UD 2: Modelo Vista Controlador
12.- DTOs

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

Objetivos de la sesión:

- *Entender que es un **DTO (Data Transfer Object)**.*
- *Comprender **como utilizar los DTOs** en nuestra webapp.*
- *Aprender a **configurar el mapeo de beans con MapStruct**.*
- *Entender las **ventajas de trabajar con DTOs**.*
- *Aprender a **configurar y utilizar Lombok** para evitar código redundante.*

UD 2: Modelo Vista Controlador
12.- DTOs

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

¿Es lógico que para guardarnos el usuario que esta logeado nos guardemos todos los datos de la estructura del bean “Usuario” (incluido el pw)?
¿Tiene sentido que para listar los alumnos pasemos el bean “Alumno” con todos sus atributos al jsp? ¿Que pasaría si el alumno tuviera 100 atributos?
¿Tenemos un buen diseño si cuando guardamos las modificaciones de un alumno se borran los documentos asociados si no vamos con cuidado?
¿Que pasa si queremos transferir algún atributo del bean en un formato diferente o procesarlo de alguna manera, como por ejemplo convertir una fecha en un texto con un formato determinado?

¿Como podemos mejorar el diseño de nuestra aplicación para solucionar estos problemas?

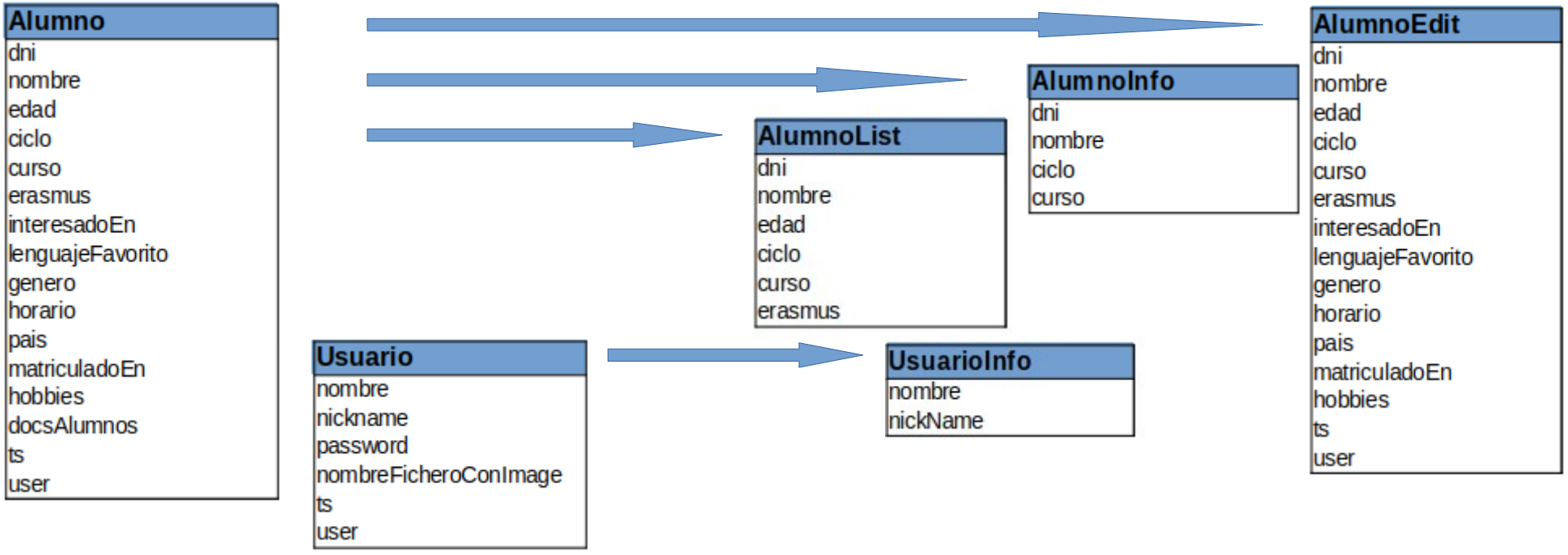
Usuario
nombre
nickname
password
nombreFicheroConImage
ts
user

Alumno
dni
nombre
edad
ciclo
curso
erasmus
interesadoEn
lenguajeFavorito
genero
horario
pais
matriculadoEn
hobbies
docsAlumnos
ts
user

Por estos y otros problemas surge el **patrón de diseñó "Data Transfer Object (DTO)"**:

Realmente **los DTOs son los beans que contiene los datos que se mostraran en la vista** (beans que se transfieren del controller al jsp).

Por ejemplo, a ***list-alumnos.jsp*** transferiremos una lista de ***"AlumnoList"*** para mostrar SOLO los datos que aparecen en el listado de alumnos, o un elemento de tipo ***"AlumnoEdit"*** para ***modificar-alumno.jsp***, o un ***"AlumnoInfo"*** en ***docAlumnos.jsp***

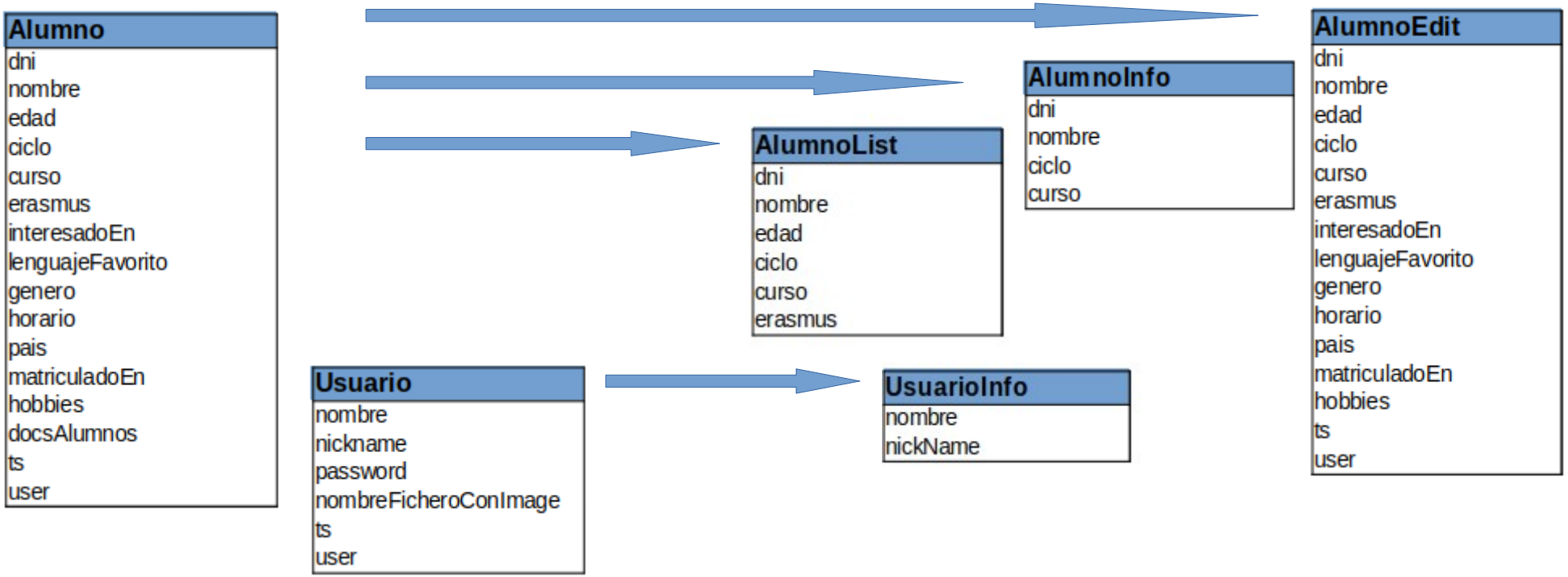


UD 2: Modelo Vista Controlador
12.- DTOs

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

El **Backend** deberá encargarse de realizar el proceso necesario para **mapear los datos de las estructuras iniciales que en siguientes prácticas estarán en BD (Usuario, Alumno,..) a los DTOs**, que son las beans con los datos (UsuarioInfo, AlumnoList, AlumnoEdit, …) que realmente necesita la vista (jsp's).

En una **aplicación profesional los controladores y las vistas (jsp`s) siempre trabajarán con DTOs** y son las implementaciones del **Service** quienes se encargar de utilizar los **mapeadores** para transformar las clases con los datos en DTOs que contienen los datos que son transferidos al controller y al revés.

