UD 2: Modelo Vista Controlador
3.- Estructura de paquetes en Spring MVC

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

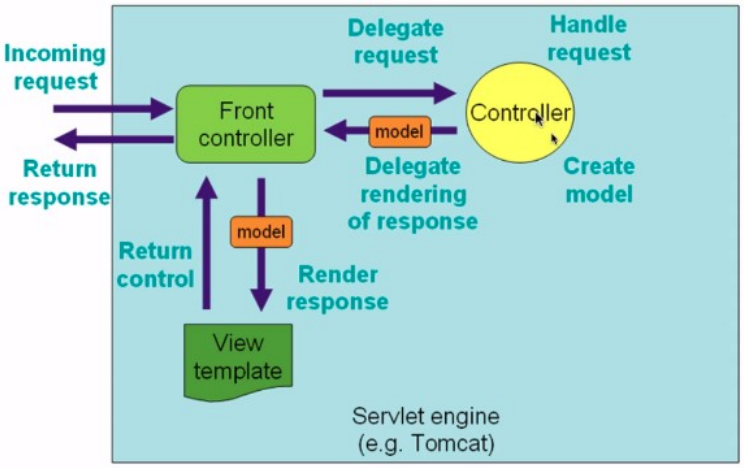
- *Entender el **patrón de diseño modificado "Front Controller"** de **Spring MVC.***
- ***Refactorizar** nuestra aplicación creando una estructura de paquetes adecuada para listar datos.*
- *Aprender a utilizar **sesiones en Spring MVC**.*
- *Aprender a realizar **redirecciones en Spring MVC**.*
- *Aprender que son los **beans** y como utilizarlos en nuestra aplicación al añadir o borrar datos para simplificar la webapp.*

UD 2: Modelo Vista Controlador
3.- Estructura de paquetes en Spring MVC

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Volvamos a recapitular lo que hemos visto hasta ahora:

Hemos configurado nuestro ***DispatcherServlet*** en ***web.xml*** (que actua de Front controller recibiendo todas las peticiones) y hemos configurado el contexto en ***alumno-servlet.xml***.



Hemos configurado un ***controlador básico (LoginController)*** con las anotaciones ***@Controller*** ***y @RequestMapping*** . Hemos añadido a @RequestMapping el RequestMethod para ser capaces de atender peticiones GET y POST.

Hemos configurado el ***ViewResolver en alumno-servlet.xml*** para que cuando hagamos ***return 'login'*** , la aplicación sepa que la vista que tiene que mostrar es ***WEB-INF/views/login.jsp***.

Hemos añadido un ***parámetro nuevo de tipo ModelMap*** a nuestros métodos del controlador para poder añadir los datos (antiguos atributos en JEE8) que necesita la vista (los JSP).

Hemos configurado los ***servicios con la anotación @Service*** para poder ***enlazar esos servicios en el controlador con @Autowired*** sin que haga falta crear una instancia nueva.

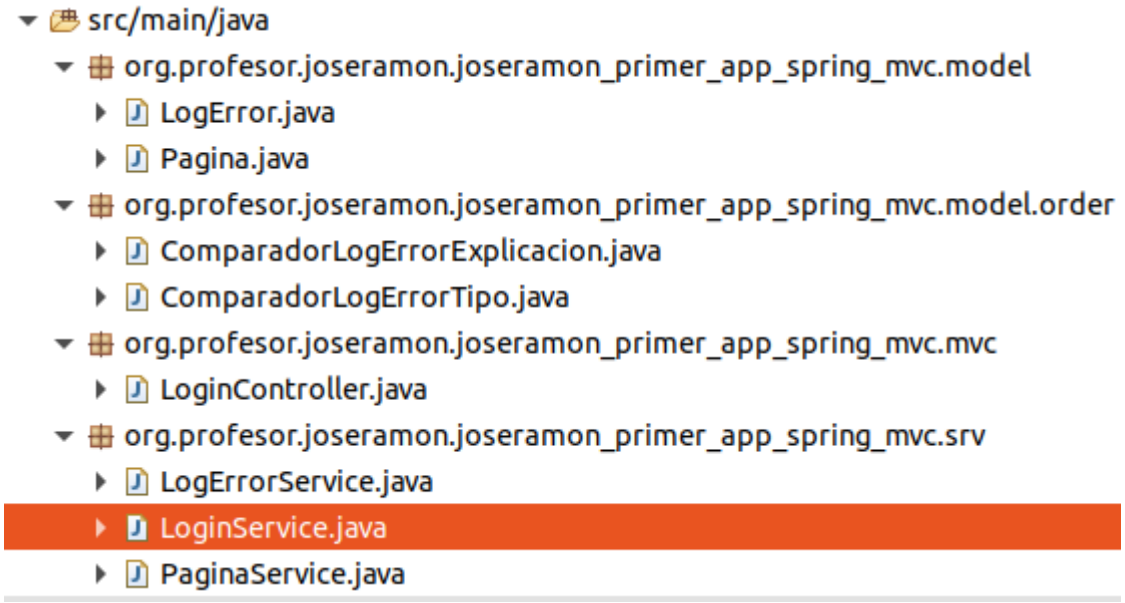
UD 2: Modelo Vista Controlador
3.- Estructura de paquetes en Spring MVC

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Continuemos poniendo un poco de orden en los paquetes de java ahora que ya conocemos el modelo MVC completo:

En vez de tenerlo todo mezclado en un mismo paquete o crear un subpaquete 'login', otro 'alumno',etc… vamos a repartir los ficheros java en paquetes según su funcionalidad. Realiza los siguientes cambios para que queden SOLO las clases:

- ***mvc:*** Contiene las clases java que actúan de controladores
- ***srv:*** Contiene las clases java que proporcionan servicios
- ***model:*** Contiene las clases java que contienen datos.
- ***model.order:*** Contiene las clases java que permiten ordenar los datos por diferentes criterios.



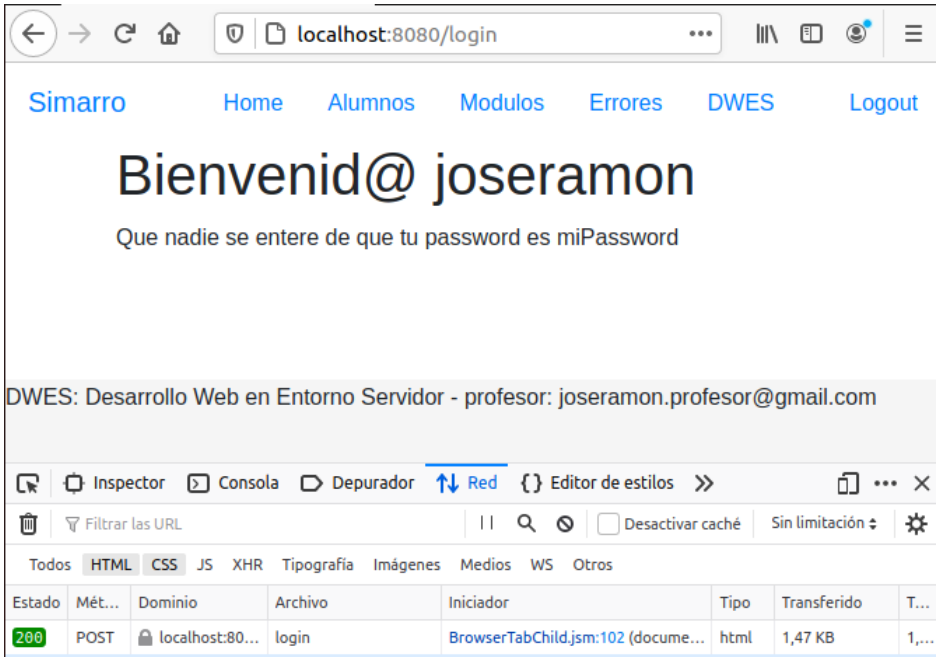
Si no tienes las clases relacionadas con LogError (servicio y comparadores) y Pagina no pasa nada, pero más adelante te harán falta. Si los tienes tendrás que crear la anotación @Service en las clases LogErrorService y PaginaService para que las puedan utilizar más adelante los controladores.

Comprueba que todo sigue funcionando y puedes hacer login y acceder a la página de bienvenida.

Ahora que ya no tenemos ni rastro de los servlets de JEE8 podemos realizar los siguientes cambios:

· Quitar el prefijo “spring-mvc” para que cuando queramos llamar a un controlador no tengamos que poner en la ruta “/spring-mvc/rutaControlador”.

