

Mo6

ALVARO ORTEGA MARMOL

ACCESO A DATOS

ÍNDICE GENERAL



TEMA 1: MANEJO DE FICHEROS (JAVA)

1. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS Y DIRECTORIOS
 - CREACIÓN
 - BORRADO
 - COPIA
 - MOVIMIENTO
2. FORMAS DE ACCESO A UN FICHERO
3. CLASES PARA GESTIÓN DE FLUJOS DATOS DESDE/HACIA FICHEROS
4. TRABAJO CON FICHEROS XML: ANALIZADORES SINTÁCTICOS (PARSER) Y VINCULACIÓN (BINDING)
5. EXCEPCIONES: DETECCIÓN Y TRATAMIENTO

FICHEROS

- Un fichero es un conjunto de bits almacenado en un dispositivo.
- Tiene una gran característica, los datos almacenados no se eliminan al apagar el dispositivo, por lo que tienen un almacenamiento persistente a diferencia de la RAM.
- Los ficheros se tienen 3 secciones muy importantes
 - RUTA → Dónde se encuentra ubicado este fichero
 - NOMBRE → Cómo se llama el fichero
 - EXTENSIÓN → Qué tipo de fichero es
- Un fichero debe tener un nombre único en su ruta, pero pueden existir dos ficheros con el mismo nombre en rutas diferentes

/Users/aortega/Desktop/nayade/10_0.jpeg

FICHEROS

RUTA

Todos los ficheros están almacenados en un dispositivo.

Los ficheros almacenados no se eliminan al apagar el dispositivo, por lo que permanecen en la memoria, a diferencia de la RAM.

- Los ficheros se tienen 3 secciones muy importantes
 - RUTA → Dónde se encuentra ubicado este fichero
 - NOMBRE → Cómo se llama el fichero
 - EXTENSIÓN → Qué tipo de fichero es
- Un fichero debe tener un nombre único en su ruta, pero pueden existir dos ficheros con el mismo nombre en rutas diferentes

/Users/aortega/Desktop/nayade/10_0.jpeg

FICHEROS

RUTA

Tenido en un dispositivo.

Almacenados no se eliminan al apagar el dispositivo, por lo que a diferencia de la RAM.

- Los ficheros se tienen 3 secciones:
 - RUTA → Dónde se encuentra el fichero
 - NOMBRE → Cómo se llama el fichero
 - EXTENSIÓN → Qué tipo de fichero es
- Un fichero debe tener un nombre único en su ruta, pero pueden existir dos ficheros con el mismo nombre en rutas diferentes

NOMBRE

/Users/aortega/Desktop/nayade/10_0.jpeg

FICHEROS

RUTA

Todos los ficheros almacenados en un dispositivo se consideran como ficheros. Los almacenados no se eliminan de la memoria, a diferencia de la RAM.

EXTENSIÓN

NOMBRE

- Los ficheros se tienen 3 secciones:
 - RUTA → Dónde se encuentra el fichero
 - NOMBRE → Cómo se llama el fichero
 - EXTENSIÓN → Qué tipo de fichero es
- Un fichero debe tener un nombre único en su ruta, pero pueden existir dos ficheros con el mismo nombre en rutas diferentes

/Users/aortega/Desktop/nayade/10_0.jpeg

EXTENSIONES

- Las extensiones determinan que tipo de fichero es y por tanto como debe tratarlo el sistema operativo.
 - ¿Es un fichero de música?
 - ¿Es un ejecutable?
 - ¿Es un fichero web?
 - ¿Es una imagen?
- Cómo podéis observar para cada tipo de fichero debe de existir una extensión y cada software puede tener las suyas propias. Por ejemplo PowerPoint con la extensión pptx.
- Aunque la mayoría de las veces las extensiones tienen 3 letras suelen ir de las 2 a las 4.
 - .sh
 - .html
 - .pptx
 - .exe
 - .dmg

¿CÓMO ES UN FICHERO POR DENTRO?

- No existe una forma predeterminada de como es la estructura de un fichero ya que cada desarrollador puede diseñarlo como el crea conveniente, pero lo más importante es que está formado por bloques de bytes que guardan la información deseada.
- Estos bloques se denominan registros y gracias a la extensión y el software correspondiente el SO puede interpretar el fichero y mostrarlo de forma correcta.
- ¿Has abierto alguna vez una imagen en formato texto?



```
-sf32
Y? B????%????????????nXYZ o?B??XYZ $????XYZ b????paraff?
[chrn??T|L??&g\mluc
enUGIMPmluc
enUsRGB??C

??C
???????
??O?? 88?
t ndfF8
?k?b?YF?1??
23
A
`bP?????
PB* CT??3?* Fi?Tff??*"$&IL?,K?c?B*?D1?0?
??Fh?WB#$??M4???*Å?1"?X!?g
*Dh??1f0
d#62?IfAFf?d????? ?1?????????33!?B34
??f@1
I??A?X?3?" ??#3b!??d??!???"?K"
?F?5A??$?? `??%?F?5?@b?? ?
?"4#18?" ?!?. Cf?D#22?*?
?1??A??5?6733s#?7 d?4??r? 0?r?2F?FH'?? PfD# ??b?B!??!??F`?Y3b?|$?A??3C1?@?'Dr?
f?`?5a?H"? ?3hCV ?0?D1?@?@?0@?" ?B?P??!??AS ?Di*? .?F#fa(??0 B?
K???
```

FICHEROS EN JAVA

- Para gestionar todas las operaciones con los ficheros, se utilizan dos librerías incluídas en el jdk de Java.
 - Java.io → Java Input Output
 - Java.nio → Java Non-blocking Input Output
 - Esta segunda librería supone una mejora en la forma en la que se realizan las operaciones. Ambas pueden ser utilizadas, pero java.nio corrigió muchas de las deficiencias de java.io

OPERACIONES SOBRE FICHEROS

- Las operaciones básicas que un fichero admite son:
 - **Creación del fichero:** El fichero se almacena en el disco duro, este debe tener un nombre y extensión única.
 - **Apertura del fichero:** para poder realizar operaciones sobre un fichero debe estar abierto dentro de nuestro programa y así de esta manera tener un apuntador hacia sus dirección de memoria.
 - **Lectura:** Consiste en leer datos del fichero y así poder recuperarlos cuando sea necesario. Se debe disponer del permiso de lectura sobre el fichero.
 - **Escritura de datos:** Consiste en escribir datos en el fichero y que así la información sea persistida. Se deberán tener permisos de escritura sobre el fichero.
 - Altas
 - Modificaciones
 - Bajas → Dependiendo del tipo de acceso se realizará de una forma u de otra:
 - Secuenciales: Se crea un nuevo archivo sin los datos que se desean
 - Aleatorios: Se desactiva el registro para posteriormente cuando se haga una modificación sobreescribirlo.
 - Valor 1: el registro está activo y no se puede sobreescribir
 - Valor 0: el registro NO está activo y se puede sobreescribir
 - **Cierre del fichero:** El fichero se debe cerrar para que quede disponible para otros programas y así no producir errores en su estructura de datos. Si se queda abierto de forma indefinida puede que se den situaciones de bloqueo en el acceso a la escritura del mismo.

FORMAS DE ACCESO A UN FICHERO

- **Acceso secuencial:** El acceso se produce desde el primer registro y se va avanzando registro a registro para leer la información. Si deseamos acceder al registro 39 debemos leer los 38 anteriores.
 - Cómo podemos observar son los menos óptimos ya que es obligatorio recorrer de forma constante el fichero desde el inicio.
 - Un ejemplo puede ser un VHS, (película antigua) donde para ir a un fragmento de la misma debíamos avanzar o rebobinar sobre la cinta electromagnética para leer su información
- **Acceso aleatorio:** Se puede acceder directamente a un registro sin haber recorrido los anteriores. Casi todos los sistemas actuales utilizan este formato ya que es mucho más rápido que el secuencial.
 - Un ejemplo es un DVD donde podemos avanzar y retroceder a nuestro antojo desde cualquier punto.
 - Pasaría lo mismo con un vídeo online, es un fichero al que podremos acceder a cualquiera de sus segundos en cualquier momento sin tener que ver todo el vídeo entero.