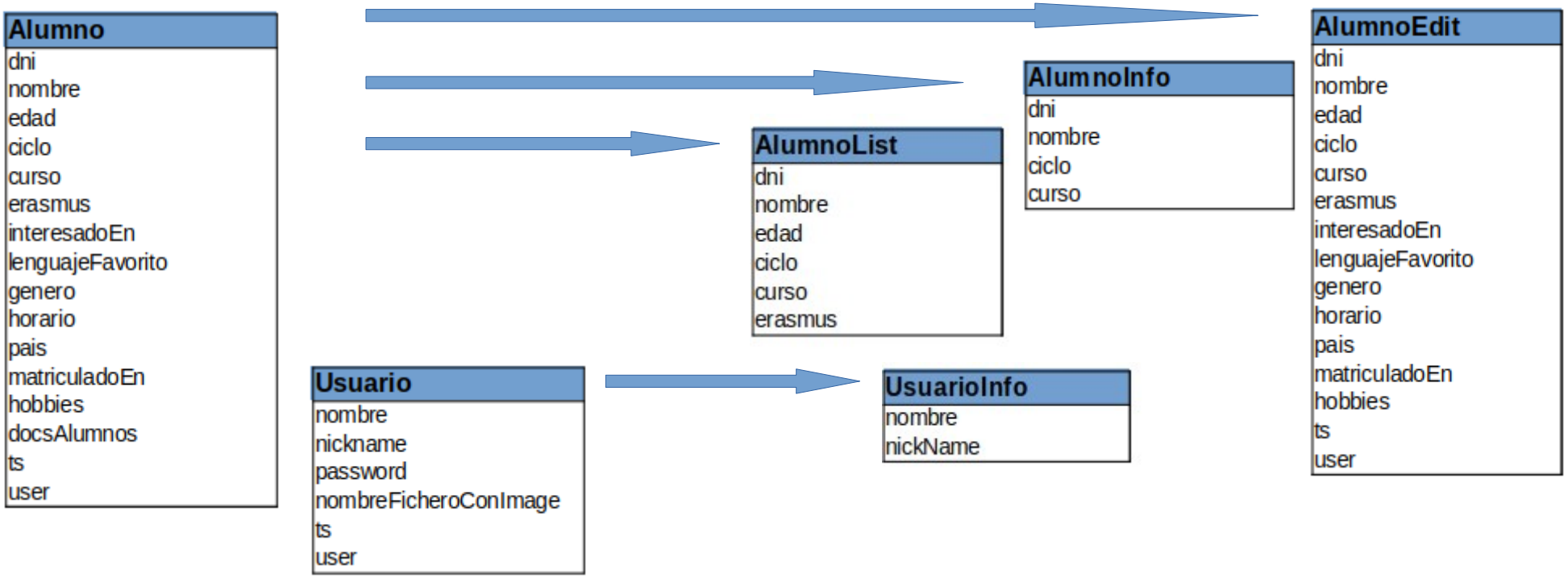


UD 2: Modelo Vista Controlador
12.- DTOs

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

Esta arquitectura permite **desacoplar la funcionalidad de una aplicación** (lógica de negocio) **del origen de los datos**. Por ejemplo: Si queremos utilizar otra fuente de datos tan solo hay que cambiar la capa de acceso a datos (Servicio), pero no hace falta tocar nada de las clases con la lógica de negocio. No solo podemos pasar de una BD a otra, podemos pasar de tener los datos en BD a ser ficheros xml u otro tipo de manera transparente sin que el controller se entere del cambio.

¿Como mapearemos los datos de una estructura a otra?



UD 2: Modelo Vista Controlador
12.- DTOs

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

Mapear o convertir una instancia de un objeto **“Alumno”** a una instancia de un objeto **“AlumnoEdit”** (en el paquete dto) significa crear un objeto de tipo “AlumnoEdit” y luego rellenar en “AlumnoEdit” todos los atributos con los datos de “Alumno”. Cuando son muchos atributos esta tarea se hace muy engorrosa.

Fijate que MAS ADELANTE cuando implementemos **AlumnoEdit no contendrà la lista de DocAlumnos** porque en modificar-alumno.jsp no mostraremos ni modificaremos los documentos.

Alumno.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model;

import java.io.Serializable;


public class Alumno implements Modificable<Alumno>,Serializable, Comparable<Alumno>{
    private static final long serialVersionUID = 1L;
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni
    private Integer edad;
    @Size(min = 3, message = "El ciclo debe tener almenos 3 carácteres")
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto
    @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
    private Integer curso;

    private boolean erasmus=false;
    private String[] interesadoEn;
    private String lenguajeFavorito="";
    private String genero;
    private String horario;
    private String pais;
    private ArrayList<Integer> matriculadoEn;
    private String hobbies;
    private ArrayList<DocAlumno> docsAlumno;
    private Date ts;
    private String user;
```

AlumnoEdit.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.dto;

import java.io.Serializable;


public class AlumnoEdit implements Serializable{
    private static final long serialVersionUID = 1L;
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 ca
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener d
    private Integer edad;
    @Size(min = 3, message = "El ciclo debe tener almenos 3 carácteres")
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato
    @Range(min = 1, max = 2, message = "El curso solo admite los valores
    private Integer curso;

    private boolean erasmus=false;
    private String[] interesadoEn;
    private String lenguajeFavorito="";
    private String genero;
    private String horario;
    private String pais;
    private ArrayList<Integer> matriculadoEn;
    private String hobbies;
    private Date ts;
    private String user;
```

¿Que herramienta podemos utilizar para realizar esta conversión fácilmente sin tener que hacerlo nosotros manualmente?

UD 2: Modelo Vista Controlador
12.- DTOs

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

Para realizar **conversiones entre estructuras de datos y poder mapear los datos** desde "Alumno" a "AlumnoEdit" **maven** proporciona la **dependencia MapStruct**. Realicemos los siguientes cambios a nuestro fichero pom.xml:
Se proporciona el código en un *fichero comprimido en el DRIVE (subcarpeta p07)*

joseramon_primer_app_spring_mvc/pom.xml

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>org.profesor.joseramon</groupId>
4    <artifactId>joseramon_primer_app_spring_mvc</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6    <packaging>war</packaging>
7    <properties>
8      <org.mapstruct.version>1.4.1.Final</org.mapstruct.version>
9  </properties>
10   <dependencies>
11         <dependency>
12             <groupId>javax</groupId>
13             <artifactId>javaee-web-api</artifactId>
14             <version>8.0</version>
15             <scope>provided</scope>
```

joseramon_primer_app_spring_mvc/pom.xml

```
48         <!-- Apache Commons IO -->
49         <dependency>
50             <groupId>commons-io</groupId>
51             <artifactId>commons-io</artifactId>
52             <version>2.8.0</version>
53         </dependency>
54         <!-- DTOs -->
55         <dependency>
56             <groupId>org.mapstruct</groupId>
57             <artifactId>mapstruct</artifactId>
58             <version>${org.mapstruct.version}</version>
59             <scope>compile</scope>
60         </dependency>
61     </dependencies>
62     <build>
63         <pluginManagement>
```

joseramon_primer_app_spring_mvc/pom.xml

```
60         </dependency>
61     </dependencies>
62     <build>
63         <pluginManagement>
64             <plugins>
65                 <plugin>
66                     <groupId>org.apache.maven.plugins</groupId>
67                     <artifactId>maven-compiler-plugin</artifactId>
68                     <version>3.8.1</version>
69                     <configuration>
70                         <verbose>true</verbose>
71                         <source>1.8</source>
72                         <target>1.8</target>
73                         <encoding>UTF-8</encoding>
74                         <showWarnings>true</showWarnings>
75                         <annotationProcessorPaths>
76                             <path>
77                                 <groupId>org.mapstruct</groupId>
78                                 <artifactId>mapstruct-processor</artifactId>
79                                 <version>${org.mapstruct.version}</version>
80                             </path>
81                             <!-- other annotation processors -->
82                         </annotationProcessorPaths>
83                     </configuration>
84                 </plugin>
```

Definir 'properties' nos permite poder actualizar la versión el día de mañana sin olvidarnos de cambiarla en los 2 sitios.

UD 2: Modelo Vista Controlador
12.- DTOs

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

Abrimos la subcarpeta p08 y debemos copiar Alumno.java o comprobar que nuestra clase Alumno.java tenga los mismos atributos y tipos que la clase Alumno.java de la carpeta comprimida porque si no la conversión a los DTOs AlumnoEdit, AlumnoInfo y AlumnoList no funcionará. Ten en cuenta que las restricciones (@Pattern, @NotNull,…) pueden variar pero los tipos de los atributos (String,…) NO:

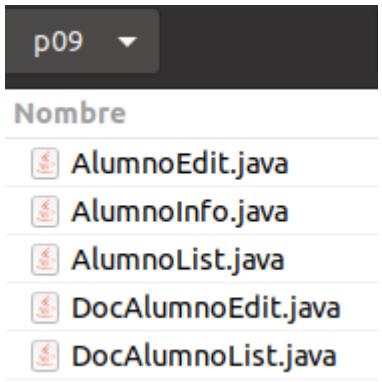
Alumno.java

