Objetivos de la sesión:

- *Poder ordenar los datos por más de un criterio pulsando en el nombre de la columna (Dni, Nombre, Edad, Ciclo o Curso) utilizando **las interfaces Comparable y Comparator**.*

Una de las mejoras que podemos aplicar a nuestro proyecto es ofrecer la posibilidad de **mostrar los datos ordenados por diferentes criterios**.

¿Como podemos realizar esta ordenación con distintos criterios?
Y sobretodo ¿Donde lo implementamos y como?

Interfaces:

· El sistema que pretendemos implementar consiste en ***ordenar los datos gracias a las interfaces Comparable y Comparator. El usuario desde pantalla indicará el sistema de ordenación y el servlet recogerá dicho parámetro para pasarselo al servicio que se encargará de devolver la lista de alumnos ordenados por dicho criterio para que la página jsp lo muestre.***

Si recuerdas como utilizar las clases Comparable y Comparator esta práctica será muy rápida!!

¿Que recuerdas de las Interfaces y más concretamente de Comparable y Comparator? ¿Quieres que repasemos los conceptos?

... continuación Interfaces:

Los **métodos de cualquier interfaz** están SIEMPRE vacíos de manera similar a las clases abstractas. Es decir, **tienen la cabecera, pero no están implementados**.

Veamos como sería la implementación de una interfaz:

```
public interface Volador{

  int alturaMax=100;

  void subir();

  void bajar();

}
```

En una interfaz también podemos definir **constantes** y siempre **serán final y static aunque no hace falta indicarselo**.

Los métodos no se implementan aquí, sino que los habrán de implementar cada clase que quiera implementar la interfaz Volador. **Aunque no se ponga, los métodos son automáticamente abstract i public** .

... continuación Interfaces:

- Un **método abstracto** es un método donde no consta la implementación.
- Una **clase abstracta** es una clase que tiene algún método abstracto. No se puede instanciar.
- Las **clases hijas de una clase abstracta** están obligadas a implementar esos métodos abstractos, o bien volver a declararlos como abstractos (y por tanto esa clase también sería abstracta).

Una **Interfaz es como una clase lo más abstracta posible** (no tiene implementado ningún método) con la **ventaja de que una clase puede implementar ("ser hija de") muchas interfaces** (pero solo extender/heredar de un clase).

Por ejemplo , En un videojuego 'Vehiculo' solo puede extender de una clase (por ejemplo 'Jugador') pero puede implementar varias interfaces (por ejemplo 'Volador' y 'Disparador').

NOTA: Una Interfaz también pueden extender de otra interfaz.

... continuación Interfaces:

Por tanto una interfaz es un **conjunto de métodos sin implementar que habrán de implementar aquellas clases que quieran comportarse así**. También pueden incluir constantes.

Al ser clases abstractas en las que todos sus métodos les falta la implementación se siguen cumpliendo la **relación ES-UN** de las clases abstractas. Si la clase Vehiculo implementa la Interfaz Volador es porque **Vehiculo ES-UN Volador**. A su vez si la clase Vehiculo también implementa la interfaz Disparador es porque **Vehiculo ES-UN Disparador**.

Las interfaces son compatibles con el operador **instanceof** para determinar si una clase es compatible en cuanto al tipo con una interfaz. **Por ejemplo podemos tener un método que se pueda aplicar solo a Voladores y podemos comprobar antes si el objeto que se suministra es una clase compatible con los Voladores (implementa la interfaz).**

Una clase que implementa un método de interfaz **no puede reducir la visibilidad**. Esto quiere decir que si son métodos públicos , todas las implementaciones tienen que ser públicas. Igual para con las excepciones, si la clase implementa un método con una lista throws no puede añadir un tipo de excepción nuevo. Por tanto al implementar el método no incluirá la lista throws.

... continuación Interfaces:

Las interfaces sirven principalmente para **proporcionar un comportamiento o funcionalidad añadida a una clase** y también para :

- Permitir **simular la herencia múltiple** (ya que una clase solo puede tener una superclase pero puede implementar muchas interfaces).

  Es una solución elegante porque **las interfaces solucionan el problema de la herencia múltiple conflictiva** que existe en otros lenguajes cuando una clase puede heredar de varias clases, ya que ambas clases padre podrían tener el mismo método definido y no sabriamos cual ejecutar. Con las interfaces este problema se soluciona porqué se debe proporcionar la implementación a dicho método.