· Hacer que la página inicial que se carga al poner la URL “localhost:8080” sea login.



Si en vez de hacer “maven build” utilizamos “Run on Server” la url a poner deberá ser “localhost:8080:/nombreAlumno_primer_app_spring_mvc/”
Ayuda:
Casi todos los cambios se realizan en web.xml.
Adicionalmente en LoginController habrá que crear un método nuevo que atienda la url “/” y haga lo mismo que hace “login”.

UD 2: Modelo Vista Controlador
3.- Estructura de paquetes en Spring MVC

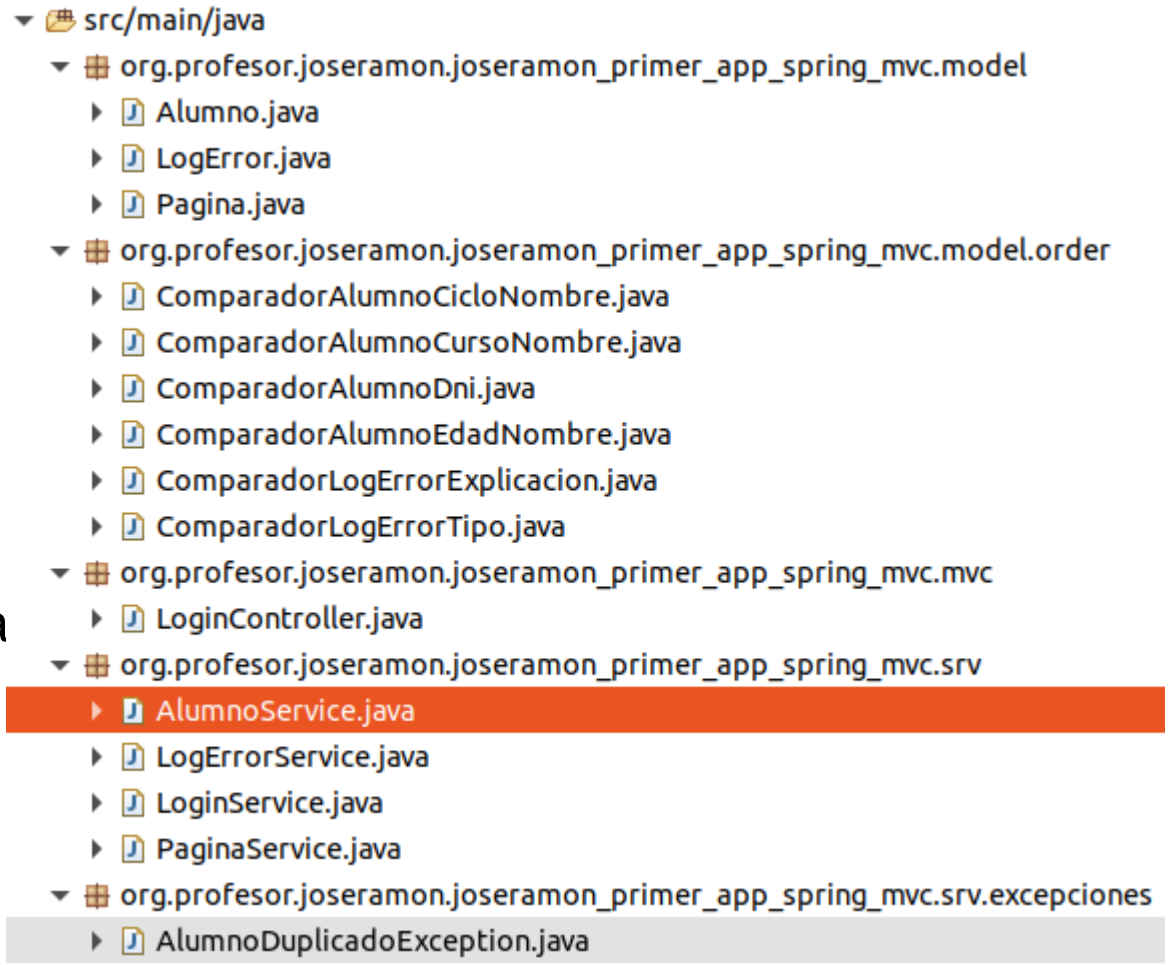
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Continuemos configurando el listado de alumnos con Spring MVC:

Para listar los alumnos deberemos realizar los siguientes cambios:

Ayuda: Podemos copiar los ficheros de la UD1

· Crear la clase “Alumno.java” en el paquete “model”.
· Crear las clases que ordenan “Alumno” por varios criterios en el paquete “model.order”.
· Crear la clase “AlumnoDuplicadoException.java” en “srv.excepciones”.
· Crear la clase “AlumnoService.java” en el paquete “srv”.
· Modificar “AlumnoService.java” para añadir la anotación “@Service”.
· Crear el fichero “list-alumno.jsp” en WEB-INF/list-alumno.jsp



Como se puede empezar a ver es importante en proyectos grandes establecer criterios claros a la hora de nombrar ficheros en los proyectos.

Ahora nos queda crear un nuevo fichero “AlumnoController.java” desde cero en el paquete mvc porque es el que más cambia respecto a la forma de trabajar en JEE8. Para poder listar los alumnos realizaremos los siguientes cambios:

· Añadirle a la clase “AlumnoController” la anotación @Controller
· Añadirle el servicio AlumnoService y enlazarlo con el servicio con @Autowired
· Añadir el servicio PaginaService y un atributo “pagina” inicializado a new Pagina(“Alumnos”,”list-alumno”);
· Crear un método nuevo “listarAlumnos” que atiende “list-alumno” en modo GET y que incorpora al modelo el listado de alumnos y la página recuperada del servicio. Acordaros que antes de recuperar la página del servicio tendremos que configurarla. La página servirá para que la sección de “Alumnos” se active con la notación bootstrap. ***De momento todavía no añadiremos la lógica para ordenar los resultados***. Después el método enviará la vista al fichero “list-alumno.jsp”.

Con respecto a las páginas jsp realizaremos los siguientes cambios:
· Crear list-alumno.jsp.
· Modificar el fragmento menuSuperior.jspf para que el enlace a los 2 controladores configurados (“Alumnos” y “Home” (login)) funcionen.

UD 2: Modelo Vista Controlador
4.- Sesiones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

El la **UD 1** cuando queríamos leer una información aunque se hubiera generado en una petición (request) diferente utilizamos **sesiones** con **JEE8:**

· **request.getSession().setAttribute(“nombreVar”,”valorVar”)** para crear un valor nuevo en la sesión

· **request.getSession().getAttribute(“nombreVar”)** para leer el valor de ese campo en la sesión.

LoginServlet.java

```
64          if (loginServicio.usuarioValido(nombre, password)) {
65              request.getSession().setAttribute("nombre", nombre);
66              //validación correcta: Redirigir al Servlet de alumno
67              response.sendRedirect("list-alumno.do");
```