FORMAS DE ACCESO A UN FICHERO



Se produce desde el primer registro y se va avanzando registro a registro hasta que queramos acceder al registro 39 debemos leer los 38 anteriores.

Es un acceso en los menos óptimos ya que es obligatorio recorrer de forma constante el fichero

en un disco duro o en un dispositivo de almacenamiento masivo (HDD, (película) o en una memoria flash (tarjeta de memoria)) donde para ir a un fragmento de la misma debíamos avanzar o retroceder por el dispositivo.



Acceso directo: accedemos directamente a un registro sin haber recorrido los anteriores. Casi como si estuviéramos accediendo directamente a una página web en formato ya que es mucho más rápido que el secuencial.

Este tipo de acceso es muy útil cuando queremos acceder a un registro de forma directa sin tener que avanzar y retroceder a nuestro antojo desde cualquier punto.

Un ejemplo de acceso directo es un fichero de video o audio, es un fichero al que podremos acceder a cualquiera de sus segundos en cualquier momento sin tener que ver todo el video entero.

Acceso secuencial

FORMAS DE ACCESO A UN FICHERO

- **Acceso secuencial:** El acceso se produce desde el principio para leer la información. Si deseamos acceder a un dato en concreto:
 - Cómo podemos observar son los menos óptimos ya que debemos leer todo lo anterior.
 - Un ejemplo puede ser un VHS, (película antigua) donde debemos rebobinar sobre la cinta electromagnética para acceder a una escena en concreto.
- **Acceso aleatorio:** Se puede acceder directamente a cualquier dato sin tener que leer los anteriores. Si deseamos acceder a un dato en concreto:
 - Un ejemplo es un DVD donde podemos avanzar directamente al momento sin tener que ver todo el vídeo entero.
 - Pasaría lo mismo con un vídeo online, es un fichero que podemos ver en cualquier punto sin tener que esperar a que termine el anterior.



Acceso aleatorio

ndo registro a registro
3 anteriores.
orma constante el fichero
mismo debíamos
ido los anteriores si
secuencia
cualquier pu
e sus segundas e cualquier



JAVA.IO

- Es la librería inicial de entrada y salida.
- Se encarga de gestionar las operaciones de entrada y salida de nuestro programa
 - Es muy usada, ya que las operaciones System.out y System.err la utilizan para poder mostrar los mensajes
- Trabaja con streams, un flujo de datos (en formato de bytes).
- La clase principal se llama File, ya que es la que nos permite abrir o crear ficheros donde luego leeremos/escribiremos

CLASE FILE

- La clase File, proporciona las herramientas necesarias para poder trabajar con ficheros y conocer sus propiedades.
- Tiene 4 constructores diferentes, dependiendo de como le pasemos la información
- Dependiendo de nuestras necesidades podemos utilizar uno u otro.
 - Aunque la clase se llama File sirve para gestionar directorios y ficheros !
 - Un detalle muy importante, el constructor no crea el fichero en el sistema, solo en RAM por lo que debemos asegurarnos de crear el fichero o directorio de forma programática (siguiente diapositiva)

```
File file1 = new File("path + nombre_fichero");
File file2 = new File ("path","nombre_fichero");
File file3 = new File(new File("path"),"nombre_fichero");
File file4 = new File(Uri uri);
```

```
File file1 = new File( pathname: "ficheros/file1.txt");
File file2 = new File( parent: "ficheros", child: "file2.txt");
File path  = new File( pathname: "ficheros");
File file3 = new File(path, child: "file3.txt");
```

FILE → FUNCIONES PRINCIPALES

Función	Explicación	Ejemplo
list()	Devuelve un listado con todos los ficheros y directorios del directorio utilizado	File ficheros = new File("ficheros"); ficheros.list();
listFiles()	Devuelve un listado con los ficheros del directorio	File ficheros = new File("ficheros"); ficheros.listFiles();
getPath()	Obtiene la ruta del fichero, el inicio es el punto de ejecución del programa	File ficheros = new File("ficheros"); System.out.println(ficheros.getPath());
getAbsolutePath()	Obtiene la ruta completa desde la raíz del sistema al fichero	File ficheros = new File("ficheros"); System.out.println(ficheros.getAbsolutePath());
getParent()	Obtiene el nombre del directorio padre	File ficheros = new File("ficheros"); System.out.println(ficheros.getParent());
canRead()	Devuelve true o false, dependiendo si se puede o no leer	File ficheros = new File("ficheros"); System.out.println(ficheros.canRead());
canWrite()	Devuelve true o false, dependiendo si se puede o no escribir	File ficheros = new File("ficheros"); System.out.println(ficheros.canWrite());
mkdir()	Crea un nuevo directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.mkdir());
createNewFile()	Crea un nuevo fichero	File ficheros = new File("ficheros"); System.out.println(ficheros.createNewFile());
delete()	Elimina un fichero o directorio, si es un directorio debe estar vacío.	File ficheros = new File("ficheros"); System.out.println(ficheros.delete());
renameTo()	Renombra un fichero o directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.renameTo(new File("eliminados")));

{ String[] }

{ String }

{ File }

{ boolean }

FILE → FUNCIONES PRINCIPALES

Función	Explicación	Ejemplo
list()	Devuelve un listado con todos los ficheros y directorios del directorio utilizado	File ficheros = new File("ficheros"); ficheros.list();
listFiles()	Devuelve un listado con los ficheros del directorio	File ficheros = new File("ficheros"); ficheros.listFiles();
getPath()	Obtiene la ruta del fichero, el inicio es el punto de ejecución del programa	File ficheros = new File("ficheros"); System.out.println(ficheros.getPath());
getAbsolutePath()	Obtiene la ruta completa desde la raíz del sistema al fichero	File ficheros = new File("ficheros"); System.out.println(ficheros.getAbsolutePath());
getParent()	Obtiene el nombre del directorio padre	File ficheros = new File("ficheros"); System.out.println(ficheros.getParent());
canRead()	Devuelve true o false, dependiendo si se puede o no leer	File ficheros = new File("ficheros"); System.out.println(ficheros.canRead());
canWrite()	Devuelve true o false, dependiendo si se puede o no escribir	File ficheros = new File("ficheros"); System.out.println(ficheros.canWrite());
mkdir()	Crea un nuevo directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.mkdir());
createNewFile()	Crea un nuevo fichero	File ficheros = new File("ficheros"); System.out.println(ficheros.createNewFile());
delete()	Elimina un fichero o directorio, si es un directorio debe estar vacío.	File ficheros = new File("ficheros"); System.out.println(ficheros.delete());
renameTo()	Renombra un fichero o directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.renameTo(new File("eliminados")));

String[]

String

File

boolean

FILE → FUNCIONES PRINCIPALES

Función	Explicación	Ejemplo
list()	Devuelve un listado con todos los ficheros y directorios del directorio utilizado	File ficheros = new File("ficheros"); ficheros.list();
listFiles()	Devuelve un listado con los ficheros del directorio	File ficheros = new File("ficheros"); ficheros.listFiles();
getPath()	Obtiene la ruta del fichero, el inicio es el punto de ejecución del programa	File ficheros = new File("ficheros"); System.out.println(ficheros.getPath());
getAbsolutePath()	Obtiene la ruta completa desde la raíz del sistema al fichero	File ficheros = new File("ficheros"); System.out.println(ficheros.getAbsolutePath());
getParent()	Obtiene el nombre del directorio padre	File ficheros = new File("ficheros"); System.out.println(ficheros.getParent());
canRead()	Devuelve true o false, dependiendo si se puede o no leer	File ficheros = new File("ficheros"); System.out.println(ficheros.canRead());
canWrite()	Devuelve true o false, dependiendo si se puede o no escribir	File ficheros = new File("ficheros"); System.out.println(ficheros.canWrite());
mkdir()	Crea un nuevo directorio	File ficheros = new File("ficheros"); ficheros.mkdir();
createNewFile()	Crea un nuevo fichero	File ficheros = new File("ficheros"); ficheros.createNewFile(); System.out.println(ficheros.createNewFile());
delete()	Elimina un fichero o directorio, si es un directorio debe estar vacío.	File ficheros = new File("ficheros"); System.out.println(ficheros.delete());
renameTo()	Renombra un fichero o directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.renameTo(new File("eliminados")));