```java
package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model;

import java.io.Serializable;


public class Alumno implements Modificable<Alumno>,Serializable, Comparable<Alumno>{
    private static final long serialVersionUID = 1L;
    @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
    private String dni;
    @Size(min=5,message="El nombre debe de tener un tamaño mínimo de 5 carácteres")
    private String nombre;
    @NotNull(message = "La edad no puede estar vacia")
    @Range(min = 18, max = 99, message = "La edad debe ser igual o mayor a 18 y menor o igual a 99")
    @Digits(integer = 2,fraction = 0, message = "La edad no puede tener decimales ni más de 2 dígitos")
    private Integer edad;
    @Size(min = 3, message = "El ciclo debe tener almenos 3 carácteres")
    private String ciclo;
    @NotNull(message = "El curso no puede estar vacio")
    @Digits(fraction = 0, integer = 1,message = "El curso tiene un formato incorrecto")
    @Range(min = 1, max = 2, message = "El curso solo admite los valores 1 o 2")
    private Integer curso;

    private boolean erasmus=false;
    private String[] interesadoEn;
    private String lenguajeFavorito="";
    private String genero;
    private String horario;
    private String pais;
    private ArrayList<Integer> matriculadoEn;
    private String hobbies;
    private ArrayList<DocAlumno> docsAlumno;
    private Date ts;
    private String user;
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

Abrimos la subcarpeta p09 y copiamos los ficheros a nuestro proyecto teniendo en cuenta que debemos poner el nombre del paquete correcto y los ficheros estaranen la subcarpeta model.dto:



AlumnoEdit.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.dto;
2
3  import java.io.Serializable;
16
17
18 public class AlumnoEdit implements Modificable<AlumnoEdit>, Serializable{
19     private static final long serialVersionUID = 1L;
20     @Pattern(regexp = "[0-9]{8}[A-Za-z]{1}", message = "El dni debe tener 8 números y una letra")
21     private String dni;
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

Copiamos de la subcarpeta p10 la **interface AlumnoMapper** y la pegamos en el subpaquete "srv.mapper". Esta interface mapeará los datos de Alumno hacia diferentes tipos de beans. A continuación se explica en más detalle:
Como estandard "totalmente personal" he utilizado las siguientes coletillas para dar nombre a las clases DTO nuevas:

·BEAN+Edit:
Dto para modificar
el bean

·BEAN+List:
Dto para listar beans
(Solo lectura)

· BEAN+Info:
Información reducida
del Bean.
(No contiene todos
los campos y es de
solo lectura)

AlumnoMapper.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.srv.mapper;
2
3  import java.util.List;
16
17 @Mapper
18 public interface AlumnoMapper {
19     AlumnoMapper INSTANCE= Mappers.getMapper(AlumnoMapper.class);
20
21     //Devuelve un objeto de tipo 'AlumnoEdit' a partir de un objeto de tipo 'Alumno'
22     AlumnoEdit alumnoToAlumnoEdit(Alumno alumno);
23     //Devuelve un objeto de tipo 'AlumnoList' a partir de un objeto de tipo 'Alumno'
24     @Mapping(target="modificado",source = "alumno.ts", dateFormat = "dd/MM/yy HH:mm:ss")
25     AlumnoList alumnoToAlumnoList(Alumno alumno);
26     //Devuelve una lista de objetos 'AlumnoList' a partir de una lista de tipo 'Alumno'
27
28     List<AlumnoList> alumnosToAlumnosList(List<Alumno> alumnos);
29
30     @Mapping(source="dni",target="dni_alumno")
31     @Mapping(source="nombre",target="nombre_alumno")
32     @Mapping(source="ciclo",target="ciclo_alumno")
33     @Mapping(source="curso",target="curso_alumno")
34         //Devuelve un objeto de tipo 'AlumnoInfo' a partir de un objeto de tipo 'Alumno'
35     AlumnoInfo alumnoToAlumnoInfo(Alumno alumno);
36         //Devuelve un objeto de tipo 'Alumno' a partir de un objeto de tipo 'AlumnoEdit'
37     Alumno alumnoEditToAlumno(AlumnoEdit alumnoEdit);
38
39     //Actualiza un objeto de tipo 'Alumno' con los datos de un objeto de tipo 'AlumnoEdit'
40     void updateAlumnoFromAlumnoEdit(AlumnoEdit alumnoEdit,@MappingTarget Alumno alumno);
41
42     //Devuelve un objeto de tipo 'DocAlumnoEdit' a partir de un objeto de tipo 'DocAlumno'
43     DocAlumnoEdit docAlumnoToDocAlumnoEdit(DocAlumno docAlumno);
44     //Devuelve una lista de objetos 'DocAlumnoList' a partir de una lista de tipo 'DocAlumno'
45     List<DocAlumnoList> docsAlumnoToDocsAlumnoList(List<DocAlumno> docAlumnos);
46     //Devuelve un objeto de tipo 'DocAlumno' a partir de un objeto de tipo 'DocAlumnoEdit'
47     DocAlumno docAlumnoEditToDocAlumno(DocAlumnoEdit docAlumnoEdit);
48 }
```

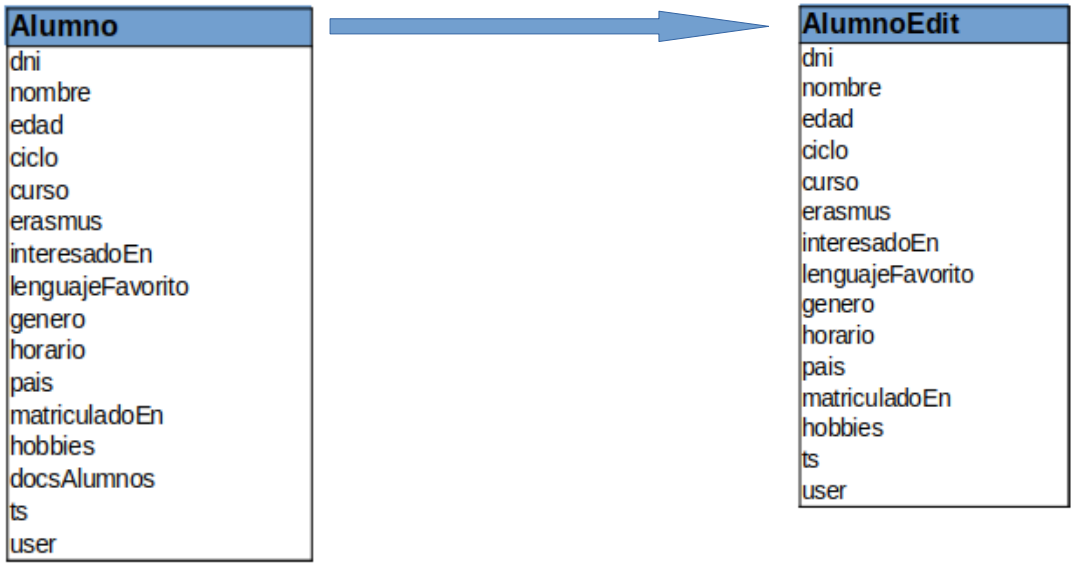
En cada método de 'AlumnoMapper' el mapeo es automático gracias a que hemos utilizado la anotación **@Mapper** en la linea 17 del código anterior.

Por ejemplo:

Queremos convertir un **objecto de tipo 'Alumno' a un objeto de tipo 'AlumnoEdit' para enviarselo al jsp** y para ello creamos el método 'alumnoToAlumnoEdit' en 'AlumnoMapper' :

```
21      //Devuelve un objeto de tipo 'AlumnoEdit' a partir de un objeto de tipo 'Alumno'
22      AlumnoEdit alumnoToAlumnoEdit(Alumno alumno);
```

MapStruct analizará los atributos 'Alumno' y 'AlumnoEdit' que tienen el mismo nombre y copiará todos los datos de esos atributos de 'Alumno' a 'AlumnoEdit':



‘AlumnoEdit’ solo contendrá los atributos que se visualizan en el jsp de modificación del Alumno, así nos evitaremos problemas como veremos más adelante.

Ahora la pregunta podría ser : ¿Que pasa si el bean origen tiene un atributo que queremos copiar al bean destino con un nombre de atributo distinto? Por ejemplo , el bean ‘Alumno’ tiene un atributo ‘dni’ y queremos copiar ese dato al bean ‘AlumnoInfo’ en el atributo ‘dni_alumno’.

Como es posible que los atributos no se llamen igual, tenemos la **notación @Mapping** que utilizamos antes de declarar el método para indicar el nombre del atributo en la clase original (por ejemplo 'dni') y el atributo donde copiarlo en la clase final (por ejemplo 'dni_alumno').

El mapeador copiará todos los datos de los atributos de ambas clases con el mismo nombre y adicionalmente copiará los datos de los atributos con nombres distintos gracias a la notación @Mapping:

```
30    @Mapping(source="dni",target="dni_alumno")
31    @Mapping(source="nombre",target="nombre_alumno")
32    @Mapping(source="ciclo",target="ciclo_alumno")
33    @Mapping(source="curso",target="curso_alumno")
34        //Devuelve un objeto de tipo 'AlumnoInfo' a partir de un objeto de tipo 'Alumno'
35    AlumnoInfo alumnoToAlumnoInfo(Alumno alumno);
```

Esta copia no la hace por arte de magia, sino que **cuando compilamos sin que nos enteremos genera el código correspondiente** a este proceso, por eso **siempre que cambiemos algún dato de algún bean implicado en un DTO tendremos que parar el Tomcat, hacer un clean y un update**.

De igual manera podemos hacer **conversiones o mapeos con listas** de beans:

En el ejemplo creamos una lista de objetos de tipo “AlumnoList” a partir de una lista de objetos de tipo “Alumno”.

```
26      //Devuelve una lista de objetos 'AlumnoList' a partir de una lista de tipo 'Alumno'
27
28      List<AlumnoList> alumnosToAlumnosList(List<Alumno> alumnos);
```

Si queremos utilizar DTOs en nuestra webapp, ahora la pregunta es:

¿Como utilizamos AlumnoMapper en nuestra webApp ?

UD 2: Modelo Vista Controlador
12.- DTOs

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

Veamos como utilizar AlumnoMapper con el proceso de alta de un nuevo alumno:

El controller enviará a la vista un objeto de tipo AlumnoEdit:

```
@RequestMapping(value = "/add-alumno", method = RequestMethod.GET)
public String mostrarAlumno(ModelMap model) {
    model.addAttribute("pagina", pagina);
    model.addAttribute("alumnoEdit", new AlumnoEdit());
    return "add-alumno";
}
```

La vista rellenará AlumnoEdit y se lo devolverá en POST al controller con los datos rellenos :

```
<mvc:form method="post" action="add-alumno" modelAttribute="alumnoEdit">
    <mvc:errors path="*" cssClass="text-warning" />
     <div class="form-row">
            <div class="col">
                <mvc:label path="dni"> <spring:message code="etiqueta.dni" />:</mvc:label>
                <mvc:input path="dni" type="text" class="form-control"/>
                <mvc:errors path="dni" cssClass="text-warning"/>
            </div>
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

El controller validará AlumnoEdit, por lo que las validaciones deberán existir en AlumnoEdit y deberemos de modificar las traducciones de los errores de los validadores para que en vez de ".alumno." sea ".alumnoEdit.". En esta pantalla todavía no intentaremos solucionar el error debido al parámetro incorrecto de alumnoService.addAlumno(alumnoEdit) :

```java
@RequestMapping(value = "/add-alumno", method = RequestMethod.POST)
public String addAlumno(ModelMap model, @Valid AlumnoEdit alumnoEdit, BindingResult validacion) {
    String errores = "";
    paginaServicio.setPagina(pagina);
    model.addAttribute("pagina", paginaServicio.getPagina());
    if (validacion.hasErrors()) {
        // Hay errores y debemos volver al formulario de alta
        return "add-alumno";
    }
    // Si llega aquí no hay errores de validación
    try {
        alumnoService.addAlumno(alumnoEdit);
        // Para evitar pasar parámetros inncesarios
        model.clear();
        /*
         * Para evitar inserciones duplicadas comentamos código y redirigimos a listar
         */
        return "redirect:list-alumno";
    } catch (AlumnoDuplicadoException e) {
        errores = e.toString();
        model.addAttribute("errores", errores);
        return "add-alumno";
    }
}
```

```
Pattern.alumnoEdit.dni=El dni debe tener 8 números y una letra
Size.alumnoEdit.nombre=El nombre debe de tener un tamaño mínimo de 5 carácteres
NotNull.alumnoEdit.edad=La edad no puede estar vacia
Range.alumnoEdit.edad=La edad debe ser igual o mayor a 18 y menor o igual a 99
Digits.alumnoEdit.edad=La edad no puede tener decimales ni más de 2 dígitos
Size.alumnoEdit.ciclo=El ciclo debe tener almenos 3 carácteres
NotNull.alumnoEdit.ciclo=El curso no puede estar vacio
Digits.alumnoEdit.curso=El curso tiene un formato incorrecto
Range.alumnoEdit.curso=El curso solo admite los valores 1 o 2
```

Cuando refactorizamos el código puede pasarnos que tengamos que hacer más cambios de los deseados, pero no por ello debemos de abandonar y dejar el diseño anterior que contiene deficiencias ...

UD 2: Modelo Vista Controlador
12.- DTOs

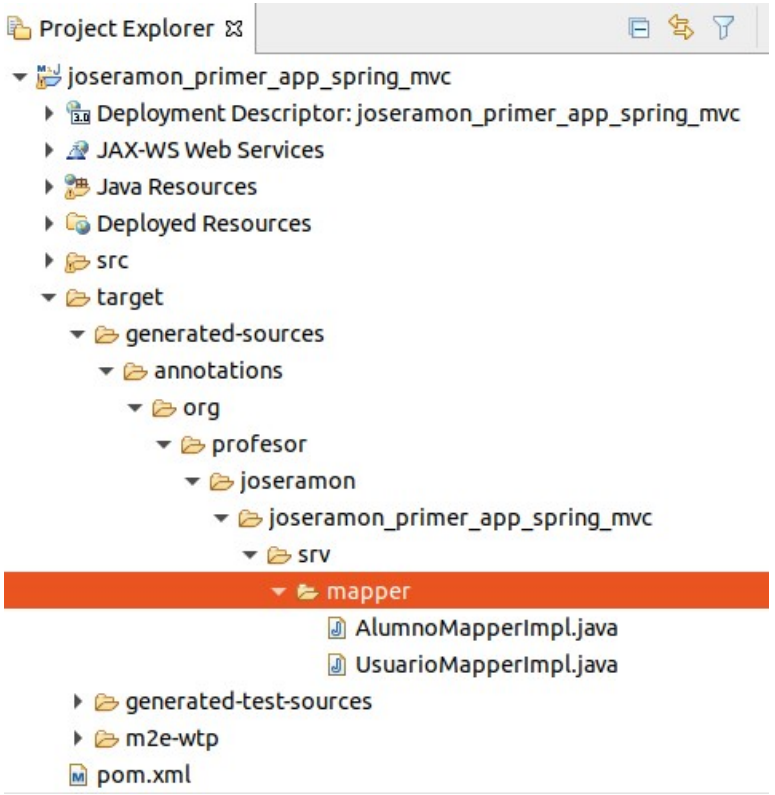
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Vamos a arreglar el Service alumnoService.addAlumno(alumnoEdit) cambiando el tipo de parámetro. Habrá que cambiar en AlumnoDuplicadoException el segundo parámetro para que sea de tipo AlumnoEdit. Si existeAlumno() no tiene como parámetro un String, habrá que copiar también el método existeAlumno(String dni):

```
//Método para añadir alumnos
public void addAlumno(AlumnoEdit alumnoEdit) throws AlumnoDuplicadoException {
    if (existeAlumno(alumnoEdit.getDni())) {
        throw new AlumnoDuplicadoException(encontrarAlumnoPorDni(alumnoEdit.getDni()),alumnoEdit);
    }else { //no existe y podemos añadirlo previa conversión de 'AlumnoEdit' a 'Alumno'
        alumnos.add(AlumnoMapper.INSTANCE.alumnoEditToAlumno(alumnoEdit));
    }
}

public boolean existeAlumno(String dni) {
    //Mapeamos de 'Alumno' a 'String' que contiene el dni y luego buscamos
    Optional<String> dniAlumno= alumnos.stream().map(a->a.getDni()).filter(id-> id.equals(dni)).findFirst();

    if (dniAlumno.isPresent()) {//Existe el dni
        return true;
    }
    //No existe el dni en la lista de alumnos
    return false;
}
```

Cuando 'Alumno' sea un registro de la BD se verá claramente que **es un error de programadores inexpertos utilizar las mismas clases que están mapeadas contra los registros de la Base de datos para transferirlas por toda la aplicación y enviarlas incluso a la vista**.