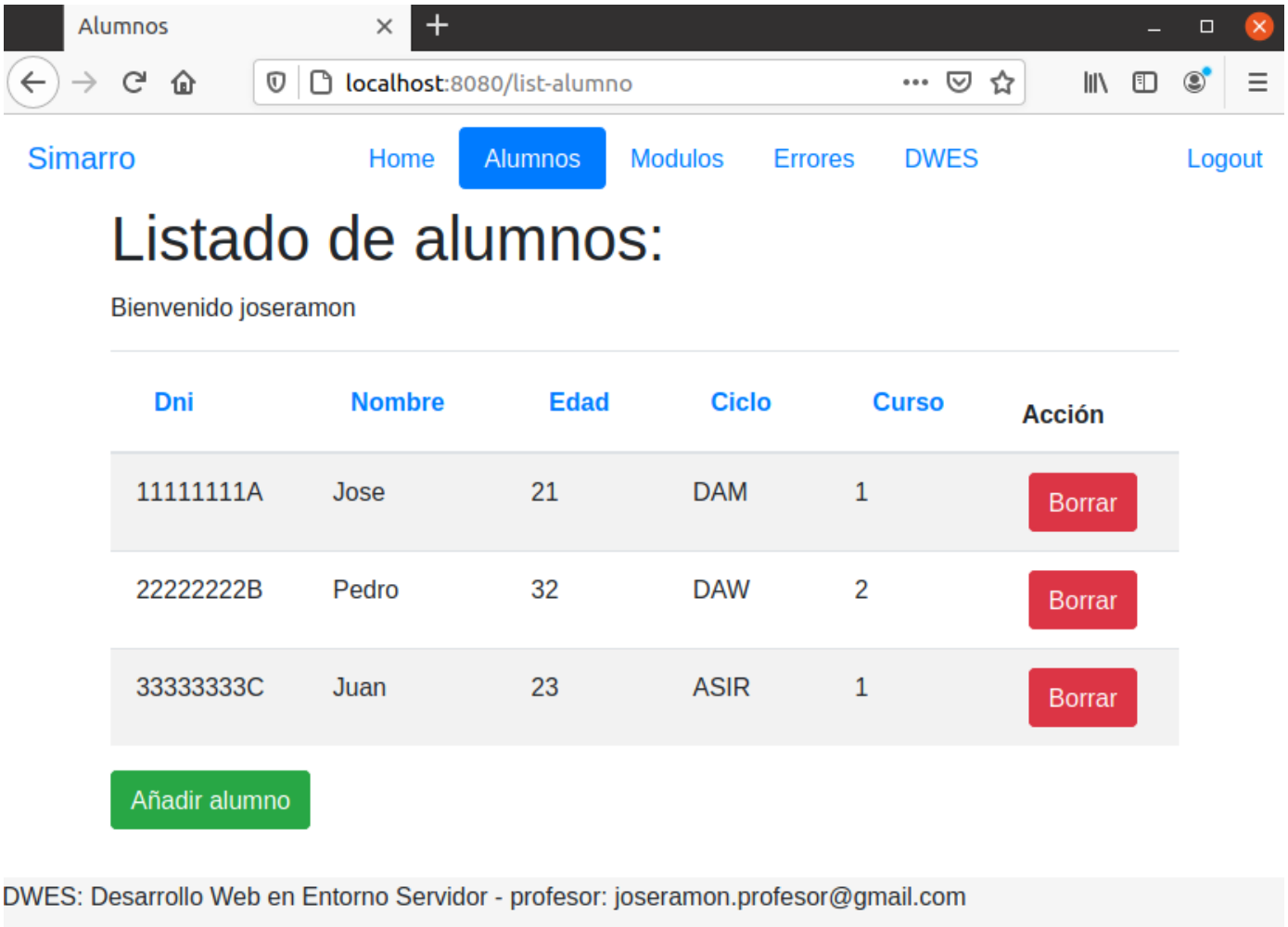
¿Como lo hacemos con Spring?

UD 2: Modelo Vista Controlador
4.- Sesiones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Vamos a ver un ***ejemplo práctico para entender como utilizar sesiones para pasar datos entre peticiones***:

Queremos que el listado de alumnos dé la bienvenida al usuario:



Para conseguirlo hay que realizar los siguientes pasos :

1º Añadir la anotación ***@SessionAttributes()*** (linea 13) debajo de @Controller:

LoginController.java

```
12 @Controller
13 @SessionAttributes("nombre")
14 public class LoginController {
15     @Autowired
16     LoginService loginService;
17
18     @RequestMapping(value="/",method = RequestMethod.GET)
19     public String principal() {
20         return "login";
21     }
22
23     @RequestMapping(value="login",method = RequestMethod.GET)
24     public String mostrarLogin() {
25         return "login";
26     }
27
28     @RequestMapping(value="login",method = RequestMethod.POST)
29     public String procesaLogin(@RequestParam String nombre,
30             @RequestParam String password,
31             ModelMap model) {
32         if (!loginService.usuarioValido(nombre, password)) {
33             //usuario inválido, volver a intentar logearse
34             model.put("errores", "Usuario '" + nombre + "' o contraseña incorrecta");
35             return "login";
36         }
37         // Si llega aquí es porque el usuario era valido
38         model.put("nombre", nombre);
39         model.put("password", password);
40         return "bienvenida";
41     }
42 }
```

En el momento que se añade la anotación @SessionAttributes() ***si se añade al modelo (linea 38) un atributo nuevo con el mismo valor (“nombre”) automáticamente se guarda en la sesión***.

2º Para leer desde el controlador del listado la variable de sesión tan solo hay que añadir la anotación ***@SessionAttributes()*** debajo de @Controller y en los modelos que se utilicen se pasará dicha información automáticamente sin hacer nada:

AlumnoController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.mvc;
2
3  import org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.Pagina;
12
13 @Controller
14 @SessionAttributes("nombre")
15 public class AlumnoController {
16         Pagina pagina= new Pagina("Alumnos","list-alumno");
17         @Autowired
18         AlumnoService alumnoService;
19         @Autowired
20         PaginaService paginaServicio;
21
22
23         @RequestMapping(value="list-alumno",method = RequestMethod.GET)
24         public String listarAlumnos(ModelMap model) {
25             paginaServicio.setPagina(pagina);
26             model.addAttribute("alumnos",alumnoService.listaAlumnos());
27             model.addAttribute("pagina",paginaServicio.getPagina());
28             return "list-alumno";
29         }
30
31 }
```

NOTA: En ningún momento hemos añadido “nombre” al modelo, pero **Spring lo añadirá automáticamente** al tener la anotación @SessionAttributes()

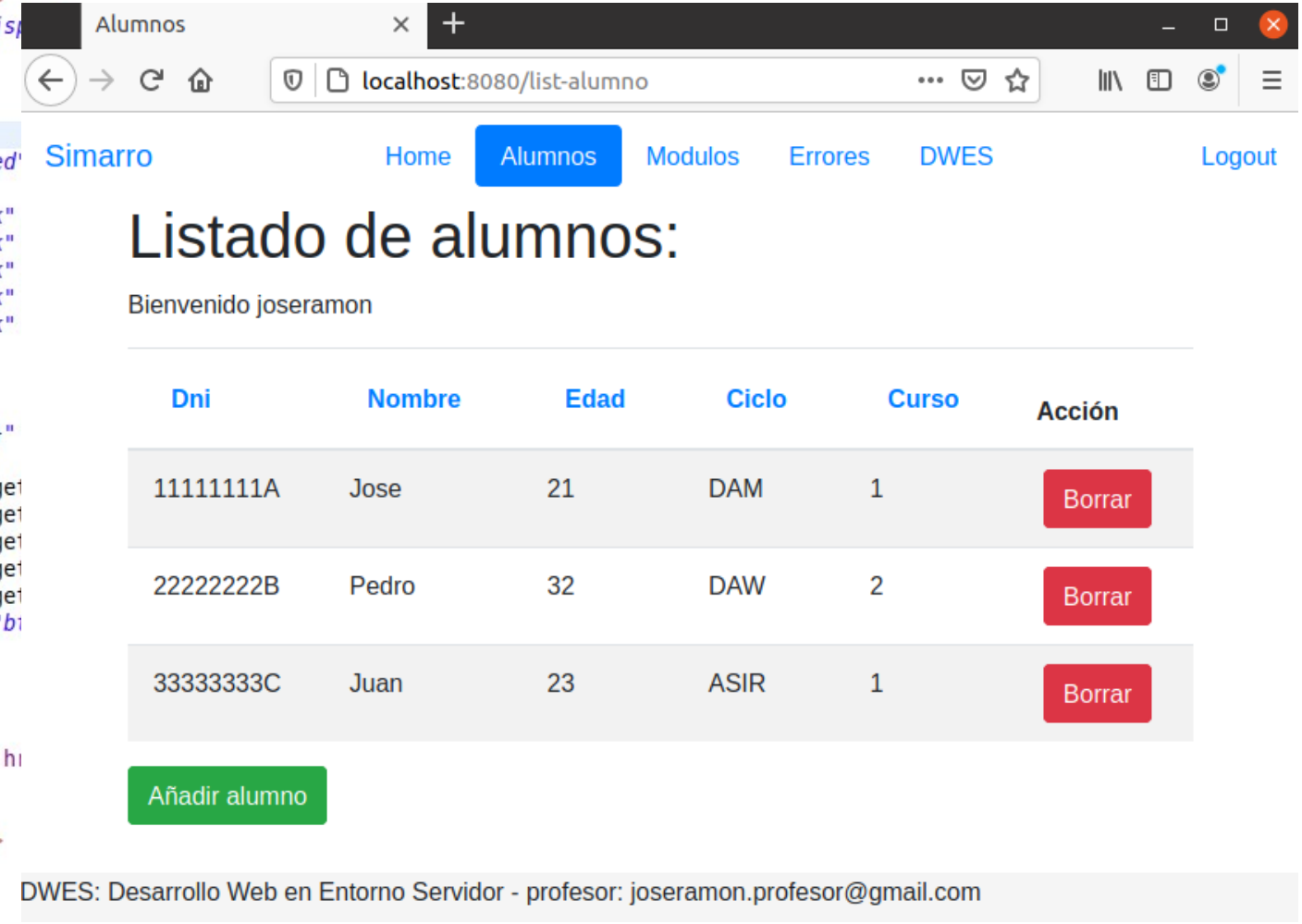
UD 2: Modelo Vista Controlador
4.- Sesiones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

3º Para que se muestre el nombre en el listado solo queda añadir el valor en el jsp:

list-alumno.jsp