String[]
String
File
boolean

FILE → FUNCIONES PRINCIPALES

Función	Explicación	Ejemplo
list()	Devuelve un listado con todos los ficheros y directorios del directorio utilizado	File ficheros = new File("ficheros"); ficheros.list();
listFiles()	Devuelve un listado con los ficheros del directorio	File ficheros = new File("ficheros"); ficheros.listFiles();
getPath()	Obtiene la ruta del fichero, en el punto de ejecución del programa	File ficheros = new File("ficheros"); ficheros.getPath();
getAbsolutePath()	Obtiene la ruta completa desde el punto de ejecución	File ficheros = new File("ficheros"); ficheros.getAbsolutePath();
getParent()	Obtiene el directorio superior	File ficheros = new File("ficheros"); ficheros.getParent();
canRead()	Devuelve true o false si el fichero es leído	File ficheros = new File("ficheros"); ficheros.canRead();
canWrite()	Devuelve true o false si el fichero se puede escribir	File ficheros = new File("ficheros"); ficheros.canWrite();
mkdir()	Crea un nuevo directorio	File ficheros = new File("ficheros"); ficheros.mkdir();
createNewFile()	Crea un nuevo fichero	File ficheros = new File("ficheros"); ficheros.createNewFile(); System.out.println(ficheros.createNewFile());
delete()	Elimina un fichero o directorio	File ficheros = new File("ficheros"); ficheros.delete();
renameTo()	Renombra un fichero o directorio	File ficheros = new File("ficheros"); System.out.println(ficheros.renameTo(new File("eliminados")));

La función `createNewFile()` puede lanzar la excepción `IOException` por lo que deberemos tratar la excepción en un bloque `try/catch` o lanzarla a la función superior

en el caso de esta entonces sólo se eliminan.

Con `mkdir/createFile`

OPERACIONES BÁSICAS DE FICHEROS

- Existen 4 operaciones sobre los ficheros esenciales para la gestión de los mismos:
 - Creación
 - Eliminación
 - Copia
 - Movimiento
- Por desgracia no todas ellas se pueden realizar con el API básico de Java.io, en esta están disponibles las operaciones de
 - Creación
 - Eliminación

```
File file = new File("teoria/operaciones_básicas");
file.createNewFile();
```

```
File file = new File("teoria/operaciones_basicas");
file.delete();
```

Existe otra operación que no está incluida en este bloque pero me gusta hacer hincapié en ella, el renombramiento de ficheros

Renombre

```
File file = new File("teoria/operaciones_basicas");
File rename = new File("teoria/operaciones_basicas1");
file.renameTo(rename);
```

FILEUTILS

- Para poder realizar más operaciones sobre los ficheros, disponemos de la clase FileUtils, de la librería de Apache Commons.
 - No viene con el jdk por defecto por lo que si deseamos utilizarla debemos añadirla de forma manual a nuestro proyecto.
- Está nos ofrece un conjunto de mecanismos para poder controlar los ficheros de forma muy sencilla, por ejemplo las funciones:
 - Copia
 - Movimiento

FILEUTILS

- Para poder realizar más operaciones sobre los ficheros, disponemos de la clase FileUtils, de la librería de Apache Commons.
 - No viene con el jdk por defecto por lo que si deseamos utilizarla debemos añadirla de forma manual a nuestro proyecto.
- Está nos ofrece un conjunto de mecanismos para poder controlar los ficheros de forma muy sencilla, por ejemplo las funciones:
 - Copia
 - Movimiento

```
try {  
    FileUtils.copyFileToDirectory(file, destino);  
    System.out.println("Fichero copiado");  
  
} catch (IOException ex) {  
    System.err.println("Error al copiar el  
    archivo");  
    ex.printStackTrace();  
}
```

FILEUTILS

- Para poder realizar más operaciones sobre los ficheros, disponemos de la clase FileUtils, de la librería de Apache Commons.
 - No viene con el jdk por defecto por lo que si deseamos utilizarla debemos añadirla de forma manual a nuestro proyecto.
- Está nos ofrece un conjunto de mecanismos para poder controlar los ficheros de forma muy sencilla, por ejemplo las funciones:
 - Copia
 - Movimiento

```
try {  
    FileUtils.moveFileToDirectory(file, destino,  
        true);  
    System.out.println("Fichero movido");  
} catch (IOException ex) {  
    System.err.println("Error al mover el  
    archivo");  
    ex.printStackTrace();  
}
```

FILEUTILS

- Para poder realizar más operaciones se incluye la clase FileUtils de Apache Commons.
 - No viene con el jdk por defecto por lo que tenemos que añadirlo a nuestro proyecto.
- Está nos ofrece un conjunto de métodos para manejar los ficheros, por ejemplo las funciones:
 - Copia
 - Movimiento

Os animo a visitar la documentación oficial para explorar todas las funciones que admite FileUtils ya que es una librería muy completa

[FileUtils.html](#)



EJERCICIOS

1. Crea un directorio llamado "ejercicios"
2. Crea un fichero llamado ejercicio1, dentro del directorio ejercicios
3. Muestra por pantalla la longitud del fichero con nombre "ejercicio1"
4. Crea un fichero llamado ejercicio2 , dentro del directorio ejercicios
5. Muestra todos los ficheros del directorio ejercicios
6. Elimina el fichero llamado ejercicio1
7. Muestra todos los ficheros del directorio ejercicios
8. Elimina nuevamente el fichero llamado fichero1.
 - ¿Has podido?

STREAMS – ACCESO SECUENCIAL

- Al momento de trabajar en Java con los streams deberemos diferenciar dos tipos de ellos:
 - Streams de bytes: Se basan en enviar la información en bloques de 8 bits, 1 byte, también son conocidos como ficheros binarios
 - Streams de caracteres: Se basan en enviar la información en bloques de 16 bits, 2 bytes, esto es debido a la codificación Unicode que usa 16bits para representar cada carácter(UTF-16), también conocidos como archivos de caracteres
- Es muy importante determinar que tipo de stream necesitamos y hacer la programación adecuada. La elección nos hará utilizar unas clases u otras.

STREAMS – ACCESO SECUENCIAL

- Al momento de trabajar en Java con los streams:
 - Streams de bytes: Se basan en enviar la información en forma de bytes. Usan para leer y escribir ficheros binarios
 - Streams de caracteres: Se basan en enviar la información en forma de caracteres. Usan la codificación Unicode que usa 16bits para representar cada carácter. Se usan para leer y escribir archivos de caracteres
- Es muy importante determinar qué tipo de stream necesitamos y hacer la programación adecuada. La elección nos hará utilizar unas clases o otras.



¿Un fichero Word es binario o de caracteres?

STREAMS – ACCESO SECUENCIAL

- Al momento de trabajar en Java con los streams:
 - Streams de bytes: Se basan en enviar la información en forma de bytes. Usados para ficheros binarios
 - Streams de caracteres: Se basan en enviar la información en forma de caracteres. Usados para archivos de caracteres
- Es muy importante determinar qué tipo de stream necesitamos y hacer la programación adecuada. La elección nos hará utilizar unas clases diferentes.

¿Un fichero Word es binario o de caracteres?

Aunque su uso principal es para texto, este es un archivo binario ya que puede incluir imágenes y formas por lo que necesita de un software especial para ser leído.