Es ***importante saber*** que para que en la aplicación vaya el mapper tiene que crear las clases que realmente copian cada uno de los atributos comunes de la clase AlumnoEdit a la clase Alumnot y este proceso se genera al realizar el install. Por ello cuando quieras ejectuar la aplicación deberás hacer un "Run as\Maven clean", un "Maven\Update project" y *para asegurarnos que se generan las clases de mapper "Run as\Maven install"*:



Sin el "install" la aplicación fallará porque no encuentra las clases del mapper!!

Pon en marcha la webApp, añade un alumno y comprueba que funciona.

OJO: Si se realiza algún cambio en los beans implicados en un mapeo (añadir campos , añadir getters/setters) da un error porque hace falta compilar el proyecto para que MapStruct funcione. Como ya hemos dicho, esto es así porque realmente las anotaciones no evitan que tengamos que tener un método donde se rellenan los atributos de un objeto, pero dicho código lo genera MapStruct automáticamente cuando se compila el proyecto de manera transparente al programador. **Si no se compila el proyecto, no se genera el código necesario y por eso falla.**

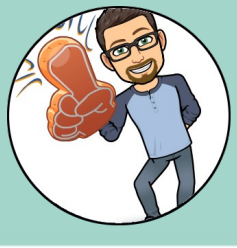
Para prácticar, cambia el listado de “Alumnos” por un listado de “AlumnosList” y comprueba que funciona correctamente.

Copiar una clase “Alumno” como “AlumnoEdit” o “AlumnoList” con sus getters y setters en un paquete “dto” puede llegar a ser redundante.

¿Podemos reducir esta redundancia evitando tener que escribir getters y setters?

UD 2: Modelo Vista Controlador
12.- DTOs

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



Para evitar tener que crear getters y setters vamos a utilizar la dependencia Lombok en su versión “1.18.20”. Para ello añadimos la dependencia y modificamos las anotaciones del procesador de anotaciones para evitar que MapStruct no funcione bien cuando utilizamos Lombok. El alumno deberá de añadir la propiedad ‘org.projectlombok.version’ al fichero pom.xml : Revisa la subcarpeta p20

joseramon_primer_app_spring_mvc/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http:/
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.profesor.joseramon</groupId>
  <artifactId>joseramon_primer_app_spring_mvc</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <properties>
    <org.projectlombok.version>1.18.20</org.projectlombok.version>
    <org.mapstruct.version>1.4.1.Final</org.mapstruct.version>
  </properties>
```

joseramon_primer_app_spring_mvc/pom.xml

```xml
            <artifactId>mapstruct</artifactId>
            <version>${org.mapstruct.version}</version>
            <scope>compile</scope>
        </dependency>
        <!-- Lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>${org.projectlombok.version}</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
```

joseramon_primer_app_spring_mvc/pom.xml

```xml
    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <verbose>true</verbose>
                        <source>1.8</source>
                        <target>1.8</target>
                        <encoding>UTF-8</encoding>
                        <showWarnings>true</showWarnings>
                        <annotationProcessorPaths>
                            <path>
                                <groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                                <version>${org.projectlombok.version}</version>
                            </path>
                            <!-- Necesario para que Lombok funcione con MapStruct -->
                            <path>
                                <groupId>org.projectlombok</groupId>
                                <artifactId>lombok-mapstruct-binding</artifactId>
                                <version>0.2.0</version>
                            </path>
                            <path>
                                <groupId>org.mapstruct</groupId>
                                <artifactId>mapstruct-processor</artifactId>
                                <version>${org.mapstruct.version}</version>
                            </path>
                            <!-- other annotation processors -->
                        </annotationProcessorPaths>
                    </configuration>
                </plugin>
```

¿Como utilizamos Lombok ?

Aunque Lombok tiene más funcionalidades y anotaciones, de momento solo vamos a utilizar las notaciones para generar getters y setters:

AlumnoList.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.dto;
2
3  import java.io.Serializable;
4  import lombok.Getter;
5  import lombok.Setter;
6
7  @Getter
8  @Setter
9  public class AlumnoList implements Serializable{
10     private static final long serialVersionUID = 1L;
11     private String dni;
12     private String nombre;
13     private Integer edad;
14     private String ciclo;
15     private Integer curso;
16     private boolean erasmus=false;
17
18     public AlumnoList() {
19     }
20
21     public String getErasmusChecked() {
22         if (erasmus)
23                 return "checked";
24         else
25             return "";
26     }
27 }
```

Borra los métodos de los getters y los setters de AlumnoList y añade las notaciones para que lombok los genere. Compila el proyecto y comprueba que funciona la webApp.

UD 2: Modelo Vista Controlador
12.- DTOs

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

¿Por que al ver los warning sobre los atributos nos dice que los valores no se están utilizando? ¿Por que ha compilado si aparentemente hay errores en AlumnoService?

AlumnoList.java

```
1  package org.profesor.joseramon.joseramon_primer_app_spring_mvc.model.dto;
2
3  import java.io.Serializable;
4  import lombok.Getter;
5  import lombok.Setter;
6
7  @Getter
8  @Setter
9  public class AlumnoList implements Serializable{
10     private static final long serialVersionUID = 1L;
11     private String dni;
12     private String nombre;
13     private Integer edad;
14     private String ciclo;
15     private Integer curso;
16     priv
17
18     publ
19     }
20
21     publ
22
23
24
25
26     }
27 }
```

AlumnoService.java

```
65          case "nombre":
66              Collections.sort(alumnos);
67              break;
68          }
69
70          return AlumnoMapper.INSTANCE.alumnosToAlumnosList(alumnos);
71      }
72
73      //Método para añadir alumnos
74      public void addAlumno(AlumnoEdit alumnoEdit) throws AlumnoDuplicadoException {
75          if (existeAlumno(alumnoEdit.getDni())) {
76              throw new AlumnoDuplicadoException(encontrarAlumnoPorDni(alumnoEdit.getDni()),alum
77          }else { //no existe y podemos añadirlo previa conversión de 'AlumnoEdit' a 'Alumno'
78              alumnos.add(AlumnoMapper.INSTANCE.alumnoEditToAlumno(alumnoEdit));
79          }
80      }
```

UD 2: Modelo Vista Controlador
12.- DTOs

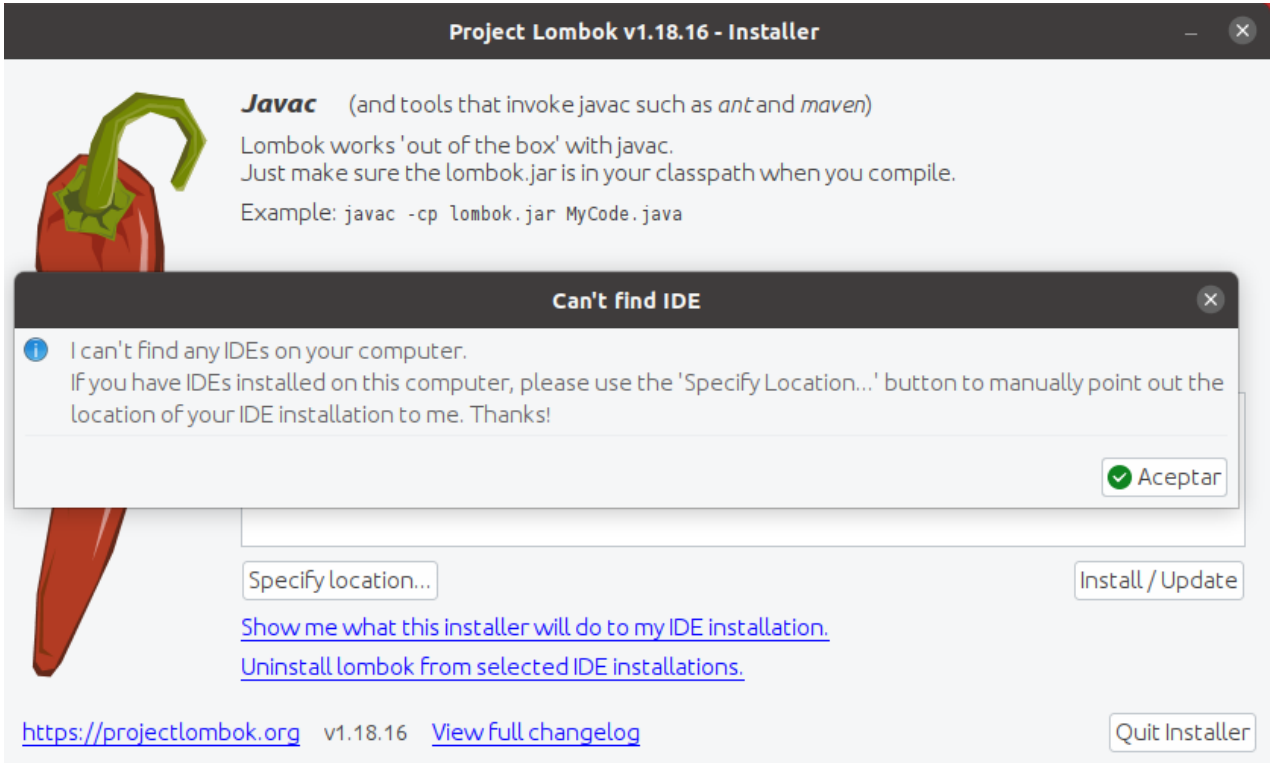
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Hemos configurado correctamente Lombok en nuestro proyecto maven, pero nuestro IDE Eclipse no se ha enterado de que Lombok ha generado los getters y los setters, así que vamos a configurar el plugin Lombok para Eclipse:

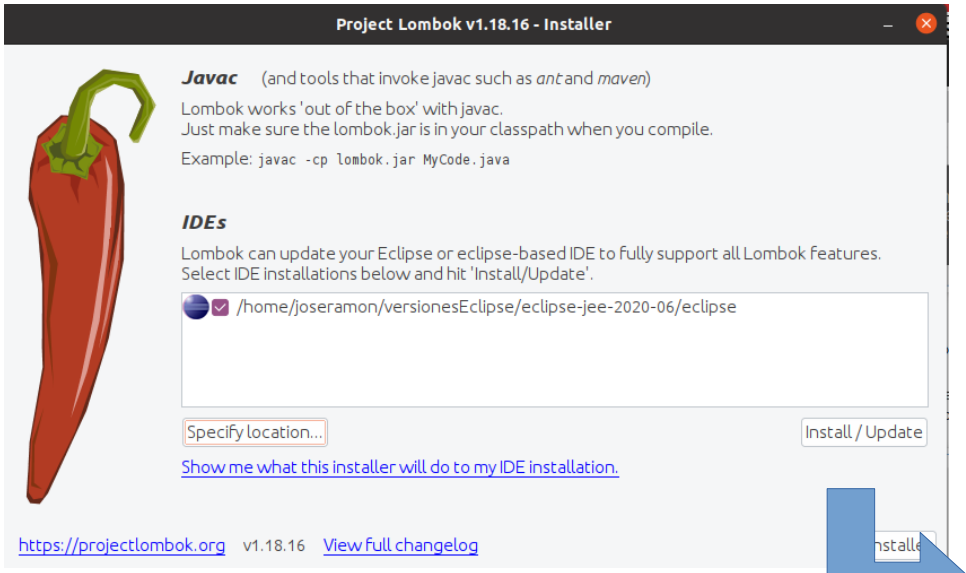
1º Descargamos Lombok de https://projectlombok.org/download

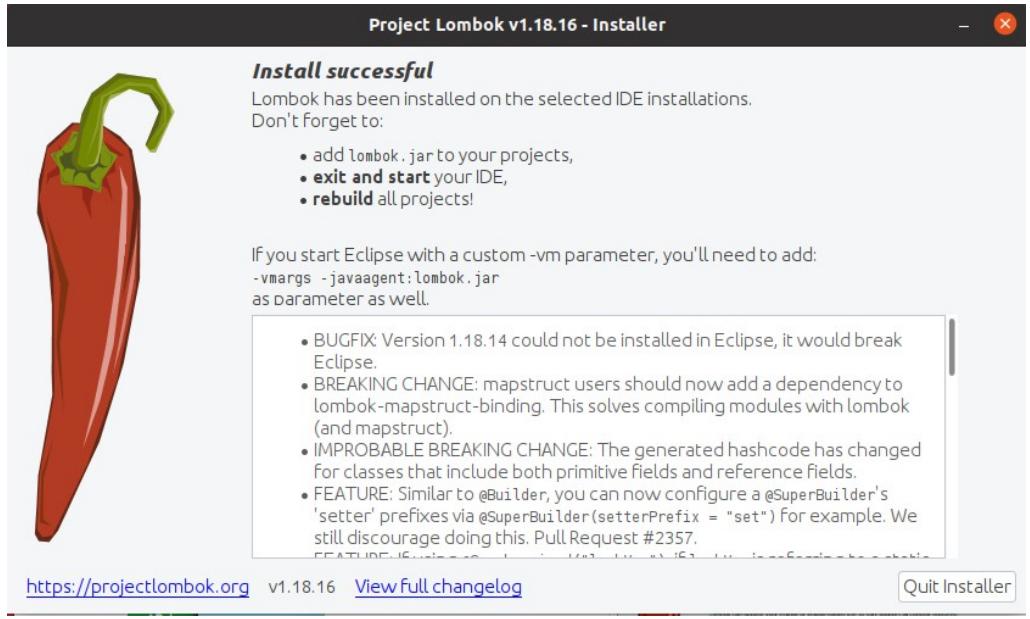
2º Si estamos en linux por defecto el fichero no tiene permisos de ejecución por seguridad. Le marcamos que es ejecutable:

3º Si ejecutamos el fichero suele “No detectar” a nuestro IDE:



4º Por lo que tendremos que decirselo manualmente desde el botón “Specify location…” y pulsaremos en “Install/Udate” :





5º Ya podemos arrrancar Eclipse.

Si actualizamos el proyecto Maven vemos que ya nos detecta los getters y los setters:

AlumnoService.java

```
67              break;
68          }
69
70          return AlumnoMapper.INSTANCE.alumnosToAlumnosList(alumnos);
71      }
72
73      //Método para añadir alumnos
74      public void addAlumno(AlumnoEdit alumnoEdit) throws AlumnoDuplicadoException {
75          if (existeAlumno(alumnoEdit.getDni())) {
76              throw new AlumnoDuplicadoException(encontrarAlumnoPorDni(alumnoEdit.getDni(
77          }else { //no existe y podemos añadirlo previa conversión de 'AlumnoEdit' a 'Alu
78              alumnos.add(AlumnoMapper.INSTANCE.alumnoEditToAlumno(alumnoEdit));
79          }
80      }
```

Simplifica el resto de beans que hay en el subpaquete “dto” de tu aplicación para que no tengan ni getters ni setters gracias a Lombok.

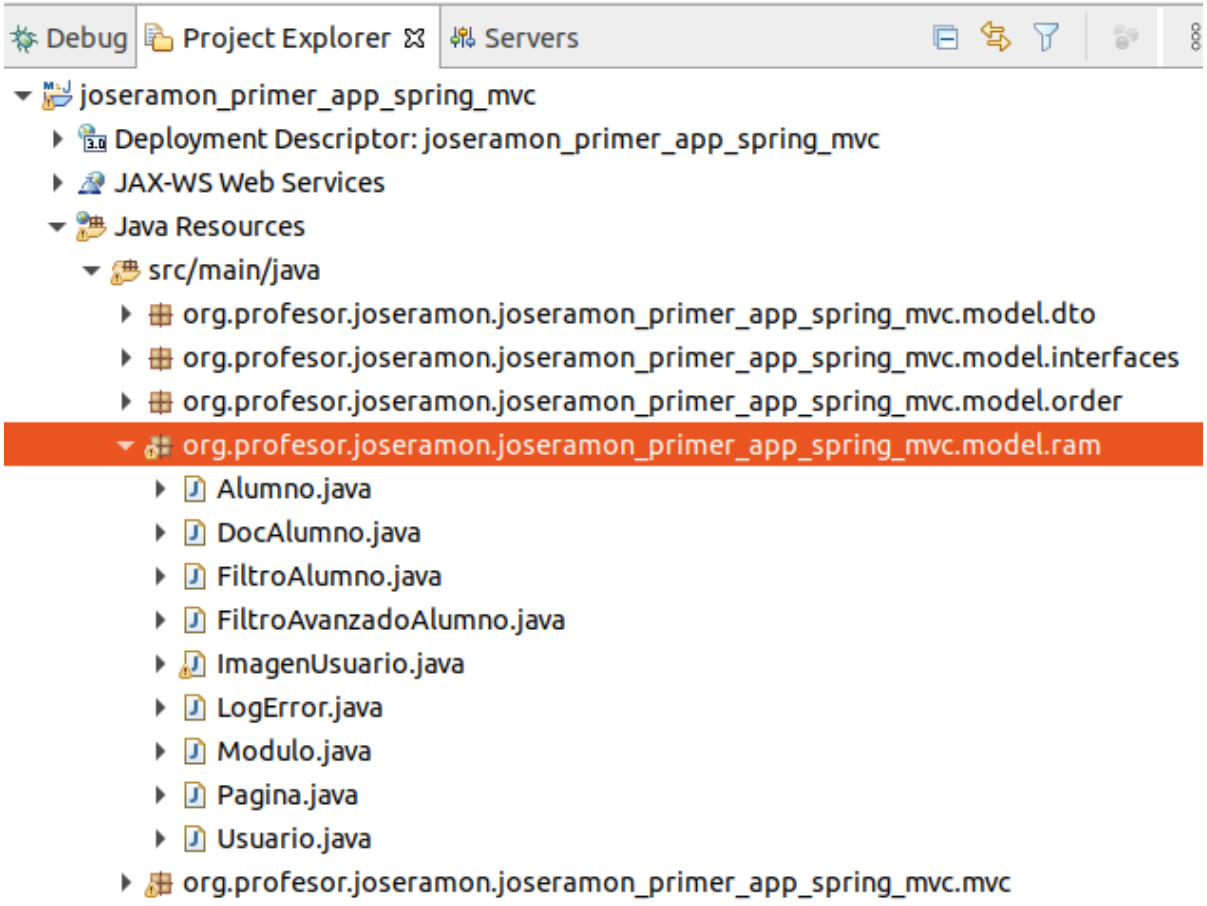
UD 2: Modelo Vista Controlador
12.- DTOs

