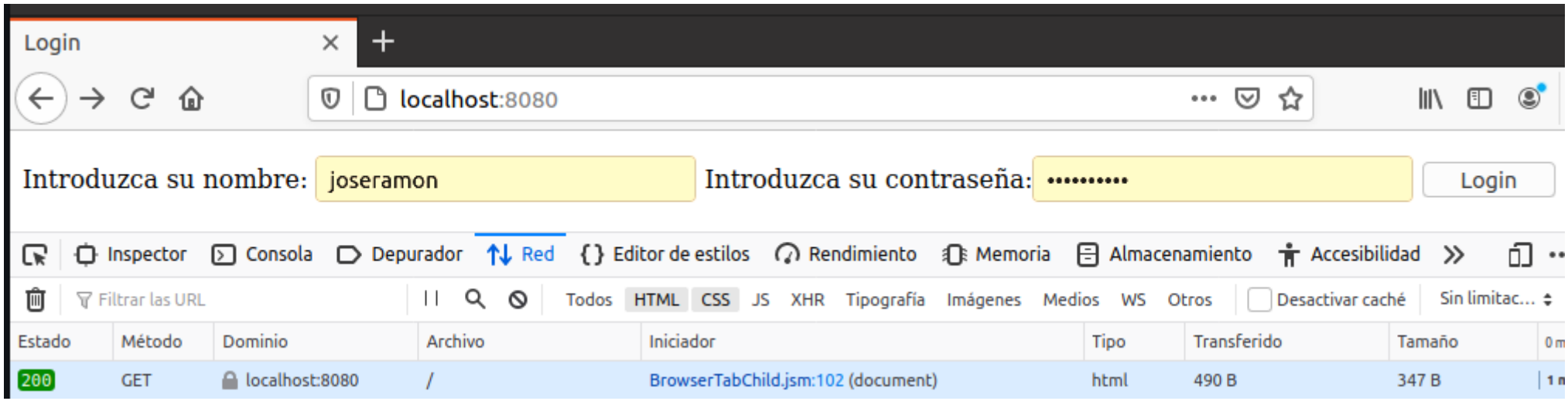


¿Por que la página JSP no tiene el valor del nombre?

... continuación Atributos vs Sesión:

Para entender porque no ha funcionado abrimos el navegador y vamos a hacer una traza de las peticiones recibidas y enviadas al servidor:

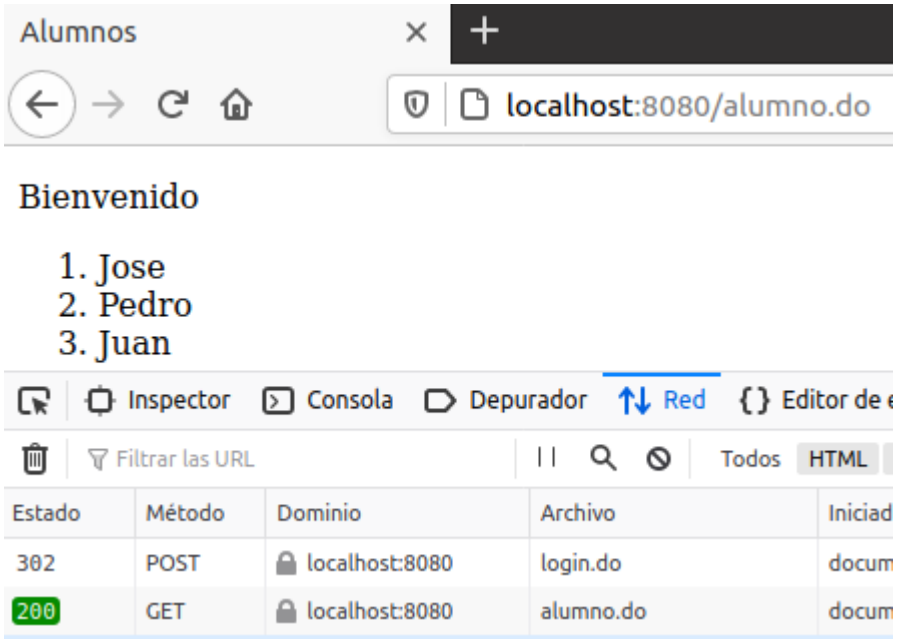
Cuando nos pide el usuario y el login se realiza una petición GET:



... continuación Atributos vs Sesión:

Y cuando pulsamos en el botón “Login” estamos haciendo un POST. En dicha petición POST (doPost en LoginServlet) configuramos el atributo nombre.

Sin embargo en el doPost no reenviamos a alumno.jsp directamente, sino que redirigimos la salida al Servlet AlumnoServlet(302= redirección), con lo cual se genera otra petición de GET sobre la url alumno.do atendida por AlumnoServlet. El atributo “nombre” lo habíamos configurado para la primera petición (login.do), no para la segunda (alumno.do).



¿Como podemos pasar información de un Servlet a otro?

... continuación **Atributos vs Sesión:**

Si un Servlet delega en una página JSP para presentar la página HTML de respuesta se esta realizando en la misma petición, por ello los atributos pasados a la petición (request) se pueden ver desde las páginas JSP.

Sin embargo ***si queremos comunicarnos entre 2 Servlets deberemos de utilizar otra técnica*** para hacerlo porque cuando un Servlet se redirige a otro se genera una petición distinta.

La solución pasa por ***almacenar dicha información en la sesión***. La sesión es algo que comienza cuando el usuario se logea y desaparece cuando hace logOut. Muchas veces no hace falta hacer logOut , porque existe un timeOut que permite cerrar la sesión automáticamente si se excede un tiempo (timeOut) sin hacer nada. El timeOut se utiliza por ejemplo en aplicaciones de Banca.

¿Como almacenamos la información en las sesiones en Java?

... continuación Atributos vs Sesión:

Para almacenar la información en la sesión debemos primero obtener la sesión (request.getSession()) y posteriormente añadirle el atributo (.setAttribute(,)).

Añadamos en LoginServlet el nombre en la sesión y comprobemos que ya funciona:

LoginServlet.java

```
1  package org.profesor.joseramon.primer_jee.login;
2
3  import java.io.IOException;
13
15  * Browser sends Http Request to Web Server
22
23 /*Java Platform, Enterprise Edition (Java EE) JEE8
35 @WebServlet(urlPatterns = "/login.do")
36 public class LoginServlet extends HttpServlet {
37     LoginService loginServicio = new LoginService();
38     AlumnoService alumnoServicio= new AlumnoService();
39
40     @Override
41     protected void doGet(HttpServletRequest request,
42             HttpServletResponse response) throws IOException, ServletException {
43         request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
44     }
45     @Override
46     protected void doPost(HttpServletRequest request,
47             HttpServletResponse response) throws IOException, ServletException {
48
49         String nombre =request.getParameter("nombre");
50         String password =request.getParameter("password");
51
52         if (loginServicio.usuarioValido(nombre, password)) {
53             request.getSession().setAttribute("nombre", nombre);
54             //validación correcta: Redirigir al Servlet de alumno
55             response.sendRedirect("alumno.do");
56         } else {
57             //validación incorrecta
58             request.setAttribute("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
59             request.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(request, response);
60         }
61     }
62 }
```

... continuación Atributos vs Sesión:

Si la información está en la sesión realmente no hace falta pasar a la página JSP el nombre como atributo. Comentemos las 2 lineas en AlumnoServlet donde se pasa el atributo “nombre” y comprobemos si se sigue viendo el nombre:

AlumnoServlet.java

```
1  package org.profesor.joseramon.primer_jee.alumno;
2
3  import java.io.IOException;
12
13 @WebServlet(urlPatterns = "/alumno.do")
14 public class AlumnoServlet extends HttpServlet {
15     AlumnoService alumnoServicio= new AlumnoService();
16
17     @Override
18     protected void doGet(HttpServletRequest request,
19             HttpServletResponse response) throws IOException, ServletException {
20         /*String nombre =request.getParameter("nombre");
21         request.setAttribute("nombre",nombre);*/
22         request.setAttribute("alumnos",alumnoServicio.listaAlumnos());
23         request.getRequestDispatcher("/WEB-INF/views/alumno.jsp").forward(request, response);
24     }
25 }
```

Ahora la pregunta de un posible alumno aventajado seria:

¿Porque no pasamos toda la información en la sesión (lista de alumnos incluida)?

... continuación Atributos vs Sesión:

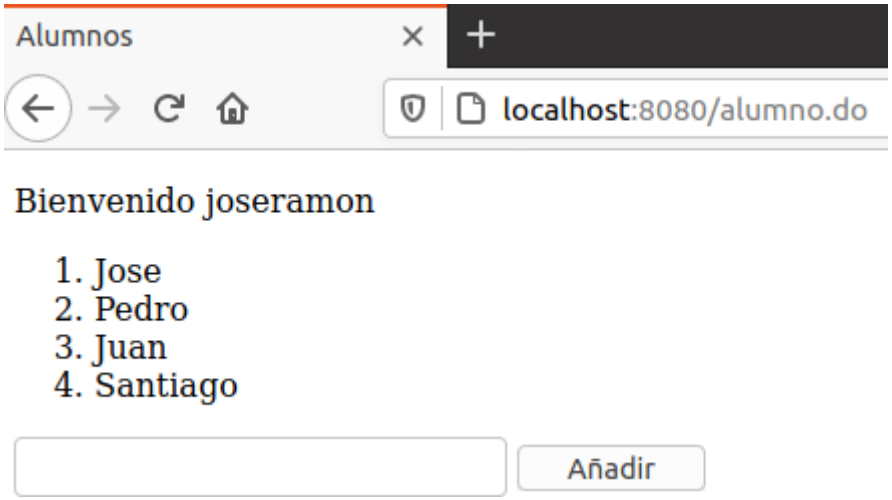
El primer motivo es que pasar toda la información e la sesión es como hacer todas las variables globales, o sea, una muy mala práctica.

El segundo motivo es que la información de ***la sesión de un usuario la guarda el servidor en memoria***, y por lo tanto debemos guardar en la sesión el menor número de datos posible para evitar que cuando el servidor tenga muchos usuarios el sistema se colapse.

Añadir datos a la lista:

Hasta ahora solo hemos mostrado el contenido de una lista, pero nos falta averiguar como podemos añadir y borrar información de esta lista.

¿Como podemos añadir alumnos nuevos a nuestra lista?



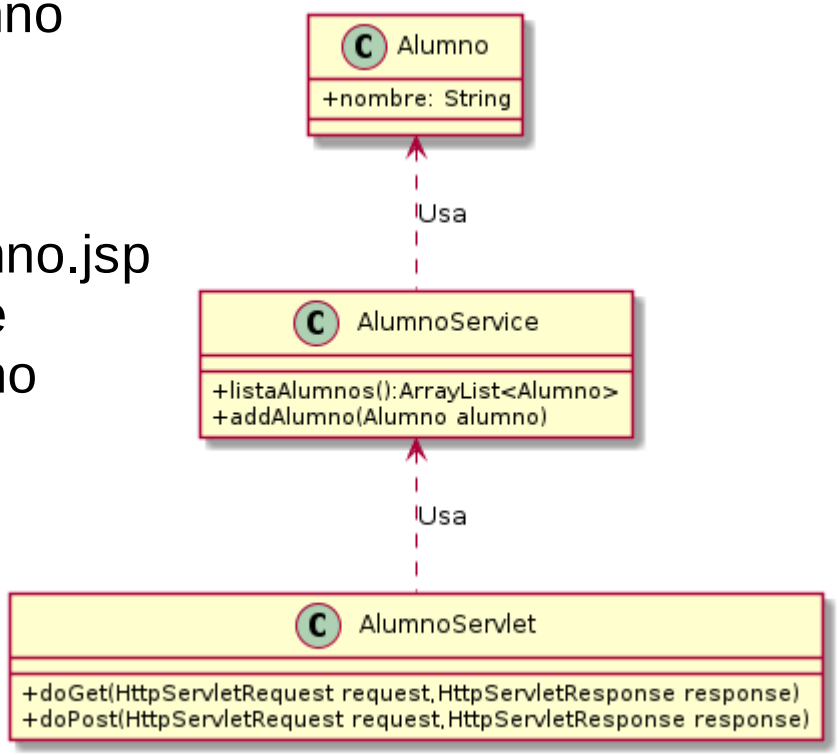
... continuación **Añadir datos a la lista:**