- Obligar a que ciertas clases utilicen los **mismos métodos** (nombres y parámetros) **sin** estar obligados a tener una relación de **herencia**. Si tenemos una interfaz 'Figura' con un método 'perimetro()' y tenemos que Cuadrado y Circulo implementan 'Figura' aunque se calcule de forma diferente llamaremos al método de la misma forma y podemos tener un método que le pasemos como parámetro una 'Figura' y haga algo utilizando el método común 'perimetro()'.

- Sabiendo que una clase implementa una determinada interfaz, podremos **utilizar los métodos porque ya sabemos que hacen** (nos da igual como estén implementados).

- Definir un cojunto de **constantes**.

Interfaces vs Clases:

	Clases	Interfaces
Cantidad de "padres" de una clase	Una clase **solo puede extender (ser hija) de una sola clase**. Pe: class Ave extends Animal	Una clase puede **implementar muchas interfaces**. Pe: class Vehiculo implements Volador,Disparador

¿Pueden definirse objetos de ese tipo?	**Si.**
	Pe :
	Vehiculo v1;
	v1= new Vehiculo();
		Si, pero no se pueden instanciar
		(igual que las clases abstractas).
		Pe:
		Volador ovni;
		ovni = ~~new Volador();~~
		ovni = new Vehiculo(); //new Ave();
		//Tambien aplicado a ArrayList
		ArrayList<Volador> voladores= new ArrayList<>();
		Voladores.add(new Vehiculo());
		Voladores.add(new Ave());

Interfaces vs Clases Abstractas:

	Clases Abstractas	Interfaces
¿Pueden tener métodos en el cuerpo?	**Si**	**No** (en versión 8 de JDK si, si ponemos delante "default")
¿Se pueden definir atributos?	**Si**	**Solo constantes**

```java
public interface Disparador {
  void disparar();
  void defender();

  default void probarConCuerpo() {
    System.out.println("Hola");
  }
}
```

Interfaz Comparable y Comparator:

Dependiendo de la estructura de datos podemos llegar a **ordenar los valores de una lista automáticamente:**

- **Array.sort(nombreArray)** para vectores.

  Por ejemplo: arrayEdades

- **Collections.sort(nombreArrayList)** para ArrayList.

  Por ejemplo: arraylistEdades

```java
package ejemplosInterfaces;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class EjemploOrdenacion {
    Run|Debug
    public static void main(String[] args) {

        int[] arrayEdades = { 4, 7, 3, 6, 9 };
        Arrays.sort(arrayEdades);
        for (int e : arrayEdades) {
            System.out.print(e + " ");
        }

        ArrayList<Integer> arraylistEdades = new ArrayList<>();
        arraylistEdades.add(4);
        arraylistEdades.add(7);
        arraylistEdades.add(3);
        arraylistEdades.add(6);
        arraylistEdades.add(9);
        Collections.sort(arraylistEdades);
        System.out.println(arraylistEdades);

    }

}
```

UD 1: Introducción a los lenguajes de servidor
4.- JEE: Servlets, JSP y JSTL

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

... continuación Interfaz Comparable y Comparator:

¿Que pasa si intentamos hacerlo con la clase Alumno? ¿Podemos ordenar un array de Alumnos?¿Podemos ordenar un ArrayList de Alumnos? ¿Porque?

```
Exception in thread "main" java.lang.ClassCastException: class ejemplosInterfaces.AlumnoNoComparable cannot be cast to class java.lang.Comparable
(ejemplosInterfaces.AlumnoNoComparable is in unnamed module of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')
        at java.base/java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
        at java.base/java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
        at java.base/java.util.Arrays.sort(Arrays.java:1249)
        at ejemplosInterfaces.ComparaArrayAlumnosMal.main(ComparaArrayAlumnosMal.java:10)
```

```java
package ejemplosInterfaces;
public class AlumnoNoComparable {
    private int id;
    private String nombre;
    private int edad;
    private String curso;

    public AlumnoNoComparable(int id, String nombre, int edad, String curso) {
        this.id = id;
        this.nombre = nombre;
        this.edad = edad;
        this.curso = curso;
    }
    public int getId() {
        return id;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public String getCurso() {
        return curso;
    }
}
```

```java
package ejemplosInterfaces;

import java.util.Arrays;

public class ComparaArrayAlumnosMal {
    Run | Debug
    public static void main(String[] args) {
        AlumnoNoComparable alNoComp1 = new AlumnoNoComparable(11,"No Comparable1",12,"1ESO");
        AlumnoNoComparable alNoComp2 = new AlumnoNoComparable(2,"No Comparable2",13,"2ESO");
        AlumnoNoComparable[] arrayAlumnos = { alNoComp1,alNoComp2};
        Arrays.sort(arrayAlumnos);
        for (AlumnoNoComparable e : arrayAlumnos) {
            System.out.print(e + " ");
        }
    }
}
```