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

Antes de continuar vamos a refactorizar nuestra webApp nuevamente para prepararla para la incorporación inminente de nuevas clases que harán referencia a las tablas de una Base de datos en la unidad didáctica que trata el acceso a BDs.

Para ello creamos un subpaquete “ram” dentro del paquete “model” y movemos todas las clases de la clase model a “ram” permitiendo a Eclipse que modifique las referencias a dichas clases en el resto del proyecto.

Hacemos esto porque en breve dentro de “model” aparte de los subpaquetes “dto” y “ram” tendremos el subpaquete “db”.



UD 2: Modelo Vista Controlador
12.- DTOs

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

Volviendo al tema de los DTOs, nos hemos evitado uno de los problemas iniciales de los que hablábamos al principio del PDF, tener que pasar toda la información de Alumno a la vista del listado de alumnos, pero...

¿Como lo utilizamos para mejorar el diseño de nuestra webApp para que al modificar alumno no tengamos que preocuparnos de controlar si machacamos los documentos? ¿Porque tenemos este problema?

AlumnoController.java

```
239    @RequestMapping(value = "/update-alumno", method = RequestMethod.POST)
240    public String procesaUpdateAlumno(ModelMap model, @Valid Alumno alumno, BindingResult validacion) {
241        model.addAttribute("pagina", pagina);
242        if (validacion.hasErrors()) {
243            // Hay errores y debemos volver al formulario de modificación
244            return "update-alumno";
245        }
246        // Si llega aquí no hay errores de validación
247        try {
248            if (model.getAttribute("usuario") == null) {
249                throw new Exception("Para modificar debe estar logeado");
250            }
251            Usuario usuarioActivo = (Usuario) model.getAttribute("usuario");
252            alumnoService.modificaAlumno(alumno, usuarioActivo.getNickname());
253            // Para evitar pasar parámetros inncesarios
254            model.clear();
255            /* Para evitar volver a modificar redirigimos a listar */
256            return "redirect:list-alumno";
257        } catch (Exception e) {
258            // Le pasamos el alumno actualizado
259            model.addAttribute(alumnoService.encontrarAlumnoPorDni(alumno.getD
260            // Pasamos los errores
261            model.addAttribute("errores", e.getMessage());
262            // Hay errores y debemos volver al formulario de modificación
263            return "update-alumno";
264        }
265    }
```

AlumnoService.java

```
129    public void modificaAlumno(Alumno alumnoModificado,String usuarioModificacion) throws Exception {
130        String errores="";
131        if (alumnoModificado==null) {
132            errores="No se ha podido actualizar el alumno porque no han llegado los datos modificados";
133        } else {
134            Alumno alumnoActual=encontrarAlumnoPorDni(alumnoModificado.getDni());
135            if (alumnoActual.sePuedeModificarUtilizando(alumnoModificado)) {
136                alumnos.remove(alumnoActual);
137                //Debemos de asegurarnos que no se borra la documentación
138                alumnoModificado.setDocsAlumno(alumnoActual.getDocsAlumno());
139                //actualizamos usuario y fecha modificación
140                alumnoModificado.setUser(usuarioModificacion);
141                alumnoModificado.setTs(new Date());
142                alumnos.add(alumnoModificado);
143            } else {
144                errores=alumnoActual.mensajeNoSePuedeModificar();
145            }
146        }
147        if (errores.length()>0) {
148            throw new Exception(errores);
149        }
150    }
```

Este problema ocurre porque en el formulario de modificación no están todos los atributos del alumno, ya que no se pasa la lista de documentos. Por eso los pasamos en el servicio al modificar (para que no se borren) y esto es una “pequeña chapuza”.

AlumnoService.java

```
128
129    public void modificaAlumno(Alumno alumnoModificado,String usuarioModificacion) throws Exception {
130        String errores="";
131        if (alumnoModificado==null) {
132            errores="No se ha podido actualizar el alumno porque no han llegado los datos modificados";
133        } else {
134            Alumno alumnoActual=encontrarAlumnoPorDni(alumnoModificado.getDni());
135            if (alumnoActual.sePuedeModificarUtilizando(alumnoModificado)) {
136                alumnos.remove(alumnoActual);
137                //Debemos de asegurarnos que no se borra la documentación
138                alumnoModificado.setDocsAlumno(alumnoActual.getDocsAlumno());
139                //actualizamos usuario y fecha modificación
140                alumnoModificado.setUser(usuarioModificacion);
141                alumnoModificado.setTs(new Date());
142                alumnos.add(alumnoModificado);
143            } else {
144                errores=alumnoActual.mensajeNoSePuedeModificar();
145            }
146        }
147        if (errores.length()>0) {
148            throw new Exception(errores);
149        }
150    }
```

Para solucionar este error y futuros lo que debemos hacer es **leer la información actual que tenemos de ese Alumno, y sobre ese registro mapearle/modificar SOLO los datos que tenemos en la pantalla de la vista**. De esta manera no hace falta que nos preocupemos de si en el formulario de modificación están todos los datos o no para copiarle los datos originales que no estuvieran en la vista.

¿Como lo implementamos con MapStruct?

UD 2: Modelo Vista Controlador
12.- DTOs

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

AlumnoController.java

```
208
209
210     @RequestMapping(value = "/update-alumno", method = RequestMethod.POST)
211     public String procesaUpdateAlumno(ModelMap model, @Valid AlumnoEdit alumnoEdit, BindingResult validacion) {
212         model.addAttribute("pagina", pagina);
213         if (validacion.hasErrors()) {
214             // Hay errores y debemos volver al formulario de modificación
215             return "update-alumno";
216         }
217         // Si llega aquí no hay errores de validación
218         try {
219             if (model.getAttribute("usuario") == null) {
220                 throw new Exception("Para modificar debe estar logeado");
221             }
222             Usuario usuarioActivo = (Usuario) model.getAttribute("usuario");
223             alumnoService.modificaAlumno(alumnoEdit, usuarioActivo.getNickname());
224             // Para evitar pasar parámetros inncesarios
225             model.clear();
226             /* Para evitar volver a modificar redirigimos a listar */
227             return "redirect:list-alumno";
228         } catch (Exception e) {
229             // Le pasamos el alumno actualizado
230             model.addAttribute(alumnoService.encontrarAlumnoEditPorDni(alumnoEdit.getDni()));
231             // Pasamos los errores
232             model.addAttribute("errores", e.getMessage());
233             // Hay errores y debemos volver al fo
234             return "update-alumno";
235         }
236     }
```