Para poder añadir alumnos a nuestra lista realizaremos los siguientes pasos:

1º AlumnoService.java: Crear un método nuevo "addAlumno(Alumno alumno)" que añada el alumno introducido como parámetro en la lista.

2º AlumnoServlet.java: Crear un método nuevo "doPost()" que antes de pasarle el testigo a alumno.jsp (podemos copiar el contenido de doGet()) realice la inserción (utilizando AlumnoService) del alumno recibido como parámetro .

3º alumnos.jsp: Añadir al final un formulario que pida el nombre del alumno nuevo y realice una llamada a "alumno.do".



Realiza los pasos anteriores y comprueba que puedes añadir nuevos alumnos.

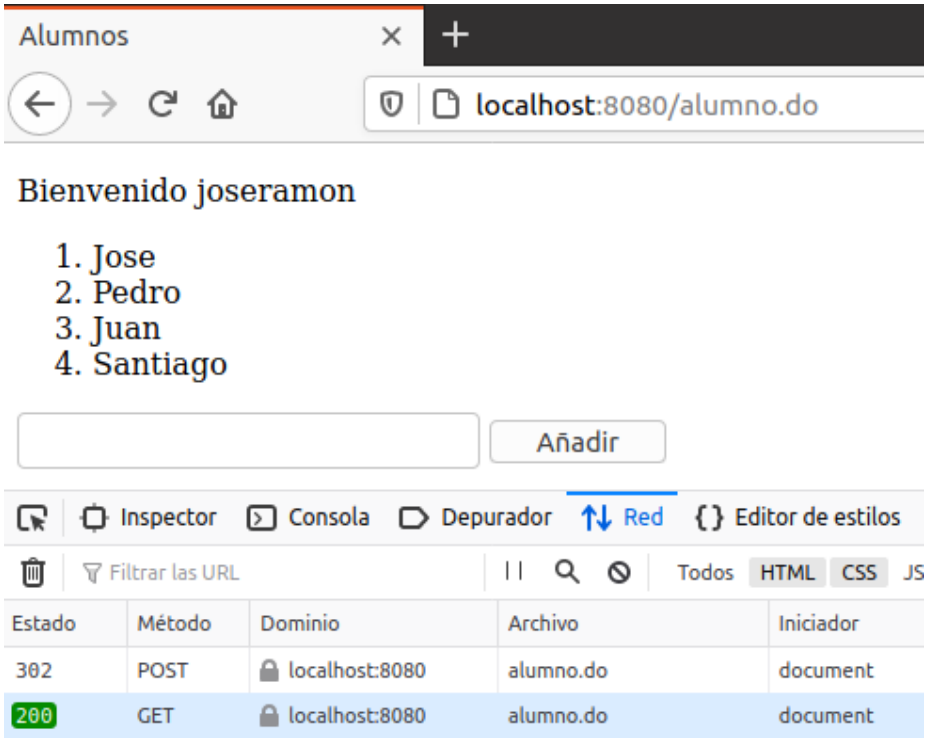
... continuación Añadir datos a la lista:

Tras haber añadido un nuevo alumno dale a “Recargar” la página:

¿Que problema tenemos?¿Como podemos evitarlo?