... continuación Interfaz Comparable y Comparator:

En breve sabremos porque no funciona. De momento si probamos lo mismo sobre una ArrayList de Alumnos nisiquera compila:

```java
package ejemplosInterfaces;
import java.util.ArrayList;
import java.util.Collections;

public class ComparaArrayListAlumnosMal {
    Run|Debug
    public static void main(String[] args) {
        AlumnoNoComparable alNoComp1 = new AlumnoNoComparable(11,"No Comparable1",12,"1ESO");
        AlumnoNoComparable alNoComp2 = new AlumnoNoComparable(2,"No Comparable2",13,"2ESO");
        ArrayList<AlumnoNoComparable> arraylistAlumnos = new ArrayList<>();
        arraylistAlumnos.add(alNoComp1);
        arraylistAlumnos.add(alNoComp2);
        Collections.sort(arraylistAlumnos);
        System.out.println(arraylistAlumnos);
    }
}
```

```
<T> void java.util.Collections.sort(List<T> list)
The method sort(List<T>) in the type Collections is not applicable for the arguments
(ArrayList<AlumnoNoComparable>) Java(67108979)
Peek Problem (Alt+F8)   Quick Fix... (Ctrl+.)
```

Si les aplicamos el método **sort a elementos que no son directamente ordenables (comparables)** como por ejemplo una lista de personas, coches, alumnos, nos da error, porque **la MVJ "no sabe comparar" esos objetos**.

¿Como lo solucionamos para que se puedan comparar/ordenar Alumnos?

... continuación Interfaz Comparable y Comparator:

Necesimos utilizar la **Interfaz Comparable** que nos permite indicar un **criterio de comparación que indique como ordenar objetos.** Es decir, definir cuando un objeto de la clase es menor que otro, es mayor o igual. En nuestro ejemplo compararemos por el campo “id”.

Este criterio de comparación se consigue implementando el **método**:

int compareTo(Object obj)

Este método retorna un número negativo, uno cero o un número positivo dependiendo de si el objeto es menor , igual o mayor a obj.

```java
package ejemplosInterfaces;
public class AlumnoSinCasting implements Comparable {
    private int id;
    private String nombre;
    private int edad;
    private String curso;

    public AlumnoSinCasting(int id, String nombre, int edad, String curso) {
        this.id = id;
        this.nombre = nombre;
        this.edad = edad;
        this.curso = curso;
    }
    @Override
    public int compareTo(Object obj) {
        //Como no le hemos especificado el tipo en
        //Comparable es obligado hacer el casting
        return id-((AlumnoSinCasting) obj).getId();
    }
    public int getId() {
        return id;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public String getCurso() {
        return curso;
    }
}
```

o mejor aun → damos el tipo para evitar el casting obligado

```java
package ejemplosInterfaces;
public class Alumno implements Comparable<Alumno> {
    private int id;
    private String nombre;
    private int edad;
    private String curso;

    public Alumno(int id, String nombre, int edad, String curso) {
        this.id = id;
        this.nombre = nombre;
        this.edad = edad;
        this.curso = curso;
    }
    @Override
    public int compareTo(Alumno alumno) {
        return id-alumno.getId();
    }
    public int getId() {
        return id;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public String getCurso() {
        return curso;
    }
}
```

... continuación Interfaz Comparable y Comparator:

Ahora ya podemos **comparar 2 objetos de la clase** Alumno y/o **ordenar una lista** de alumnos (array, ArrayList…) **con el sort**: (Que es nuestro objetivo en la webapp...)

```java
package ejemplosInterfaces;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class ComparaAlumnos {
    Run | Debug
    public static void main(String[] args) {
        Alumno alumno1 = new Alumno(11,"Comparable11",12,"1ESO");
        Alumno alumno2 = new Alumno(2,"Comparable2",13,"2ESO");

        //Ahora podemos comparar por tener la interfaz Comparable
        if (alumno1.compareTo(alumno2)>0){
            System.out.println("Alumno '" +alumno1.getNombre()
            + "' es mayor al Alumno ''" +alumno2.getNombre()+"'");
        }

        //Array de Alumnos
        Alumno[] arrayAlumnos = { alumno1,alumno2};
        Arrays.sort(arrayAlumnos); //Podemos ordenar
        System.out.println("Array de Alumnos Ordenados:");
        for (Alumno a : arrayAlumnos) {
            System.out.println(a + " ");
        }
        //ArrayList de alumnos
        System.out.println("ArrayList de Alumnos ordenados:");
        ArrayList<Alumno> arraylistAlumnos = new ArrayList<>();
        arraylistAlumnos.add(alumno1);
        arraylistAlumnos.add(alumno2);
        Collections.sort(arraylistAlumnos); //Podemos ordenar
        System.out.println(arraylistAlumnos);
        // o bien si queremos darle mejor aspecto
        System.out.println("Recorremos ArrayList ordenado:");
        for (Alumno a : arraylistAlumnos) {
            System.out.println(a + " ");
        }
    }
}
```

```
Alumno 'Comparable11' es mayor al Alumno ''Comparable2'
Array de Alumnos Ordenados:
Alumno [id=2, nombre=Comparable2, edad=13, curso=2ESO];
Alumno [id=11, nombre=Comparable11, edad=12, curso=1ESO];
ArrayList de Alumnos ordenados:
[Alumno [id=2, nombre=Comparable2, edad=13, curso=2ESO]; , Alumno [id=11, nombre=Comparable11, edad=12, c
urso=1ESO]; ]
Recorremos ArrayList ordenado:
Alumno [id=2, nombre=Comparable2, edad=13, curso=2ESO];
Alumno [id=11, nombre=Comparable11, edad=12, curso=1ESO];
```

... continuación Interfaz Comparable y Comparator:

Otro motivo para usar **interfaces** es que mediante la implementación de interfaces todos los programadores pueden utilizar el **mismo nombre de método y estructura formal** para comparar objetos (o clonarlos, u otras operaciones).

Imaginate que estás trabajando con un equipo de programadores y has de utilizar una clase que ha codificado otro programador. Si quieres comparar 2 objetos de esa clase, solo viendo que implementa la interfaz Comparable, ya sabes que métodos puedes usar sin saber como está implementado.

Utilizar el mismo nombre de métodos para hacer las mismas tareas facilita el desarrollo de programas y ayuda a comprenderlos, sobretodo cuando intervienen centenares de clase distintas.

... continuación Interfaz Comparable y Comparator:

Si queremos establecer **un único criterio** de ordenación usamos la **interfaz Comparable**, pero si queremos establecer **diferentes criterios** de ordenación usaremos **Comparator**.

Para poder **ordenar por N criterios** diferentes (que es lo que haremos en nuestra webapp) crearemos **N clases especiales que implementen la interfaz Comparator**, una por cada criterio de ordenación donde sobreescribiremos el **método compare**.

Veamos algunos **ejemplos**:

Criterio 1: Queremos ordenar por nombre:

```java
package ejemplosInterfaces;
import java.util.Comparator;

public class ComparadorAlumnoNom implements Comparator<Alumno> {
    @Override //Comparamos por nombre
    public int compare(Alumno a1,Alumno a2) {
        return a1.getNombre().compareTo(a2.getNombre());
    }
}
```

Criterio 2: Ordenamos por curso y a igualdad de curso miramos la edad:

```java
package ejemplosInterfaces;
import java.util.Comparator;

public class ComparadorAlumnoCursoEdad implements Comparator<Alumno> {
    @Override //Comparamos primero por Curso y después por edad
    public int compare(Alumno a1,Alumno a2) {
        int comparaCurso=a1.getCurso().compareTo(a2.getCurso());
        if (comparaCurso!=0) // Si son diferentes cursos ya tenemos orden
            return comparaCurso;
        else // Cursos iguales, mirar edad
            return a1.getEdad()-a2.getEdad();
    }
}
```