STREAM 8 BITS

- Los streams de 8 bits, 1 byte, utilizan las clases **InputStream** para entrada de datos y **OutputStream** para salida de datos.
 - Un **InputStream** nos permite leer bytes de un array, de un **String**, de un fichero, etc.
 - Un **OutputStream** nos permite todo lo contrario hacer salida de datos a un array o un fichero

STREAM 8 BITS

- Los streams de 8 bits, 1 byte utilizan las clases **InputStream** para entrada de datos y **OutputStream**
 - Un **InputStream** nos permite leer datos de un dispositivo de lectura, de un array de bytes o de un fichero.
 - Un **OutputStream** nos permite todo lo contrario hacer salida de datos a un array o un fichero

ByteArrayInputStream

Permite leer un array de bytes de la RAM

ObjectInputStream

Permite convertir bytes en objetos

FileInputStream

Permite leer un fichero de bytes

STREAM 8 BITS

- Los streams de 8 bits, 1 byte, utilizan las clases **InputStream** para entrada de datos y **OutputStream** para salida de datos.
 - Un **InputStream** nos permite leer bytes de un array, de un **String**, de un fichero, etc.
 - Un **OutputStream** nos permite todo lo contrario hacer salida de datos a un array o un fichero

STREAM 8 BITS

La clase debe implementar la interfaz Serializable

- Un OutputStream nos permite hacer salidas a un fichero

Permite escribir un array de bytes almacenado en la RAM

ByteArrayOutputStream

Permite convertir objetos en bytes

ObjectOutputStream

Permite escribir bytes en ficheros

CONSTRUCTORES: *INPUTSTREAM

ByteArrayInputStream

ByteArrayInputStream(byte[] **buf**)

buf: buffer de entrada

ByteArrayInputStream(byte[] **buf**, int **offset**,int **length**)

buf: Buffer de entrada

offset: Desplazamiento inicial dentro del buffer

length: Número máximo de bytes leídos en el buffer

ObjectInputStream

ObjectInputStream(InputStream **in**)

in: InputStream con los datos para leer

FileInputStream

FileInputStream(String **name**)

name: Nombre del fichero

FileInputStream(File **file**)

file: Fichero de entrada

CONSTRUCTORES: *INPUTSTREAM

ByteArrayInputStream

ByteArrayInputStream(byte[] buffer)

buffer: buffer de entrada

ByteArrayInputStream(int length)

length: Tamaño del buffer

bytes leídos en el buffer

ObjectInputStream

ObjectInputStream()

bytes para leer

Pero... ¿Qué es un buffer?

Un Buffer es un espacio de memoria en el que se almacenan los datos de forma temporal, una vez que los datos se leen estos se eliminan



FileInputStream(InputStream(String name))

name: Nombre del fichero

FileInputStream

FileInputStream(File file)

file: Fichero de entrada

CONSTRUCTORES: *OUTPUTSTREAM

ByteArrayOutputStream

ByteArrayOutputStream(int size)

size: Capacidad inicial

ObjectOutputStream

ObjectOutputStream(OutputStream output)

output: outputStream donde escribir los datos

FileOutputStream

FileOutputStream(String name)

name: Nombre del fichero de salida

FileOutputStream(String name, boolean append)

name: Nombre del fichero de salida
append: opción para escribir al inicio o final de fichero

FileOutputStream(File file)

file: Fichero de salida

FileOutputStream(File file, boolean append)

file: Fichero de salida
append: opción para escribir al inicio o final de fichero

EJEMPLO: CONVERTIR UN OBJETO EN UN FICHERO

```
File file = new File("ficheros/8bits");
FileOutputStream fileOutputStream = new FileOutputStream(file);
Ejemplo ejemplo = new Ejemplo(1, "Texto de prueba para el ejemplo");
System.out.println("Ejemplo antes de fichero:");
System.out.println(ejemplo);
byte[] bytes = objetToBytes(ejemplo);
fileOutputStream.write(bytes);
fileOutputStream.close();
```

```
private static byte[] objetToBytes(Object object) {
    byte[] bytes = new byte[] {};
    try {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(object);
        bytes = baos.toByteArray();
        baos.close();
        oos.close();
    } catch (IOException ex) {
        System.err.println("Error en la descomposición del fichero");
        ex.printStackTrace();
    }
    return bytes;
}
```

EJEMPLO: CONVERTIR UN OBJETO EN UN FICHERO

```
File file = new File("ficheros/8bits");
FileOutputStream fileOutputStream = new FileOutputStream(file);
Ejemplo ejemplo = new Ejemplo(1, "Texto de prueba para el ejemplo");
System.out.println("Ejemplo antes de fichero:");
System.out.println(ejemplo);
byte[] bytes = objetToBytes(ejemplo);
fileOutputStream.write(bytes);
fileOutputStream.close();
```

Es importante cerrar los streams para asegurarnos que todos los datos han sido procesados (y así limpiar los buffers)

```
private static byte[] objetToBytes(Object object) {
    byte[] bytes = new byte[] {};
    try {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(object);
        bytes = baos.toByteArray();
        baos.close();
        oos.close();
    } catch (IOException ex) {
        System.err.println("Error en la descomposición del fichero");
        ex.printStackTrace();
    }
    return bytes;
}
```

EJEMPLO: CONVERTIR UN FICHERO EN UN OBJETO

```
File file = new File("ficheros/8bits");
FileInputStream fileInputStream = new FileInputStream(file);
Ejemplo ejemplo1 = (Ejemplo)
byteToObject(fileInputStream.readAllBytes());
fileInputStream.close();
System.out.println("Ejemplo despues del fichero");
System.out.println(ejemplo1);
```

```
private static Object byteToObject(byte[] bytes) {
    Object object = new Object();
    try {
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ObjectInputStream ois = new ObjectInputStream(bais);
        object = ois.readObject();
        ois.close();
        bais.close();
    } catch (IOException | ClassNotFoundException ex) {
        System.err.println("Error en la creación del objeto desde un fichero");
        ex.printStackTrace();
    }
    return object;
}
```

PUNTOS IMPORTANTES

- Todo objeto que se desee guardar en un fichero deberá implementar la interfaz Serializable
`public class Ejemplo implements Serializable`
- Existen diferentes tipos de Exceptions que deben ser controladas:
 - IOException: Excepción producida al leer o escribir datos
 - FileNotFoundException: Excepción producida cuando no encuentra el fichero solicitado
 - ClassNotFoundException: Excepción producida cuando la JVM no es capaz de recuperar la clase que se está indicando.

STREAM DE 16 BITS

- Los streams de 16 bits, 2 bytes, orientados a caracteres utilizan unas clases que ya están preparadas para esta lectura de datos. En este caso nos encontramos con las clases **Reader** y **Writer**.
- De estas clases utilizaremos las implementaciones de `FileReader` y `FileWriter` para nuestras operaciones de lectura y escritura respectivamente para tratar los ficheros.
 - Al crear un objeto de la clase `FileWriter` se puede pasar un segundo parámetro de tipo boolean, este especificará si escribirá en el fichero desde el inicio (`false`) o continuará desde el último punto (`true`).

CONSTRUCTORES

FileReader

```
FileReader(String name)  
FileReader(String File)  
FileReader(String name, Charset charset)  
FileReader(File file, Charset charset)
```

FileWriter

```
FileWriter(String name)  
FileWriter(String name, boolean append)  
FileWriter(File file)  
FileWriter(File file, boolean append)  
FileWriter(String name, Charset charset)  
FileWriter(String name, Charset charset, boolean append)  
FileWriter(File file, Charset charset)  
FileWriter(File file, Charset charset, boolean append)
```

CONSTRUCTORES

FileReader

FileReader(String name)

FileReader(String File)

FileReader(String name, Charset charset)

FileReader(File file, Charset charset)

FileWriter

FileWriter(String name)

FileWriter(String name, boolean append)

FileWriter(File file)

FileWriter(File file, boolean append)

FileWriter(String name, Charset charset)

FileWriter(String name, Charset charset, boolean append)

FileWriter(File file, Charset charset)

FileWriter(File file, Charset charset, boolean append)

Un charset es la codificación de los caracteres. Nos permite leer caracteres que de forma inicial no están previstos como símbolos de otros lenguajes

EJEMPLO: ESCRIBIR EN UN FICHERO

```
File file = new File("ficheros/caracteres.txt");
try {
    FileWriter fileWriter = new FileWriter(file);
    fileWriter.write("Esto es un texto de prueba");
    fileWriter.close();
} catch (IOException ex) {
    System.err.println("Error de apertura/escritura en el fichero: " +
file.getName());
}
```

EJEMPLO: LEER DE UN FICHERO

```
try {  
    File file = new File("ficheros/caracteres.txt");  
    FileReader fileReader = new FileReader(file);  
    int read;  
    while ((read = fileReader.read()) != -1) {  
        System.out.print((char) read);  
    }  
    System.out.println();  
} catch (IOException ex) {  
    System.err.println("Error de apertura/escritura en el fichero: " +  
file.getName());  
}
```