... continuación Añadir datos a la lista:

Para ***evitar inserciones duplicadas*** deberemos insertar el alumno y en vez de pasar el testigo a alumno.jsp para que muestre el listado de alumnos, deberemos redirigir de nuevo a "alumno.do" para que sea el método doGet el que liste los alumnos.



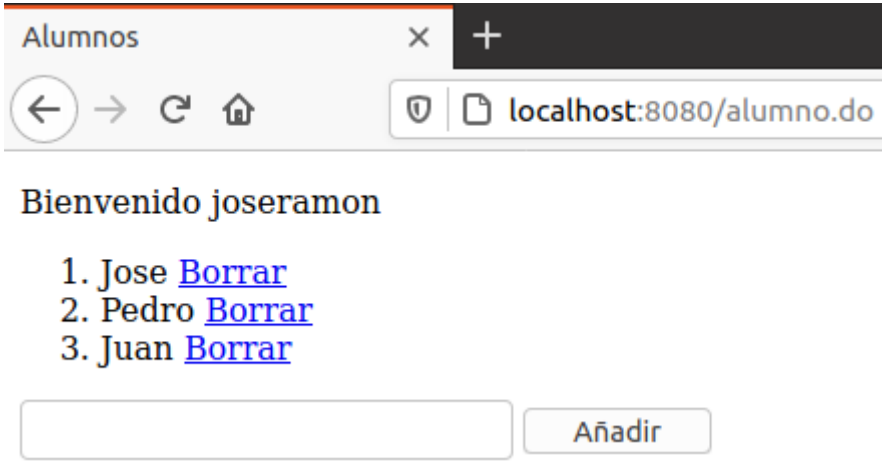
Realiza el cambio y comprueba que ahora ya no se insertan alumnos duplicados si recargamos la página.

TRUCO: Como norma cuando queremos realizar una acción y volver a mostrar los datos una vez realizada la acción llamaremos a la página que muestra los datos con response.sendRedirect() en vez de cargar los datos con request.setAttribute() y ejecutar request.getRequestDispatcher().

Borrar datos de la lista:

Hasta ahora ya sabemos listar y añadir alumnos a nuestra lista.

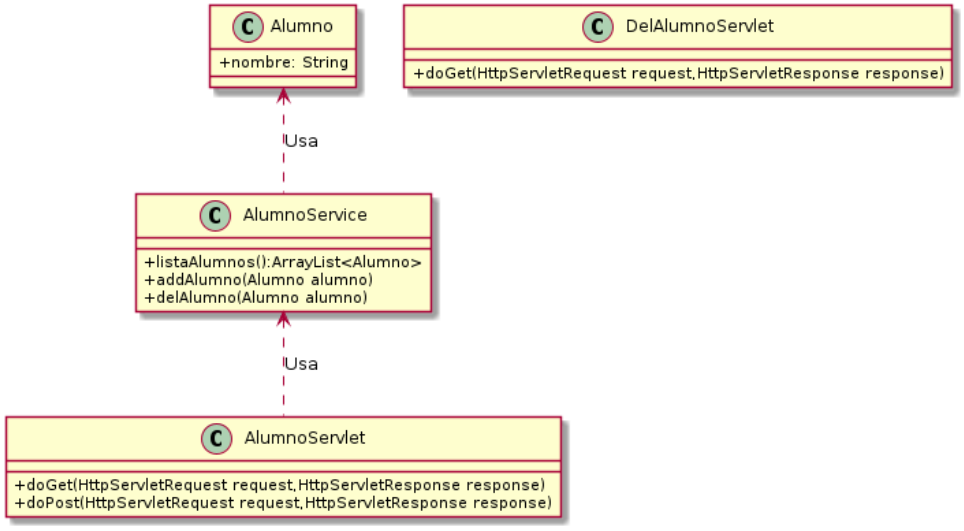
¿Como podemos borrar alumnos de nuestra lista?