```
1  <%@ include file="../jspf/header.jspf"%>
2  <%@ include file="../jspf/menuSuperior.jsp
3
4      <div class="container">
5          <h1> Listado de alumnos:</h1>
6          <p>Bienvenido ${nombre}</p>
7          <table class="table table-striped"
8              <thead>
9                  <th> <a  class="nav-link"
10                 <th> <a  class="nav-link"
11                 <th> <a  class="nav-link"
12                 <th> <a  class="nav-link"
13                 <th> <a  class="nav-link"
14                 <th> Acción</th>
15             </thead>
16             <tbody>
17             <c:forEach items="${alumnos}"
18                 <tr>
19                     <td> ${alumno.get
20                     <td> ${alumno.get
21                     <td> ${alumno.get
22                     <td> ${alumno.get
23                     <td> ${alumno.get
24                     <td> <a class="bt
25                 </tr>
26             </c:forEach>
27             </tbody>
28         </table>
29         <p> <a  class="btn btn-success" hr
30     </div>
31
32 <%@ include file="../jspf/footer.jspf"%>
```



UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Ahora que ya tenemos la funcionalidad de listar alumnos vamos a añadir la funcionalidad de añadir alumnos. Para ello hay que realizar los siguientes cambios:

1º Comprobamos que tenemos en la vista (list-alumno.jsp) un enlace a “add-alumno”: OJO: add-alumno , no add-alumno.do

list-alumno.jsp

```
1  <%@ include file="../jspf/header.jspf"%>
2  <%@ include file="../jspf/menuSuperior.jspf"%>
3
4      <div class="container">
5          <h1> Listado de alumnos:</h1>
6          <p>Bienvenido ${nombre}</p>
7          <table class="table table-striped">
8              <thead>
9                  <th> <a  class="nav-link" href="list-alumno.do?orden=dni">Dni</a></th>
10                 <th> <a  class="nav-link" href="list-alumno.do?orden=nombre">Nombre</a></th
11                 <th> <a  class="nav-link" href="list-alumno.do?orden=edadNombre">Edad</a></
12                 <th> <a  class="nav-link" href="list-alumno.do?orden=cicloNombre">Ciclo</a>
13                 <th> <a  class="nav-link" href="list-alumno.do?orden=cursoNombre">Curso</a>
14                 <th> Acción</th>
15             </thead>
16             <tbody>
17             <c:forEach items="${alumnos}" var="alumno">
18                 <tr>
19                     <td> ${alumno.getDni()} </td>
20                     <td> ${alumno.getNombre()} </td>
21                     <td> ${alumno.getEdad()} </td>
22                     <td> ${alumno.getCiclo()} </td>
23                     <td> ${alumno.getCurso()} </td>
24                     <td> <a class="btn btn-danger" href="del-alumno.do?dni=${alumno.ge
25                 </tr>
26             </c:forEach>
27             </tbody>
28         </table>
29         <p> <a  class="btn btn-success" href="add-alumno">Añadir alumno</a></p>
30     </div>
31
32 <%@ include file="../jspf/footer.jspf"%>
33
34
```

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

2º Añadimos un método para atender “add-alumno” en AlumnoController.java:

AlumnoController.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.mvc;
2
3  import org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.Pagina;
12
13 @Controller
14 @SessionAttributes("nombre")
15 public class AlumnoController {
16         Pagina pagina= new Pagina("Alumnos","list-alumno");
17         @Autowired
18         AlumnoService alumnoService;
19         @Autowired
20         PaginaService paginaServicio;
21
22
23         @RequestMapping(value="list-alumno",method = RequestMethod.GET)
24         public String listarAlumnos(ModelMap model) {
25             paginaServicio.setPagina(pagina);
26             model.addAttribute("alumnos",alumnoService.listaAlumnos());
27             model.addAttribute("pagina",paginaServicio.getPagina());
28             return "list-alumno";
29         }
30
31         @RequestMapping(value="add-alumno",method = RequestMethod.GET)
32         public String mostrarAlumno(ModelMap model) {
33             paginaServicio.setPagina(pagina);
34             model.addAttribute("pagina",paginaServicio.getPagina());
35             return "add-alumno";
36         }
37 }
```

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

3º Añadimos la vista (add-alumno.jsp de la UD1) para mostrar el formulario de entrada de un nuevo alumno teniendo en cuenta que enviaremos los datos a add-alumno en modo POST:

add-alumno.jsp

```
1  <%@ include file="../jspf/header.jspf"%>
2  <%@ include file="../jspf/menuSuperior.jspf"%>
3
4      <div class="container">
5          <h1>Nuevo alumno:</h1>
6          <p>
7              <font color="red">${errores}</font>
8          </p>
9          <p> Introduzca los datos del nuevo alumno:</p>
10
11         <form action="add-alumno" method="post">
12              <div class="form-row">
13                     <div class="col">
14                         <label>Dni:</label>
15                         <input type="text" id="dni" name="dni" required
16                             class="form-control" minlength="9"/>
17                     </div>
18                     <div class="col">
19                         <label>Nombre:</label>
20                         <input type="text" id="nombre" name="nombre" required
21                         class="form-control" minlength="5"/>
22                     </div>
23             </div>
24             <div class="form-row">
25                     <div class="col">
26                         <label>Edad:</label>
27                         <input type="number" id="edad" name="edad" required
28                             class="form-control" min="18" max="100"/>
29                     </div>
```

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

4º Añadimos el nuevo método que atenderá la url “add-alumno” en modo POST para añadir un alumno y volver al listado de alumnos.

Para poder realizar esta tarea tenemos que preguntarnos como hacer los siguientes pasos en Spring:

¿Como podemos leer los datos del formulario si no tenemos request.getParameter()?
¿Como podemos redirigir al listado una vez añadido el alumno si ya no tenemos response.sendRedirect()?

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

· Si hemos estado atentos , leer los datos del formulario ya sabemos, de hecho lo hemos hecho para leer el nombre del formulario del login. Debemos añadir los parámetros de los campos del formulario en el método como parámetros del método e incorporarles la anotación @RequestParam

· Para redirigir la petición debemos utilizar **“redirect:urlARedireccionar”**

AddAlumnoServlet.java

```java
package org.profesor.joseramon.primer_jee.alumno;

import java.io.IOException;

@WebServlet(urlPatterns = "/add-alumno.do")
public class AddAlumnoServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    AlumnoService alumnoServicio= new AlumnoService();
    PaginaService paginaServicio= new PaginaService();
    static Pagina paginaAddAlumno=new Pagina("Alumnos","list-alumno.do");

    protected void doGet(HttpServletRequest request,

    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws IOException, ServletException {
        paginaServicio.setPagina(paginaAddAlumno);
        request.getSession().setAttribute("pagina", paginaServicio.getPagina());
        try {
            alumnoServicio.addAlumno(new Alumno(
                    request.getParameter("dni"),
                    request.getParameter("nombre"),
                    Integer.parseInt(request.getParameter("edad")),
                    request.getParameter("ciclo"),
                    Integer.parseInt(request.getParameter("curso"))));
                    /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
                    response.sendRedirect("list-alumno.do");
        } catch (NumberFormatException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (AlumnoDuplicadoException e) {
            request.setAttribute("errores",e.toString());
            request.getSession().setAttribute("pagina", paginaServicio.getPagina());
            request.getRequestDispatcher("/WEB-INF/views/add-alumno.jsp").forward(request, response);
        }
    }
}
```