La función read, lee en cada vuelta del bucle 2 bytes, correspondientes a 1 carácter. Esta función realiza la lectura en número enteros por lo que debemos convertirlo en un carácter. Esta última acción se puede hacer con un simple casteo ya que cada número entero tiene una representación en forma de carácter. Podemos ver estas equivalencias gracias a una tabla de código ASCII

<https://elcodigoascii.com.ar>

EJERCICIOS

9. Crea una nueva clase llamada Persona con los atributos (id,nombre,edad,dni).
 1. Crea una función para guardar un objeto Persona en un fichero con el nombre persona1
 2. Crea una función para recuperar un objeto Persona del fichero persona1
 3. Modifica sus propiedades y vuelve a guardarlo en el fichero persona1
10. Crea un fichero de texto utilizando la clase FileWriter
 - El fichero debe contener la información:

“Esto es un texto de prueba,
Estamos creando nuestro primer fichero de texto
<https://codigonline.com>”
11. Lee un fichero de tipo imagen y muestra su contenido por pantalla.
 - ¿Se puede leer de forma correcta?

ACCESO ALEATORIO

- Para el acceso secuencial hemos visto diferentes clases ya sean para trabajar con ficheros binarios (1byte) o ficheros de caracteres (2 bytes).
- Para el acceso aleatorio la cosa se simplifica y únicamente disponemos de una sola clase. `RandomAccessFile`, esta nos proporciona todos los mecanismos para movernos por un fichero de forma aleatoria utilizando un apuntador que se va moviendo por los registros.



APUNTADOR

ACCESO ALEATORIO

- Para el acceso secuencial hemos visto diferentes clases ya sean para texto (1byte) o ficheros de caracteres (2 bytes).
- Para el acceso aleatorio la cosa se simplifica y únicamente disponemos de una clase, **RandomAccessFile**, esta nos proporciona todos los mecanismos para movernos por un fichero de forma aleatoria utilizando un apuntador que se va moviendo por los registros.

R1	R2	R3	R4	R5	R6	R7	R8	R9
----	----	----	----	----	----	----	----	----

APUNTADOR

VAMOS AL REGISTRO 5
(R5)

ACCESO ALEATORIO

- Para el acceso secuencial hemos visto diferentes clases ya sean para textos (1byte) o ficheros de caracteres (2 bytes).
- Para el acceso aleatorio la cosa se simplifica y únicamente disponemos de una clase, **RandomAccessFile**, esta nos proporciona todos los mecanismos para movernos por un fichero de forma aleatoria utilizando un apuntador que se va moviendo por los registros.

VAMOS AL REGISTRO 2
(R2)

R1	R2	R3	R4	R5	R6	R7	R8	R9
----	----	----	----	----	----	----	----	----

APUNTADOR

ACCESO ALEATORIO

- Para el acceso secuencial hemos visto diferentes clases ya sean para texto (1byte) o ficheros de caracteres (2 bytes).
- Para el acceso aleatorio la cosa se simplifica y únicamente disponemos de un **RandomAccessFile**, esta nos proporciona todos los mecanismos para movernos por un fichero de forma aleatoria utilizando un apuntador que se va moviendo por los registros.

R1	R2	R3	R4	R5	R6	R7	R8	R9
----	----	----	----	----	----	----	----	----

APUNTADOR

VAMOS AL REGISTRO 8
(R8)

CONSTRUCTORES

RandomAccessFile

**RandomAccessFile(String
name, String
accesMode)**

name: Nombre del
fichero a abrir

accesMode: Modo
de acceso al
fichero

r: lectura

rw: lectura y escritura

**RandomAccessFile(File
file, String accesMode)**

file: fichero a abrir

accesMode: Modo
de acceso al
fichero

r: lectura

rw: lectura y escritura

FUNCIONES MÁS IMPORTANTES

RandomAccessFile

getFilePointer

Devuelve la posición
del puntero

seek(long pos)

Establece la posición
del puntero

length()

Devuelve el tamaño
del fichero en bytes

skipBytes(int salto)

Desplaza el puntero
"salto" posiciones

EJEMPLO RANDOMACCESFILE

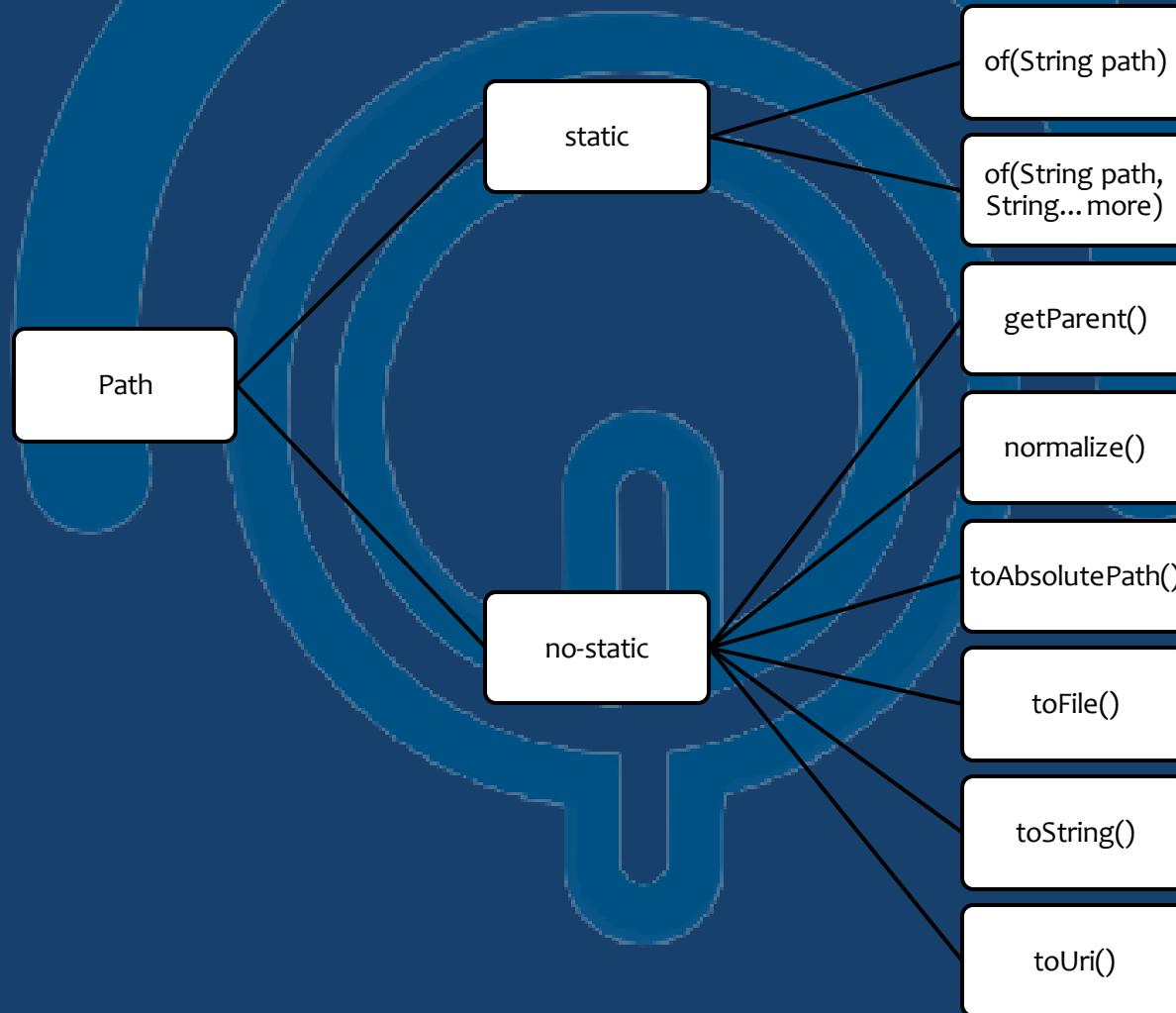
```
File file = new File("teoria/aleatorio.txt");
try {
    RandomAccessFile randomAccessFile = new
RandomAccessFile(file, "rw");
    System.out.println("Posición del puntero: " +
randomAccessFile.getFilePointer());
    String texto = "Linea 1\nLinea 2";
    randomAccessFile.seek(0);
    randomAccessFile
        .write(texto.getBytes(StandardCharsets.UTF_8));
    System.out.println("Posición del puntero: " +
randomAccessFile.getFilePointer());
    randomAccessFile.seek(0);
    String line;
    while ((line= randomAccessFile.readLine())!=null){
        System.out.println("Texto leído del fichero:");
        System.out.println(line);
    }
}
```

```
System.out.println("Posición del puntero: " +
randomAccessFile.getFilePointer());
randomAccessFile.seek(2);
System.out.println("Posición del puntero: " +
randomAccessFile.getFilePointer());
randomAccessFile.close();
} catch (FileNotFoundException e) {
    System.err.println("No se ha encontrado el fichero: " +
file.getName());
    e.printStackTrace();
} catch (IOException e) {
    System.err.println("Error al recuperar o insertar información");
    e.printStackTrace();
}
```