UD 1: Introducción a los lenguajes de servidor
4.- JEE: Servlets, JSP y JSTL

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

… continuación Interfaz Comparable y Comparator:

```java
package ejemplosInterfaces;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
public class ComparadorAlumnosApp {
    Run | Debug
    public static void main(String[] args) {
        Scanner leer= new Scanner(System.in);
        Alumno alumno1 = new Alumno(1,"Comparable41",13,"1ESO");
        Alumno alumno2 = new Alumno(2,"Comparable22",13,"2ESO");
        Alumno alumno3 = new Alumno(3,"Comparable13",12,"1ESO");
        //COMPARAR:Podemos comparar con la clase Comparator que queramos
        if (new ComparadorAlumnoCursoEdad().compare(alumno1, alumno3)>0){
            System.out.println("Alumno '" +alumno1.getNombre()+ "' es mayor"
            +" al Alumno '" +alumno3.getNombre()+"' por curso y edad");
        }
        //ORDENAR:
        // Alumno[] arrayAlumnos:
        //Con un Array de Alumnos si ordenamos por el compareTo
        //solo podemos tener un criterio de comparación/ordenación

        //Para poder tener MAS DE UN CRITERIO DE ORDENACIÓN
        //necesitamos un ARRAYLIST utilizando COLLECTIONS.SORT
        ArrayList<Alumno> arraylistAlumnos = new ArrayList<>();
        arraylistAlumnos.add(alumno1);
        arraylistAlumnos.add(alumno2);
        arraylistAlumnos.add(alumno3);
        System.out.println("¿En que orden quieres la lista?");
        System.out.println("(1=Nombre; 2=Curso y edad)");
        switch (leer.nextInt()) {
            case 1://ordenamos por nombre
                Collections.sort(arraylistAlumnos,new ComparadorAlumnoNombre());
                break;
            case 2://ordenamos por curso y edad
                Collections.sort(arraylistAlumnos,new ComparadorAlumnoCursoEdad());
                break;
        }
        //Mostrar resultado
        System.out.println("Recorremos ArrayList ordenado:");
        for (Alumno a : arraylistAlumnos) {
            System.out.println(a + " ");
        }
        //Cerrar Scanner
        leer.close();
    }
}
```

1º Ejecución: Orden por Nombre:

```
Alumno 'Comparable41' es mayor al Alumno 'Comparable13' por curso y edad
¿En que orden quieres la lista?
(1=Nombre; 2=Curso y edad)
1
Recorremos ArrayList ordenado:
Alumno [id=3, nombre=Comparable13, edad=12, curso=1ESO];
Alumno [id=2, nombre=Comparable22, edad=13, curso=2ESO];
Alumno [id=1, nombre=Comparable41, edad=13, curso=1ESO];
```