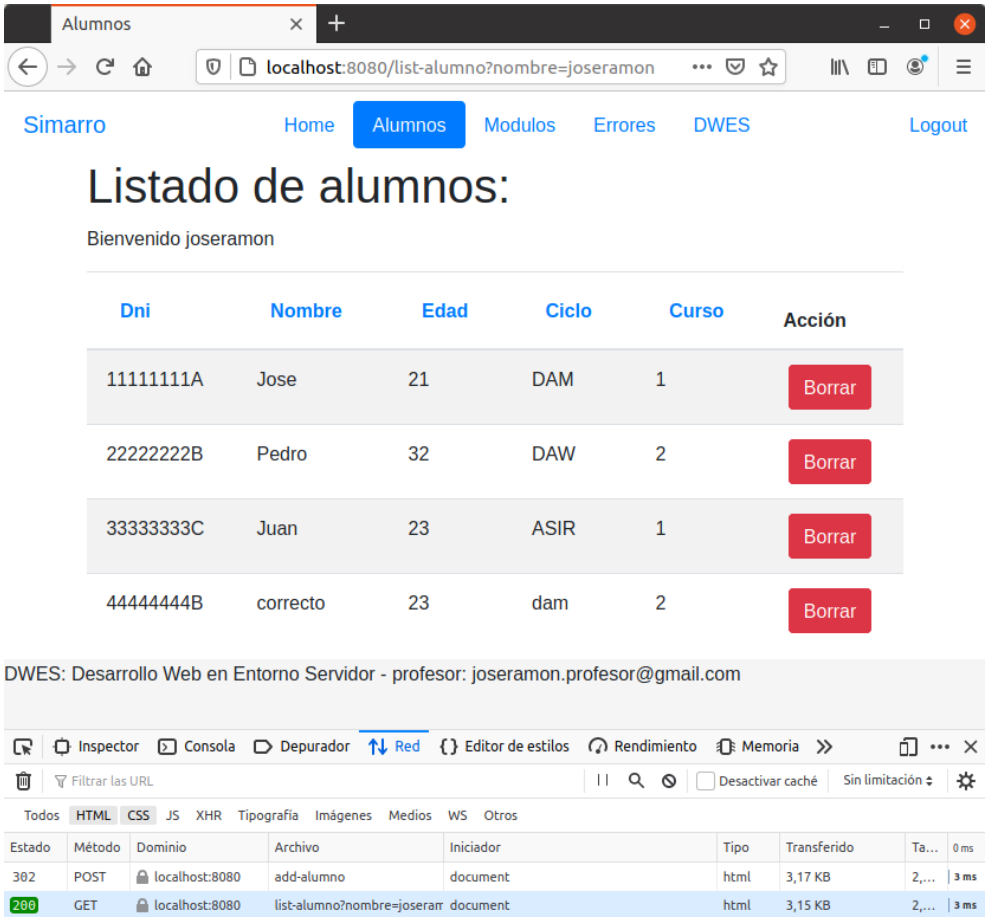
AlumnoController.java

```java
    @RequestMapping(value="add-alumno",method = RequestMethod.POST)
    public String addAlumno(@RequestParam String dni,
            @RequestParam String nombre,
            @RequestParam String edad,
            @RequestParam String ciclo,
            @RequestParam String curso,
            ModelMap model) {
        String errores="";
        paginaServicio.setPagina(pagina);
        model.addAttribute("pagina",paginaServicio.getPagina());
        try {
            alumnoService.addAlumno(new Alumno(
                    dni,nombre,Integer.parseInt(edad),
                    ciclo,Integer.parseInt(curso)));
            return "redirect:list-alumno";
        } catch (NumberFormatException e) {
            errores=e.getMessage();
        } catch (AlumnoDuplicadoException e) {
            errores=e.toString();
        }
        //Si llegamos aquí ha habido un error porque no ejecuta linea 54
        model.addAttribute("errores",errores);
        return "add-alumno";
    }
```

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Comprobemos si podemos añadir un alumno duplicado (nos debería de avisar del error) y comprobemos si podemos añadir un alumno nuevo correcto:



¿ Porque nos pasa como parámetro el nombre al redireccionar?

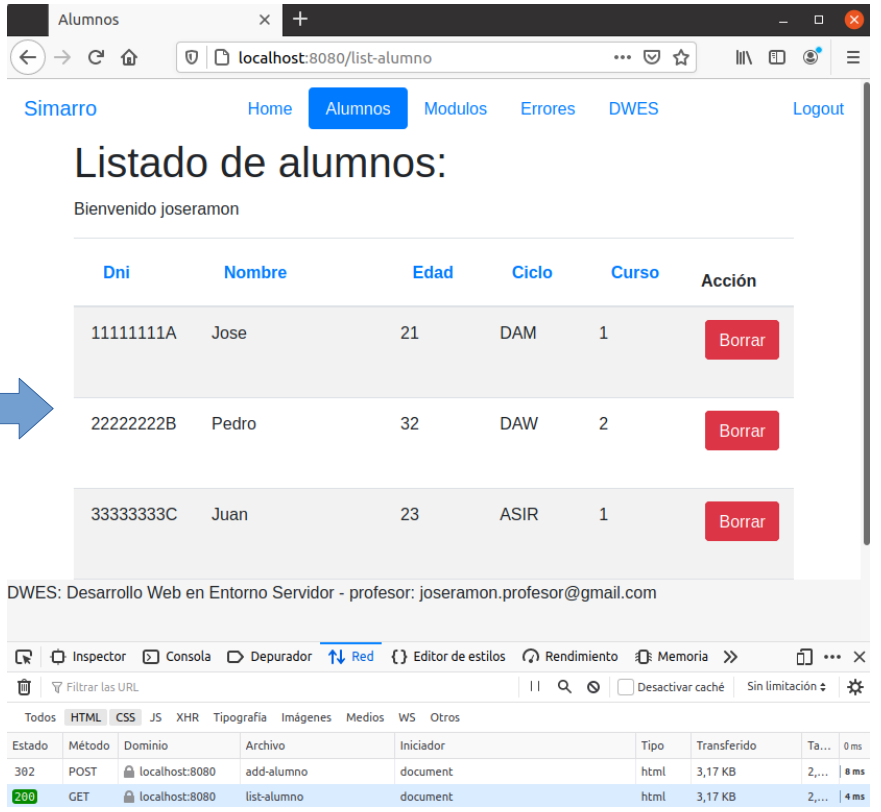
UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Cuando tenemos la notación @SessionAttributes y redireccionamos se pasan también como atributos (como parámetros en la url) los datos almacenados en la sesión. Para evitar pasar como parámetros los valores de la sesión debemos limpiar el modelo:

AlumnoController.java

```
40      @RequestMapping(value="add-alumno",method = RequestMethod.POST)
41      public String addAlumno(@RequestParam String dni,
42              @RequestParam String nombre,
43              @RequestParam String edad,
44              @RequestParam String ciclo,
45              @RequestParam String curso,
46              ModelMap model) {
47          String errores="";
48          paginaServicio.setPagina(pagina);
49          model.addAttribute("pagina",paginaServicio.getPagina());
50          try {
51              alumnoService.addAlumno(new Alumno(
52                      dni,nombre,Integer.parseInt(edad),
53                      ciclo,Integer.parseInt(curso)));
54              //Para evitar pasar parámetros inncesarios
55              model.clear();
56              /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
57              return "redirect:list-alumno";
58          } catch (NumberFormatException e) {
59              errores=e.getMessage();
60          } catch (AlumnoDuplicadoException e) {
61              errores=e.toString();
62          }
63          //Si llegamos aquí ha habido un error porque no ejecuta linea 54
64          model.addAttribute("errores",errores);
65          return "add-alumno";
66      }
```



UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

¿Como podemos borrar un alumno utilizando Spring MVC?

Realmente es muy similar a como lo haciamos en JEE8 y **ya sabemos hacerlo**.
Realiza los siguientes pasos:

· Asegurate que tenemos un enlace en el JSP del listado en cada linea que apunta a del-alumno con el parámetro del dni correspondiente.

· Crea en el controlador un nuevo método delAlumno que sea capaz de atender dicha petición.

· En el método anterior deberemos de borrar el alumno correspondiente y reenviar al listado nuevamente. Asegurate de limpiar el modelo antes de redireccionar…

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Antes de continuar avanzando es conveniente aprender un poco de “cultura Java” y saber que es y cual es la diferencia entre un POJO , un EJB y un JavaBean (más conocido como Bean).

¿Que es un POJO, un EJB y un Bean?