JAVA.NIO

- Java.nio incluye nuevas clases para trabajar con ficheros
 - **Path:** Sirve para manejar las rutas de los ficheros.
 - **Files:** Sirve para controlar las operaciones básicas de los ficheros
 - **FileSystem:** Sirve para obtener referencias al sistema de archivos

PATH – FUNCIONES MÁS IMPORTANTES



FILES – FUNCIONES MÁS IMPORTANTES

Files

copy(Path source, Path target)

copy(Path source, OutputStream out)

copy(InputStream in, Path target)

createDirectory(Path dir)

createFile(Path path)

delete(Path path)

deleteIfExists(Path path)

exists(Path path)

readAllLines(Path path)

readAllBytes(Path path)

write(Path path, byte[] bytes)

writeString(Path path, CharSequence chr)

FILES – FUNCIONES MÁS IMPORTANTES

Files

copy(Path source, Path target)

copy(Path source, OutputStream out)

copy(InputStream in, Path target)

createDirectory(Path dir)

createFile(Path path)

delete(Path path)

deleteIfExists(Path path)

exists(Path path)

readAllLines(Path path)

readAllBytes(Path path)

write(Path path, byte[] bytes)

writeString(Path path, CharSe

Todas las funciones de la clase Files son estáticas.



EJEMPLO – LISTADO DE ARCHIVOS

```
Path path = Path.of("teoria");
try {
    Stream<Path> files = Files.list(path);
    for(Path path1: files.collect(Collectors.toList())){
        System.out.println(path1);
    }
} catch (IOException e) {
    System.err.println("Error al leer el directorio: "+path.getFileName());
    e.printStackTrace();
}
```

EJEMPLO – ESCRIBIR BYTES

```
Path path = Path.of("teoria/nio.txt");
byte[] a = { 20, 10, 30, 5 };
System.out.println("Byte[] inicial");
for (byte item: a){
    System.out.println(item);
}
try {
    Files.write(path, a);
} catch (IOException ex) {
    System.err.println("Error al escribir los bytes");
}
```

EJEMPLO – LEER BYTES

```
Path path = Path.of("teoria/nio.txt");
try {
    byte[] b = Files.readAllBytes(path);
    System.out.println("Byte[] recuperado");
    for (byte item: b){
        System.out.println(item);
    }
} catch (IOException ex){
    System.err.println("Error al leer los bytes");
}
```

EJEMPLO – ESCRIBIR CARACTERES

```
Path path = Path.of("teoria/nio.txt");
try {
    Files.writeString(path, "Esto es un texto de prueba");
} catch (IOException ex) {
    System.err.println("Error al escribir en el fichero: " +
path.getFileName());
}
```

EJEMPLO – ESCRIBIR CARACTERES

```
Path path = Path.of("teoria/nio.txt");
try{
    String texto = Files.readString(path);
    System.out.println(texto);
}catch (IOException ex){
    System.err.println("Error al leer el fichero: "+path.getFileName());
}
```

EXTRAS CLASE FILES

- Adicionalmente la clase Files nos ofrece mecanismos para copiar y mover archivos que no se encontraba en la clase File.
- Estas operaciones son muy sencillas de realizar y no necesitan de librerías externas ([FileUtils](#)).
 - Copia:
 - Movimiento:

```
Path path = Path.of("teoria/nio.txt");
Path copy = Path.of("teoria/nio_copia.txt");
try{
    Files.copy(path, copy);
} catch (IOException ex){
    System.err.println("No se ha podido copiar el
fichero");
}
```

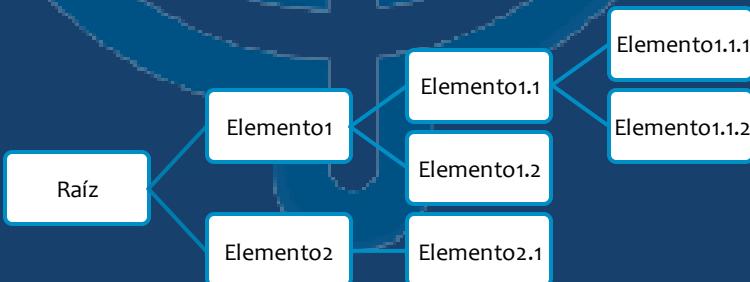
EXTRAS CLASE FILES

- Adicionalmente la clase Files nos ofrece mecanismos para copiar y mover archivos que no se encontraba en la clase File.
- Estas operaciones son muy sencillas de realizar y no necesitan de librerías externas ([FileUtils](#)).
- Copia:
- Movimiento:

```
Path path = Path.of("teoria/nio.txt");
Path move = Path.of("teoria/tema1/nio.txt");
try{
    Files.move(path, move);
} catch(IOException ex) {
    System.err.println("No se ha podido mover el
fichero");
}
```

FICHEROS XML

- XML: Las siglas de XML vienen de eXtensible Markup language.
- Este nos permite crear estructuras anidando los elementos unos dentro de otros y definir el contenido de cada elemento.
- Algo muy importante de XML es que sigue una jerarquía en forma de árbol, por lo que todo elemento (excepto la raíz) tiene un único padre, pero cada elemento puede tener muchos hijos.
 - De la misma forma no se pueden dar elementos cíclicos, un elemento no puede ser padre e hijo de otro elemento



FICHEROS XML

- Los documentos XML son muy fáciles de leer, ya que tiene una sintaxis muy sencilla.
 - Está basado en etiquetas que tienen un nombre y a su vez pueden tener atributos
- Son muy usados en ficheros de configuración de algunos programas o protocolos SOAP para enviar información a los servidores y ejecutar diferentes rutinas.
- Etiquetas XML
 - Cada etiqueta XML tendrá un inicio que se definirá dentro de los símbolos “<etiqueta>”
 - De la misma forma tendrá un fin que se definirá dentro de los símbolos </etiqueta>
 - Entre la etiqueta de inicio y de fin pueden ir otras etiquetas creando así la estructura
- Atributos XML
 - A su vez una etiqueta puede tener o..* atributos, elementos opcionales que dotan al XML de mayor información.
 - Algo muy importante es que toda información en atributos puede ser representada en elementos, mientras que no todos los elementos pueden ser atributos.
 - ¿Cómo saber donde va la información?
 - La única forma de determinarlo es saber si el elemento estará repetido y si tendrá hijos. En estos dos casos la información no puede ir en atributos

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

↔
Equivalentes, a excepción
de la cantidad en
ingredientes

```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

ETIQUETA

```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

ETIQUETA

ATRIBUTOS

```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

ETIQUETA

ATRIBUTOS

VALORES

```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

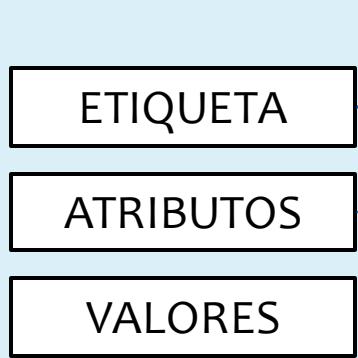
```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```