... continuación **Borrar datos de la lista:**

Para poder borrar alumnos en nuestra lista realizaremos los siguientes pasos:

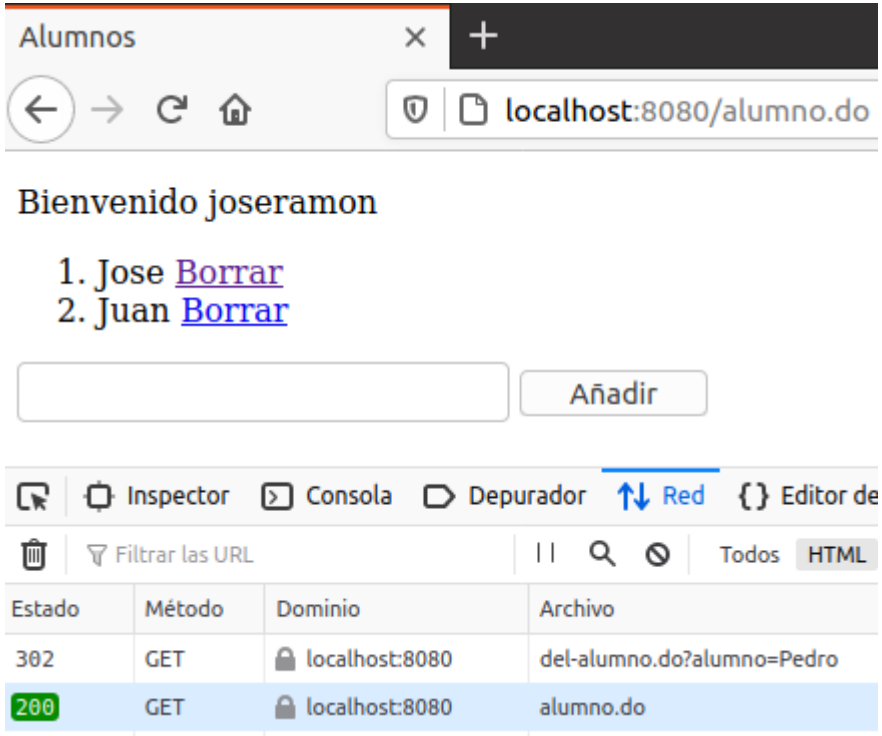
1º AlumnoService.java: Crear un método nuevo "delAlumno(Alumno alumno)" que borre el alumno introducido como parámetro en la lista. Recuerda que si no tienes el método equals() en Alumno tendrás un problema para borrar ...
2º alumnos.jsp: Añadir junto a cada alumno un enlace a la url "del-alumno.do". Para saber que alumno se borrará se pasará el nombre del alumno como parámetro en la url.
3º Crear DelAlumnoServlet.java: Para crear este servlet podemos copiar el contenido de AlumnoServlet y quitarle el doPost(). La url de este Servlet será "del-alumno.do" . En el método doGet se utilizara el servicio AlumnoService para borrar el alumno y posteriormente se redirigirá otra vez a "alumno.do".



Realiza los pasos anteriores y comprueba que puedes borrar nuevos alumnos.

... continuación **Borrar datos de la lista:**

Si lo hemos hecho bien al borrar a Pedro veremos que se ha generado una petición GET hacia “del-alumno.do?alumno=Pedro” que ha sido redirigida (estado 302) y posteriormente se ha realizado una petición GET hacia “alumno.do”.



EJERCICIO:

Sigue todos los pasos de los PDF y sube la aplicación final al moodle.

Para ello:

1º Haz un "Run As \Maven Clean" para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).

2º Comprime la carpeta de tu aplicación y ponle como nombre al fichero comprimido UD1_practica5_nombreAlumno.tar.gz donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviesemos Windows, podemos comprimir en ZIP.