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

- **POJO** es un acronimo de “Plain Object Java Object”. Si lo traducimos literalmente “Objeto Java Plano Antiguo”. En la práctica un POJO es una **instancia de una clase que no extiende ni implementa nada especial**. Realmente es una clase sencilla que no depende de ningún framework especial.

```java
public class Persona{
    private String dni;
    private String nombre;
    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public Persona(String dni, String nombre) {
        super();
        this.dni = dni;
        this.nombre = nombre;
    }
}
```

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

· **Enterprise Java Bean (EJB)** es un componente de negocio Java Enterprise que implementa interfaces especiales y para su ejecución necesita un contenedor EJB/J2EE (Jboss, WAS, OAS, etc..):

```
package com.saludo;

import java.rmi.RemoteException;

import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HolaMundoBean implements SessionBean{
        private static final long serialVersionUID = 1L;// Quita warning de serialización del objeto

        // Nuestro método "de negocio"
        public String saludo(String nombre){
                System.out.println("Un cliente me ha invocado");
                return "Hola, " + nombre;
        }
        // Métodos del ciclo de vida del Bean (obligatorios)
        public void ejbCreate(){}
        public void ejbActivate() throws EJBException, RemoteException {}
        public void ejbPassivate() throws EJBException, RemoteException {}
        public void ejbRemove() throws EJBException, RemoteException {}
        public void setSessionContext(SessionContext arg0) throws EJBException, RemoteException {}
}
```

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

· ***JavaBean*** o simplemente **Bean** es una clase Java que cumple con ciertas normas con los nombres de sus propiedades y métodos.

Bean en inglés significa vaina, de ahí que sea una clase destinada a acceder y guardar datos de nuestro programas.

Su fin es encapsular información, para reutilizar código fuente, estructurando el código en unidades lo más sencillas posibles.

Un **JavaBean** es una clase **serializable** que debe tener un **constructor sin argumentos**, tener declarados todos sus **atributos como privados** y para cada uno de elllos un método **setter y getter**, añadiendole la palabra “set” o “get” al nombre del atributo.

```
import java.io.Serializable;

public class Persona implements Serializable{
    private String dni;
    private String nombre;
    public String getDni() {
        return dni;
    }
    public void setDni(String dni) {
        this.dni = dni;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public Persona() {
        super();
    }
    public Persona(String dni, String nombre) {
        super();
        this.dni = dni;
        this.nombre = nombre;
    }
}
```

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Una vez aclarados los conceptos vamos a ver un ***ejemplo práctico para entender como simplificamos el código con beans***:

¿Como podemos simplificar la creación de nuevos alumnos con Beans?

add-alumno.jsp

```
<%@ include file="../jspf/header.jspf"%>
<%@ include file="../jspf/menuSuperior.jspf"%>

    <div class="container">
        <h1>Nuevo alumno:</h1>
        <p>
            <font color="red">${errores}</font>
        </p>
        <p> Introduzca los datos del nuevo alumno:</p>

        <form action="add-alumno" method="post">
             <div class="form-row">
                    <div class="col">
                        <label>Dni:</label>
                        <input type="text" id="dni" name="dni" required
                            class="form-control" minlength="9"/>
                    </div>
                    <div class="col">
                        <label>Nombre:</label>
                        <input type="text" id="nombre" name="nombre" required
                        class="form-control" minlength="5"/>
                    </div>
            </div>
```

AlumnoController.java

```
        @RequestMapping(value="add-alumno",method = RequestMethod.POST)
        public String addAlumno(@RequestParam String dni,
                @RequestParam String nombre,
                @RequestParam String edad,
                @RequestParam String ciclo,
                @RequestParam String curso,
                ModelMap model) {
            String errores="";
            paginaServicio.setPagina(pagina);
            model.addAttribute("pagina",paginaServicio.getPagina());
            try {
                alumnoService.addAlumno(new Alumno(
                        dni,nombre,Integer.parseInt(edad),
                        ciclo,Integer.parseInt(curso)));
                //Para evitar pasar parámetros inncesarios
                model.clear();
                /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
                return "redirect:list-alumno";
            } catch (NumberFormatException e) {
                errores=e.getMessage();
            } catch (AlumnoDuplicadoException e) {
                errores=e.toString();
            }
            //Si llegamos aquí ha habido un error porque no ejecuta linea 54
            model.addAttribute("errores",errores);
            return "add-alumno";
        }
```

1º Modificar add-alumno.jsp para poder utilizar los tags de la librería de spring en nuestros formularios:
<%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>

add-alumno.jsp

```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ include file="../jspf/header.jspf"%>
3  <%@ include file="../jspf/menuSuperior.jspf"%>
4
5      <div class="container">
6          <h1>Nuevo alumno:</h1>
7          <p>
8              <font color="red">${errores}</font>
9          </p>
10         <p> Introduzca los datos del nuevo alumno:</p>
11
12         <form method="post" action="add-alumno">
13             <div class="form-row">
14                 <div class="col">
15                     <label path="dni">Dni:</label>
16                     <input path="dni" type="text" required="required"
17                     class="form-control" minlength="9"/>
18                 </div>
19                 <div class="col">
20                     <label path="nombre">Nombre:</label>
21                     <input type="text" id="nombre" path="nombre"
22                     required ="required" class="form-control" minlength="5"/>
23                 </div>
24             </div>
```

... continuación 1º Modificar add-alumno.jsp :

Ahora ya podemos utilizar la sintaxis de la librería bajo el prefijo “mvc” en el formulario del fichero jsp: **mvc:form, modelAttribute, mvc:label y mvc:input**.

Importante: Mirar como especificar que un campo es required en el ejemplo. Hace falta añadir **“path”** indicando el nombre del campo al que hacen referencia y quitar “name”.

Modifica el formulario , etiquetas (label) y campos (inputs) para utilizar la anotación “mvc” que hace referencia los tags de spring form:

add-alumno.jsp

```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ include file="../jspf/header.jspf"%>
3  <%@ include file="../jspf/menuSuperior.jspf"%>
4
5      <div class="container">
6          <h1>Nuevo alumno:</h1>
7          <p>
8              <font color="red">${errores}</font>
9          </p>
10         <p> Introduzca los datos del nuevo alumno:</p>
11
12         <form method="post" action="add-alumno">
13             <div class="form-row">
14                 <div class="col">
15                     <label path="dni">Dni:</label>
16                     <input path="dni" type="text" required="required"
17                     class="form-control" minlength="9"/>
18                 </div>
19                 <div class="col">
20                     <label path="nombre">Nombre:</label>
21                     <input type="text" id="nombre" path="nombre"
22                     required ="required" class="form-control" minlength="5"/>
23                 </div>
24             </div>
```