```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

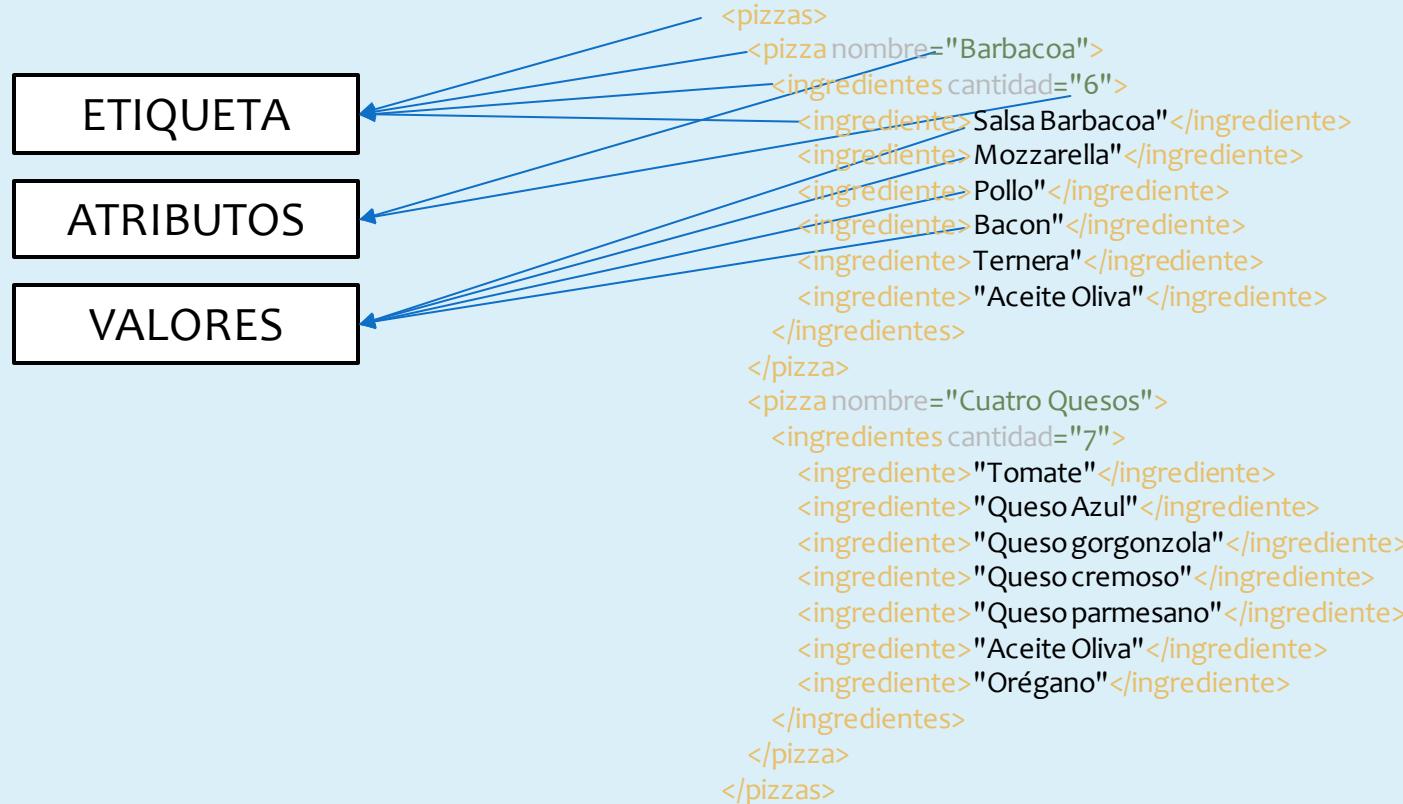
```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```



```
<pizzas>
  <pizza nombre="Barbacoa">
    <ingredientes cantidad="6">
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza nombre="Cuatro Quesos">
    <ingredientes cantidad="7">
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```

EJEMPLO XML

```
<pizzas>
  <pizza>
    <nombre>Barbacoa</nombre>
    <ingredientes>
      <ingrediente>Salsa Barbacoa"</ingrediente>
      <ingrediente>Mozzarella"</ingrediente>
      <ingrediente>Pollo"</ingrediente>
      <ingrediente>Bacon"</ingrediente>
      <ingrediente>Ternera"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
    </ingredientes>
  </pizza>
  <pizza>
    <nombre>Cuatro Quesos</nombre>
    <ingredientes>
      <ingrediente>"Tomate"</ingrediente>
      <ingrediente>"Queso Azul"</ingrediente>
      <ingrediente>"Queso gorgonzola"</ingrediente>
      <ingrediente>"Queso cremoso"</ingrediente>
      <ingrediente>"Queso parmesano"</ingrediente>
      <ingrediente>"Aceite Oliva"</ingrediente>
      <ingrediente>"Orégano"</ingrediente>
    </ingredientes>
  </pizza>
</pizzas>
```



FICHEROS XML

- Existe formas diferentes de poder leer ficheros XML, ver su estructura y atributos. Estas herramientas son conocidas como XML-parser
- Las más utilizadas son DOM y SAX
 - DOM: Document Object Model, el procesador lee todo el documento XML y lo almacena en memoria RAM.
 - Es muy útil cuando queremos acceder de forma rápida a un elemento del árbol, ya que tiene toda la información precargada
 - Contra más grande es el documento más tiempo y memoria necesita para procesarlo
 - SAX: Simple Api for XML, el procesador va leyendo de forma secuencial el fichero y va lanzando eventos que nuestro programa debe capturar. Para cada etiqueta de inicio/fin, atributo y valor emitirá un evento.
 - Es mucho más rápido que DOM y consume menos memoria.
 - En contra si queremos acceder a un elemento en concreto debemos de recorrer todo el documento.

FICHEROS XML - DOM

- Clases más importantes:
 - **DocumentBuilderFactory**: Es una clase especial, nos da la capacidad de poder crear parsers para nuestro programa.
 - **DocumentBuilder**: Define el parser DOM que se va a utilizar.
 - **Document**: Objeto que contiene la lectura completa del XML.
 - **Node**: Representa a cualquier Nodo del árbol.
 - **NodeList**: Lista que contiene todos los nodos hijos de un nodo.
 - **Element**: Es un tipo de nodo, representa un elemento del XML
 - **Attr**: Representa un atributo de un Nodo
 - **Text**: Representa el texto de un elemento
- Gracias a estas clases podemos leer/escribir documentos XML utilizando DOM

DOM – LEER ARCHIVOS

- Antes de empezar, debemos tener en cuenta que los espacios en blanco dentro de una etiqueta los lee como texto y debemos tener mucho cuidado al momento de tratar el fichero.
 - Recomiendo utilizar algún simplificador de XML como por ejemplo <https://codebeautify.org/xmlviewer>, donde podremos eliminar todos los espacios en blanco dándole al botón de *Minify*
- Pasos para leer un documento:
 1. Crear el DocumentBuilderFactory
 2. Crear el DocumentBuilder
 3. Crear el Document, este punto es muy importante ya que cargará todo el documento en memoria.
 4. Leer las propiedades del documento deseadas
 1. Este último paso variará según el documento y las propiedades que deseamos leer

EJEMPLO – DOM LECTURA

```
Path path = Path.of("teoria/tema1/pizzas.xml");
File file = path.toFile();
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
try {
    builder = factory.newDocumentBuilder();
} catch (ParserConfigurationException ex){
    System.err.println("Error al crear el parser");
    ex.printStackTrace();
    return;
}
Document document = null;
try {
    document = builder.parse(file);
} catch (IOException | SAXException ex){
    System.err.println("Error al parsear el fichero " + file.getName());
}
```

```
NodeList pizzas =
document.getElementsByTagName("pizzas").item(0).getChildNodes();
System.out.println("Pizzas en el menu");
for(int i = 0; i < pizzas.getLength(); i++){
    Node node = pizzas.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE)
        switchElement(node);
}
```

```
private static void switchElement(Node node){
    Element element = (Element) node;
    switch (element.getNodeName()){
        case "pizza":
            System.out.print("Pizza: \t");
            System.out.println(getAttribute(element,
"nombre"));
            switchElement(element.getFirstChild());
            break;
        case "ingredientes":
            System.out.println("\tLista de ingredientes:");
            NodeList ingredientes = element.getChildNodes();
            for(int i = 0; i < ingredientes.getLength(); i++){
                switchElement(ingredientes.item(i));
            }
            break;
        case "ingrediente":
            System.out.print("\t\tIngrediente: ");
            System.out.println(getText(element));
            break;
        default:
    }}
```

```
private static String getAttribute(Element element,
String name){
    return element.getAttribute(name);
}

private static String getText(Element element){
    return element.getTextContent();
}
```