AlumnoService.java

```
166
167     public void modificaAlumno(AlumnoEdit alumnoEditModificado,String usuarioModificacion) throws Exception {
168         String errores="";
169         if (alumnoEditModificado==null) {
170             errores="No se ha podido actualizar el alumno porque no han llegado los datos modificados";
171         } else {
172             Alumno alumno=encontrarAlumnoPorDni(alumnoEditModificado.getDni());
173             AlumnoEdit alumnoEditActual= AlumnoMapper.INSTANCE.alumnoToAlumnoEdit(alumno);
174             if (alumnoEditActual.sePuedeModificarUtilizando(alumnoEditModificado)) {
175                 //actualizamos usuario y fecha modificación antes de mapear sobre Alumno
176                 alumnoEditModificado.setUser(usuarioModificacion);
177                 alumnoEditModificado.setTs(new Date());
178                 //Mapeamos los datos del formulario de modificación('AlumnoEdit') al Alumno
179                 AlumnoMapper.INSTANCE.updateAlumnoFromAlumnoEdit(alumnoEditModificado,alumno);
180             } else {
181                 errores=alumno.mensajeNoSePuedeModificar();
182             }
183         }
184         if (errores.length()>0) {
185             throw new Exception(errores);
186         }
187     }
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

Modifica la inserción de nuevos documentos de alumnos para que el controller, la vista y el service utilicen Dtos. Una posible solución sería la que se muestra a continuación en las siguientes páginas:

AlumnoService.java

```
127     public List<DocAlumnoList> encontrarDocsAlumnoListPorDni(String dni) {
128         ArrayList<DocAlumnoList> resultado = new ArrayList<DocAlumnoList>();
129         Alumno alumno = encontrarAlumnoPorDni(dni);
130         if (alumno == null) // No existe alumno-> lista vacia
131             return resultado;
132         else {// existe alumno
133             if (alumno.getDocsAlumno() == null)// No tiene documentos -> lista vacia
134                 return resultado;
135             else { // Hay documentos
136                 return AlumnoMapper.INSTANCE.docsAlumnoToDocsAlumnoList(alumno.getDocsAlumno());
137             }
138         }
139     }
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

AlumnoService.java

```
189    public void addDocAlumnoEdit(DocAlumnoEdit docAlumnoEdit, String usuarioModificacion) throws Exception {
190        if (docAlumnoEdit == null) {
191            throw new Exception("No se ha podido actualizar el documento porque no han llegado los datos modificados");
192        } else {
193            String dni = (String) docAlumnoEdit.getDni();
194            Alumno alumno = encontrarAlumnoPorDni(dni);
195            if (alumno == null) throw new Exception("Alumno desconocido");
196            if (usuarioModificacion == null) throw new Exception("Para añadir documentación debe estar logeado");
197
198            // Guardar fichero en el Sistema Operativo:
199            // 1º Componer nombre del fichero
200            String extension = fileServicio.getExtensionMultipartfile(docAlumnoEdit.getFichero());
201            String nombreFicheroAGuardar = String.format("%s_idDoc_%s.%s", dni, docAlumnoEdit.getId(), extension);
202            // 2º Guardar fichero en la carpeta:Si no se ha podido listaErroresAlGuardar no
203            // estará vacio
204            ArrayList<String> listaErroresAlGuardar = fileServicio.guardaDocumentacionAlumno(docAlumnoEdit.getFichero(),
205                    nombreFicheroAGuardar);
206            if (!listaErroresAlGuardar.isEmpty()) {// Rellenar los errores al intentar guardar para pasarselos a la
207                                                   // excepcion
208                String mensajeCompleto = "";
209                for (String mensaje : listaErroresAlGuardar)
210                    mensajeCompleto += i18nService.getTraduccion(mensaje) + "<br>";
211                // lanzar excepción
212                throw new Exception(mensajeCompleto);
213            }
214            // completar tipo de documento del alumno
215            docAlumnoEdit.setTipoFichero(extension);
216            docAlumnoEdit.setContentTypeFichero(docAlumnoEdit.getFichero().getContentType());
217            // 3º Añadir el nuevo documento al alumno :
218            // Añadir el docAlumno
219            DocAlumno docAlumno = AlumnoMapper.INSTANCE.docAlumnoEditToDocAlumno(docAlumnoEdit);
220            if (alumno.getDocsAlumno() != null)// lista de documentos no vacia
221                alumno.getDocsAlumno().add(docAlumno);
222            else {// Lista de documentos vacia-> 1º elemento el docAlumno actual
223                alumno.setDocsAlumno(new ArrayList<DocAlumno>(List.of(docAlumno)));
224            }
225            // 4º Al modificar el alumno (su lista de docs) debemos actualizar la fecha y
226            // usuario
227            modificaAlumno(AlumnoMapper.INSTANCE.alumnoToAlumnoEdit(alumno), usuarioModificacion);
228        }
229    }
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

AlumnoController.java

```
152    @RequestMapping(value = "/docs-alumno", method = RequestMethod.GET)
153    public String getDocsAlumnos(ModelMap model, @RequestParam String dni) {
154
155        pagina.setIdioma(i18nService.getIdioma());
156        model.addAttribute("alumnoInfo", alumnoService.encontrarAlumnoInfoPorDni(dni));
157        model.addAttribute("docAlumnoEdit",new DocAlumnoEdit(alumnoService.siguienteDoc(dni)));
158        model.addAttribute("docsAlumnoList",alumnoService.encontrarDocsAlumnoListPorDni(dni));
159        model.addAttribute("pagina", pagina);
160        return "docs-alumno";
161    }
162
163    @RequestMapping(value = "/add-docAlumno", method = RequestMethod.POST)
164    public String addDocAlumno(ModelMap model, @Valid DocAlumnoEdit docAlumnoEdit, BindingResult validacion) {
165        String dni = "";
166        if (docAlumnoEdit != null)
167            dni = docAlumnoEdit.getDni();
168        pagina.setIdioma(i18nService.getIdioma());
169        model.addAttribute("pagina", pagina);
170        if (validacion.hasErrors()) {
171            // Hay errores y debemos volver al formulario
172            // Si no añadimos alumno, la cabecera de los datos del alumno no se imprimiran
173            model.addAttribute("alumnoInfo", alumnoService.encontrarAlumnoInfoPorDni(dni));
174            model.addAttribute("docsAlumnoList",alumnoService.encontrarDocsAlumnoListPorDni(dni));
175            return "docs-alumno";
176        }
177        // Si llega aquí no hay errores de validación
178        try {
179            Usuario usuarioActivo = (Usuario) model.getAttribute("usuario");
180            alumnoService.addDocAlumnoEdit(docAlumnoEdit, usuarioActivo.getNickname());
181            // Rellenamos el modelo para la respuesta:
182            // Si no añadimos alumno, la cabecera de los datos del alumno no se imprimiran
183            model.addAttribute("alumnoInfo", alumnoService.encontrarAlumnoInfoPorDni(dni));
184            // Al igual que en el GET, debemos crear un doc vacio con el siguiente id
185            model.addAttribute("docAlumnoEdit", new DocAlumnoEdit(alumnoService.siguienteDoc(dni)));
186            model.addAttribute("docsAlumnoList",alumnoService.encontrarDocsAlumnoListPorDni(dni));
187            // Para evitar pasar parámetros inncesarios
188            return "docs-alumno";
189        } catch (Exception e) {
190            // Le pasamos el alumno actualizado
191            model.addAttribute(alumnoService.encontrarAlumnoInfoPorDni(dni));
192            model.addAttribute("docsAlumnoList",alumnoService.encontrarDocsAlumnoListPorDni(dni));
193            // Pasamos los errores
194            model.addAttribute("errores", e.getMessage());
195            // Hay errores y debemos volver al formulario de modificación
196            return "docs-alumno";
197        }
198    }
```

UD 2: Modelo Vista Controlador
12.- DTOs

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

Veamos otra funcionalidad de MapStruct, modifica la web para que muestre la fecha de modificación del alumno teniendo en cuenta el siguiente mapeo:

DWES Home Alumnos Modulos Errores joseramon

Listado de alumnos:

Dni: Ninguno Ciclo: Ninguno Horario: Ninguno Filtrar

Dni	Nombre	Edad	Ciclo	Curso	Erasmus	Modificado	Acción
11111111A	Jose	21	DAM	1	☐	15/01/21 12:25:32	Modificar Borrar Documentación
22222222B	Pedro	32	DAW	2	☐	13/01/21 10:40:32	Modificar Borrar Documentación
33333333C	Juan	23	ASIR	1	☐	17/01/21 11:10:32	Modificar Borrar Documentación

Nuevo

AlumnoMapper.java

```
18 public interface AlumnoMapper {
19     AlumnoMapper INSTANCE= Mappers.getMapper(AlumnoMapper.class);
20
21     //Devuelve un objeto de tipo 'AlumnoEdit' a partir de un objeto de tipo 'Alumno'
22     AlumnoEdit alumnoToAlumnoEdit(Alumno alumno);
23     //Devuelve un objeto de tipo 'AlumnoList' a partir de un objeto de tipo 'Alumno'
24     @Mapping(target="modificado",source = "alumno.ts", dateFormat = "dd/MM/yy HH:mm:ss")
25     AlumnoList alumnoToAlumnoList(Alumno alumno);
```

EJERCICIO:

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_practica10_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.