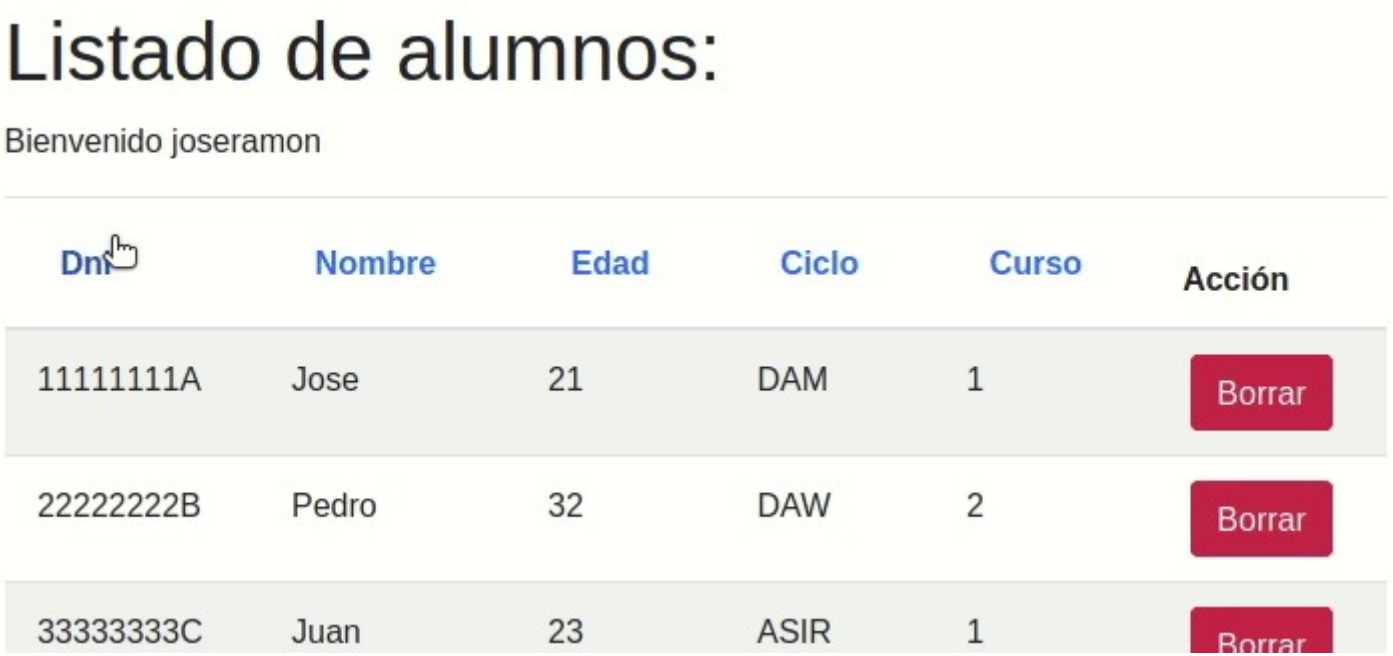
2º Ejecución: Orden por Curso y Edad:

```
Alumno 'Comparable41' es mayor al Alumno 'Comparable13' por curso y edad
¿En que orden quieres la lista?
(1=Nombre; 2=Curso y edad)
2
Recorremos ArrayList ordenado:
Alumno [id=3, nombre=Comparable13, edad=12, curso=1ESO];
Alumno [id=1, nombre=Comparable41, edad=13, curso=1ESO];
Alumno [id=2, nombre=Comparable22, edad=13, curso=2ESO];
```

Fijate porque en nuestra webapp lo tendrás que utilizar…

... continuación Interfaces:

Vamos a hacer que Alumno se ordene por cualquiera de las columnas pulsando sobre el encabezado en azul del nombre del atributo:



Puedes ver una animación explicativa de que debería de hacer la aplicación en el moodle en el ejercicio "Interfaces y ordenación".

... continuación Interfaces:

Inicialmente vamos a hacer que Alumno sea comparable para tener un criterio de ordenación por defecto, en nuestro caso el nombre.

Adicionalmente habrá que crear tantos comparadores como criterios de ordenación. Estos comparadores serán:

- *Ordenar por Dni*
- *Ordenar por edad y a igualdad de edad por nombre*
- *Ordenar por ciclo y a igualdad de ciclo por nombre*
- *Ordenar por curso y a igualdad de curso por nombre*

... continuación Interfaces:

El servicio deberá de tener un método nuevo:

public List<Alumno> listaAlumnos(String criterioOrdenacion)

Si no se especifica criterio la ordenación por defecto será por nombre.

En el jsp cuando se pulse sobre el nombre de la columna deberá de llamar al listado ordenado por uno de los criterios anteriores. Por ejemplo, si se pulsa sobre el nombre de columna "edad" deberemos de ordenar por edad y a igualdad de edad por el nombre.

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 ${nombre}</p>
7          <table class="table table-striped">
8              <thead>
9                  <th> <a  class="nav-link" href="list-alumno.do?orden=dni">Dni</a></th>
10                 <th> <a  class="nav-link" href="list-alumno.do?orden=nombre">Nombre</a></th>
11                 <th> <a  class="nav-link" href="list-alumno.do?orden=edadNombre">Edad</a></th>
12                 <th> <a  class="nav-link" href="list-alumno.do?orden=cicloNombre">Ciclo</a></th>
13                 <th> <a  class="nav-link" href="list-alumno.do?orden=cursoNombre">Curso</a></th>
14                 <th> Acción</th>
15             </thead>
16             <tbody>
17             <c:forEach items="${alumnos}" var="alumno">
```

EJERCICIO:

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

Para ello:

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

2º Comprime la carpeta de tu aplicación y ponle como nombre al fichero comprimido UD1_practica9_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.