add-alumno.jsp

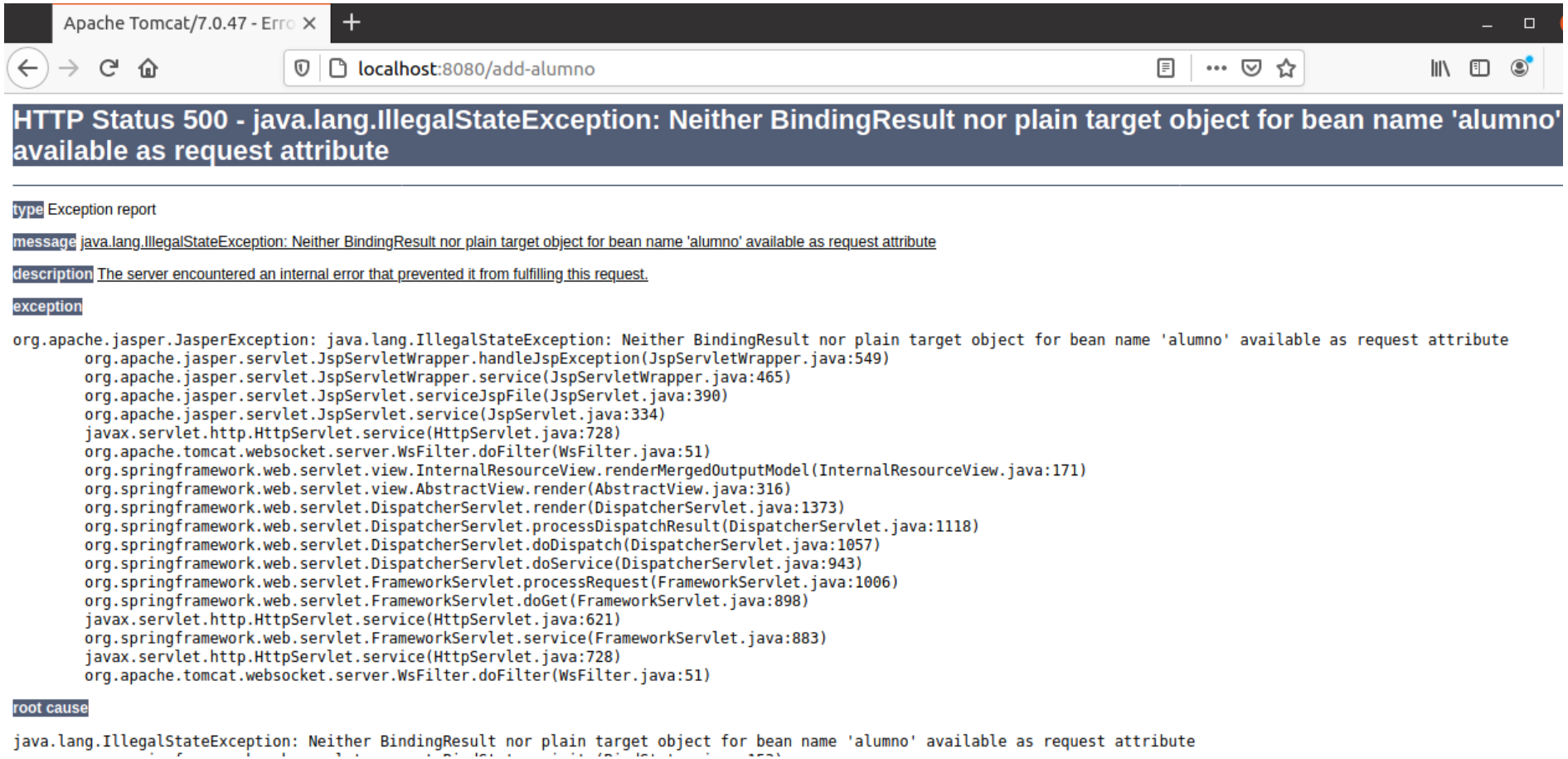
```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ include file="../jspf/header.jspf"%>
3  <%@ include file="../jspf/menuSuperior.jspf"%>
4
5      <div class="container">
6          <h1>Nuevo alumno:</h1>
7          <p>
8              <font color="red">${errores}</font>
9          </p>
10         <p> Introduzca los datos del nuevo alumno:</p>
11
12         <mvc:form method="post" action="add-alumno" modelAttribute="alumno">
13             <div class="form-row">
14                 <div class="col">
15                     <mvc:label path="dni">Dni:</mvc:label>
16                     <mvc:input path="dni" type="text" required="required"
17                     class="form-control" minlength="9"/>
18                 </div>
19                 <div class="col">
20                     <mvc:label path="nombre">Nombre:</mvc:label>
21                     <mvc:input type="text" id="nombre" path="nombre"
22                     required ="required" class="form-control" minlength="5"/>
23                 </div>
24             </div>
```

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

... continuación 1º Modificar add-alumno.jsp :

Si lo ejecutamos vemos que hay un problema, el controlador no tiene ningún bean “alumno”:



2º Modificar Alumno para que sea un bean (lo hacemos serializable y le añadimos un constructor vacio) :

Alumno.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model;
2
3  import java.io.Serializable;
4
5  public class Alumno implements Serializable, Comparable<Alumno>{
6      private static final long serialVersionUID = 1L;
7      private String dni;
8      private String nombre;
9      private Integer edad;
10     private String ciclo;
11     private Integer curso;
12
13     public Alumno() {
14     }
15
16     public Alumno(String dni) {
17         super();
18         this.dni = dni;
19     }
20
21     public Alumno(String dni, String nombre, Integer edad, String ciclo, Integer curso) {
22         super();
23         this.dni = dni;
```

3º Incluir el bean en el modelo para solucionar el error de ejecución:

AlumnoController.java

```
34        @RequestMapping(value="add-alumno",method = RequestMethod.GET)
35        public String mostrarAlumno(ModelMap model) {
36            paginaServicio.setPagina(pagina);
37            model.addAttribute("pagina",paginaServicio.getPagina());
38            return "add-alumno";
39        }
```

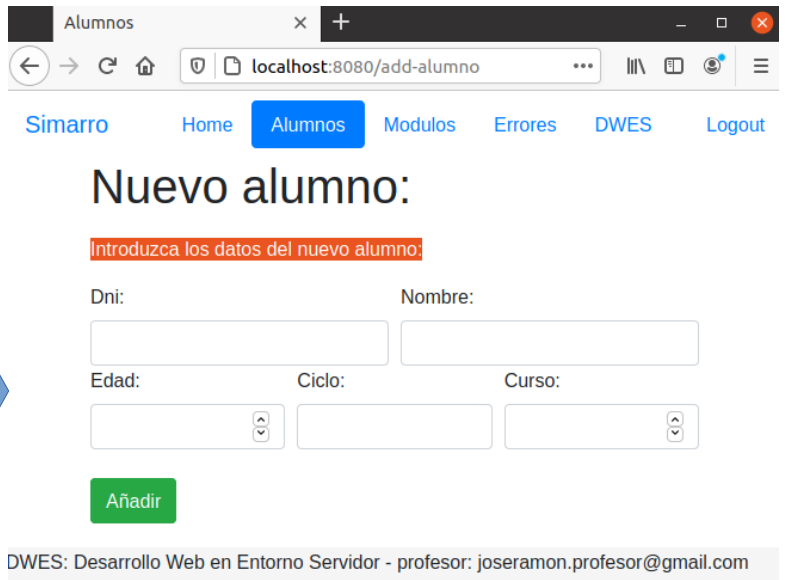
AlumnoController.java

```
34        @RequestMapping(value="add-alumno",method = RequestMethod.GET)
35        public String mostrarAlumno(ModelMap model) {
36            paginaServicio.setPagina(pagina);
37            model.addAttribute("pagina",paginaServicio.getPagina());
38            model.addAttribute("alumno",new Alumno());
39            return "add-alumno";
40        }
```

Y si desde el listado pulsamos en el botón “Añadir alumno” ya nos mostrará la pantalla:

add-alumno.jsp

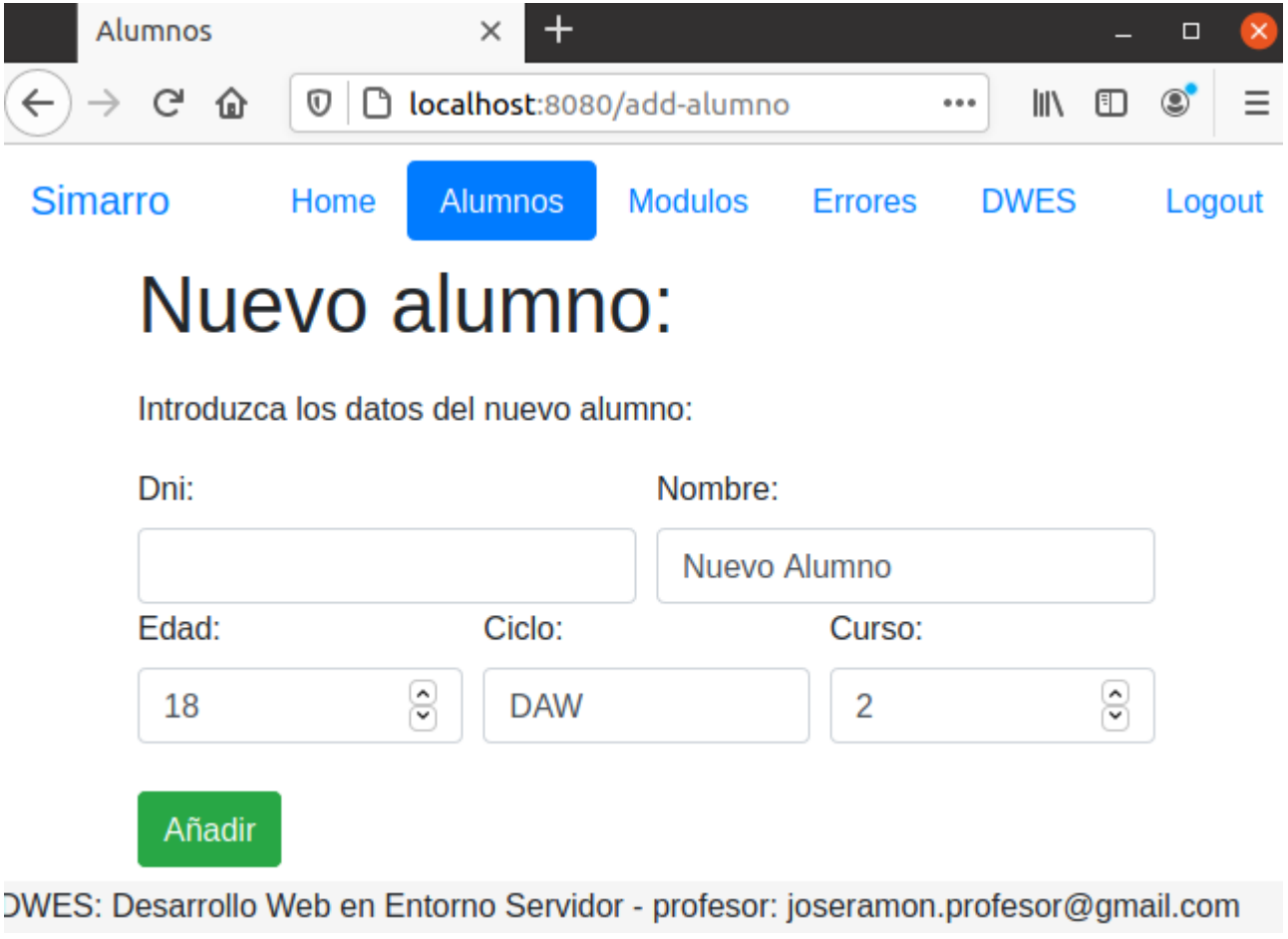
```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ include file="../jspf/header.jspf"%>
3  <%@ include file="../jspf/menuSuperior.jspf"%>
4
5      <div class="container">
6          <h1>Nuevo alumno:</h1>
7          <p>
8              <font color="red">${errores}</font>
9          </p>
10         <p> Introduzca los datos del nuevo alumno:</p>
11
12         <mvc:form method="post" action="add-alumno" modelAttribute="alumno">
13             <div class="form-row">
14                 <div class="col">
15                     <mvc:label path="dni">Dni:</mvc:label>
16                     <mvc:input path="dni" type="text" required="required"
17                     class="form-control" minlength="9"/>
18                 </div>
19                 <div class="col">
20                     <mvc:label path="nombre">Nombre:</mvc:label>
21                     <mvc:input type="text" id="nombre" path="nombre"
22                     required ="required" class="form-control" minlength="5"/>
23                 </div>
24             </div>
```



UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

¿Como podemos darle valores por defecto a los campos de nuestro formulario al añadir un nuevo alumno?



La primera idea sería incluir un tag value=”valorPorDefecto” en los inputs del JSP, pero no aprovecharíamos toda la potencia de Spring MVC.

La segunda solución es **crear un Alumno con valores por defecto directamente en el método del controlador que muestra el formulario** y **no hará falta modificar el JSP** porqué estos valores se copiarán automáticamente a los campos del formulario. Modifica el controlador para crear estos valores por defecto:

AlumnoController.java

```
34      @RequestMapping(value="add-alumno",method = RequestMethod.GET)
35      public String mostrarAlumno(ModelMap model) {
36          paginaServicio.setPagina(pagina);
37          model.addAttribute("pagina",paginaServicio.getPagina());
38          model.addAttribute("alumno",new Alumno("","Nuevo Alumno",18,"DAW",2));
39          return "add-alumno";
40      }
```

add-alumno.jsp

```
1  <%@taglib uri="http://www.springframework.org/tags/form" prefix="mvc"%>
2  <%@ include file="../jspf/header.jspf"%>
3  <%@ include file="../jspf/menuSuperior.jspf"%>
4
5      <div class="container">
6          <h1>Nuevo alumno:</h1>
7          <p>
8              <font color="red">${errores}</font>
9          </p>
10         <p> Introduzca los datos del nuevo alumno:</p>
11
12         <mvc:form method="post" action="add-alumno" modelAttribute="alumno">
13             <div class="form-row">
14                 <div class="col">
15                     <mvc:label path="dni">Dni:</mvc:label>
16                     <mvc:input path="dni" type="text" required="required"
17                     class="form-control" minlength="9"/>
18                 </div>
19                 <div class="col">
20                     <mvc:label path="nombre">Nombre:</mvc:label>
21                     <mvc:input type="text" id="nombre" path="nombre"
22                     required ="required" class="form-control" minlength="5"/>
23                 </div>
24             </div>
```

Estamos empezando a ver la mágia de Spring MVC!!!

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

¿Podemos simplificar aún más el controlador gracias a los beans?

AlumnoController.java

```
41        @RequestMapping(value="add-alumno",method = RequestMethod.POST)
42        public String addAlumno(@RequestParam String dni,
43                @RequestParam String nombre,
44                @RequestParam String edad,
45                @RequestParam String ciclo,
46                @RequestParam String curso,
47                ModelMap model) {
48            String errores="";
49            paginaServicio.setPagina(pagina);
50            model.addAttribute("pagina",paginaServicio.getPagina());
51            try {
52                alumnoService.addAlumno(new Alumno(
53                        dni,nombre,Integer.parseInt(edad),
54                        ciclo,Integer.parseInt(curso)));
55                //Para evitar pasar parámetros inncesarios
56                model.clear();
57                /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
58                return "redirect:list-alumno";
59            } catch (NumberFormatException e) {
60                errores=e.getMessage();
61            } catch (AlumnoDuplicadoException e) {
62                errores=e.toString();
63            }
64            //Si llegamos aquí ha habido un error porque no ejecuta linea 54
65            model.addAttribute("errores",errores);
66            return "add-alumno";
67        }
68
69        @RequestMapping(value="del-alumno",method = RequestMethod.GET)
70        public String delAlumno(ModelMap model,@RequestParam String dni) {
71            alumnoService.delAlumno(new Alumno(dni));
72            model.clear();
73            /* Para evitar recargar la página e intentar borrar algo ya borrado
74             * redirigimos a listar*/
75            return "redirect:list-alumno";
76        }
77    }
```

UD 2: Modelo Vista Controlador
6.- Formularios: Beans

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Como os habréis imaginado SI podemos simplificar el controlador utilizando beans. ¿Para que pasarle los parámetros del formulario al controlador si sabemos que es una instancia de la clase “alumno” (se lo hemos especificado en el formulario con “modelAttribute”) ?

AlumnoController.java

```
41      @RequestMapping(value="add-alumno",method = RequestMethod.POST)
42      public String addAlumno(@RequestParam String dni,
43              @RequestParam String nombre,
44              @RequestParam String edad,
45              @RequestParam String ciclo,
46              @RequestParam String curso,
47              ModelMap model) {
48          String errores="";
49          paginaServicio.setPagina(pagina);
50          model.addAttribute("pagina",paginaServicio.getPagina());
51          try {
52              alumnoService.addAlumno(new Alumno(
53                      dni,nombre,Integer.parseInt(edad),
54                      ciclo,Integer.parseInt(curso)));
55              //Para evitar pasar parámetros inncesarios
56              model.clear();
57              /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
58              return "redirect:list-alumno";
59          } catch (NumberFormatException e) {
60              errores=e.getMessage();
61          } catch (AlumnoDuplicadoException e) {
62              errores=e.toString();
63          }
64          //Si llegamos aquí ha habido un error porque no
65          model.addAttribute("errores",errores);
66          return "add-alumno";
67      }
```

Modifica el controlador:

De momento no nos preocupemos de comprobar si un campo es númerico o no.

AlumnoController.java

```
41      @RequestMapping(value="add-alumno",method = RequestMethod.POST)
42      public String addAlumno(Alumno alumno,ModelMap model) {
43          String errores="";
44          paginaServicio.setPagina(pagina);
45          model.addAttribute("pagina",paginaServicio.getPagina());
46          try {
47              alumnoService.addAlumno(alumno);
48              //Para evitar pasar parámetros inncesarios
49              model.clear();
50              /* Para evitar inserciones duplicadas comentamos código y redirigimos a listar*/
51              return "redirect:list-alumno";
52          } catch (AlumnoDuplicadoException e) {
53              errores=e.toString();
54              model.addAttribute("errores",errores);
55              return "add-alumno";
56          }
57      }
```

UD 2: Modelo Vista Controlador
5.- Redirecciones

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

EJERCICIO:

Sigue todos los pasos de los PDF**.** 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 UD2_practica2_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.