EJEMPLO – DOM LECTURA

```
Path path = Path.of("teoria/tema1/pizzas.xml");
File file = path.toFile();
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
try {
    builder = factory.newDocumentBuilder();
} catch (ParserConfigurationException ex){
    System.err.println("Error al crear el parser");
    ex.printStackTrace();
    return;
}
Document document = null;
try {
    document = builder.parse(file);
} catch (IOException | SAXException ex){
    System.err.println("Error al parsear el fichero " + file.getName());
}
```

```
private static void switchElement(Node node){
    Element element = (Element) node;
    switch (element.getNodeName()){
        case "pizza":
            System.out.print("Pizza: \t");
            System.out.println(getAttribute(element,
"nombre"));
            switchElement(element.getFirstChild());
            break;
        case "ingredientes":
            System.out.println("\tLista de ingredientes:");
            NodeList ingredientes = element.getChildNodes();
            for(int i = 0; i < ingredientes.getLength(); i++){
                switchElement(ingredientes.item(i));
            }
            break;
        case "ingrediente":
            System.out.print("\t\tIngrediente: ");
            System.out.println(getText(element));
            break;
        default:
    }
}
```

```
private static String getAttribute(Element element,
String name){
    return element.getAttribute(name);
}

private static String getText(Element element){
    return element.getTextContent();
}
```

EJEMPLO – DOM LECTURA

```
<?xml version="1.0" encoding="UTF-8"?>



Salsa Barbacoa
Mozzarella
Pollo
Bacon
Ternera
Aceite Oliva




Tomate
Queso Azul
Queso gorgonzola
Queso cremoso
Queso parmesano
Aceite Oliva
Oregano



```

TRANSFORMACIÓN

Pizzas en el menu
Pizza: Barbacoa
Ingredientes
Lista de ingredientes:
Ingrediente: Salsa Barbacoa
Ingrediente: Mozzarella
Ingrediente: Pollo
Ingrediente: Bacon
Ingrediente: Ternera
Ingrediente: Aceite Oliva

Pizza: Cuatro Quesos
Ingredientes
Lista de ingredientes:
Ingrediente: Tomate
Ingrediente: Queso Azul
Ingrediente: Queso gorgonzola
Ingrediente: Queso cremoso
Ingrediente: Queso parmesano
Ingrediente: Aceite Oliva
Ingrediente: Oregano

DOM – ESCRIBIR ARCHIVOS

- Escribir un documento XML con DOM es muy sencillo (**pero bastante tedioso**), únicamente debemos ir línea a línea estableciendo las características que deseamos.
- Pasos para escribir un documento:
 1. Crear el DocumentBuilderFactory
 2. Crear el DocumentBuilder
 3. Crear el DOMImplementation (**Difiere de la lectura donde no hace falta**)
 4. Crear el Document, este punto es muy importante ya que cargará todo el documento en memoria.
 5. Escribir las propiedades deseadas
 6. Transformar el DOM en memoria en un archivo del disco
 1. Se necesita un objeto Source (origen de los datos) y un Result (destino de los datos)
 2. Se enviará del origen al destino mediante la clase Transform

EJEMPLO – DOM ESCRITURA

```
DocumentBuilderFactory factory =  
DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = null;  
try {  
    builder = factory.newDocumentBuilder();  
} catch (ParserConfigurationException e) {  
    e.printStackTrace();  
}  
assert builder != null;  
DOMImplementation implementation =  
builder.getDOMImplementation();  
Document document = implementation.createDocument(null, null, null);  
document.setXmlVersion("1.0");  
document.setXmlStandalone(true);
```

```
Element pizzas =  
document.createElement("Pizzas");  
document.appendChild(pizzas);  
crearPizza(pizzas, document);
```

```
Source source = new DOMSource(document);  
Result result = new StreamResult(new  
File("teoria/tema1/pizzas_dom.xml"));  
Transformer transformer =  
TransformerFactory.newInstance().newTransformer();  
transformer.transform(source, result);
```

```
private static void crearPizza(Element element, Document document){  
    Element pizza = document.createElement("Pizza");  
    pizza.setAttribute("nombre", "Barbacoa");  
    ArrayList<Ingrediente> ingredienteList = new ArrayList<>(){  
        add(new Ingrediente("Salsa barbacoa", 200.0));  
        add(new Ingrediente("Mozzarella", 100));  
        add(new Ingrediente("Pollo", 20));  
        add(new Ingrediente("Bacon", 20));  
        add(new Ingrediente("Ternera", 20));  
        add(new Ingrediente("Aceite de Oliva", 10));  
    };  
    Element ingredientes = document.createElement("ingredientes");  
    ingredienteList.forEach(ingrediente -> {  
        Element element1 = document.createElement("ingrediente");  
        Text nombreIngrediente =  
        document.createTextNode(ingrediente.nombre);  
        element1.appendChild(nombreIngrediente);  
        Attr cantidadIngrediente = document.createAttribute("cantidad");  
        cantidadIngrediente.setValue(String.valueOf(ingrediente.cantidad));  
        element1.setAttributeNode(cantidadIngrediente);  
        ingredientes.appendChild(element1);  
    });  
    pizza.appendChild(ingredientes);  
    element.appendChild(pizza);  
}
```

FICHEROS XML - SAX

- Los parsers de tipo SAX funcionan de forma diferente a los de DOM, van leyendo el documento poco a poco y cada vez que encuentren un elemento, atributo, texto, etc., lanzarán un evento para que lo capturemos y podamos actuar en consecuencia.
- La API de SAX es mucho más compleja que la de DOM pero vamos a analizarla.
- Clases más importantes:
 - SAXParserFactory → Clase especial, nos permitirá crear nuevos SAX parsers
 - SAXParser → Creación de un nuevo parser para SAX, utiliza la clase anterior
 - XMLReader → Objeto que permite leer el documento XML elemento a elemento
 - DefaultHandler → Clase abstracta que debemos implementar, contiene las llamadas a los eventos

FICHEROS XML - SAX

- La clase DefaultHandler entre otras, contiene las siguientes funciones que son las que más nos interesan
 - startDocument()
 - endDocument()
 - **startElement()** → Elemento que empieza, podemos en este punto leer los atributos si los contiene
 - endElement()
 - **characters()** → Caracteres que contiene el elemento

FICHEROS XML - SAX

- Para leer un fichero XML con SAX seguiremos los siguientes pasos:
 1. Crear SAXParserFactory
 2. Crear el SAXParser
 3. Crear el XMLReader
 4. Implementar el DefaultHandler
 1. Dar funcionalidad a las diferentes funciones según nuestros requisitos
 5. Parsear el documento

EJEMPLO – SAX LECTURA

```
SAXParserFactory parserFactory = SAXParserFactory.newInstance();
SAXParser parser = parserFactory.newSAXParser();
XMLReader reader = parser.getXMLReader();
```

1

```
reader.parse("teoria/tema1/pizzas.xml");
```

3

```
reader.setContentHandler(new DefaultHandler() {
    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException{
        switch (qName){
            case "pizzas":
                System.out.println(qName);
                break;
            case "pizza":
                System.out.println("\tNombre: " + attributes.getValue(0));
                break;
            case "ingredientes":
                System.out.println("\t\tListado de ingredientes");
                break;
            case "ingrediente":
                System.out.print("\t\t\tIngrediente: ");
                break;
        }
    }
    @Override
    public void characters(char[] ch, int start, int length) throws SAXException{
        String text = new String(ch, start, length);
        System.out.println(text);
    }
});
```

2

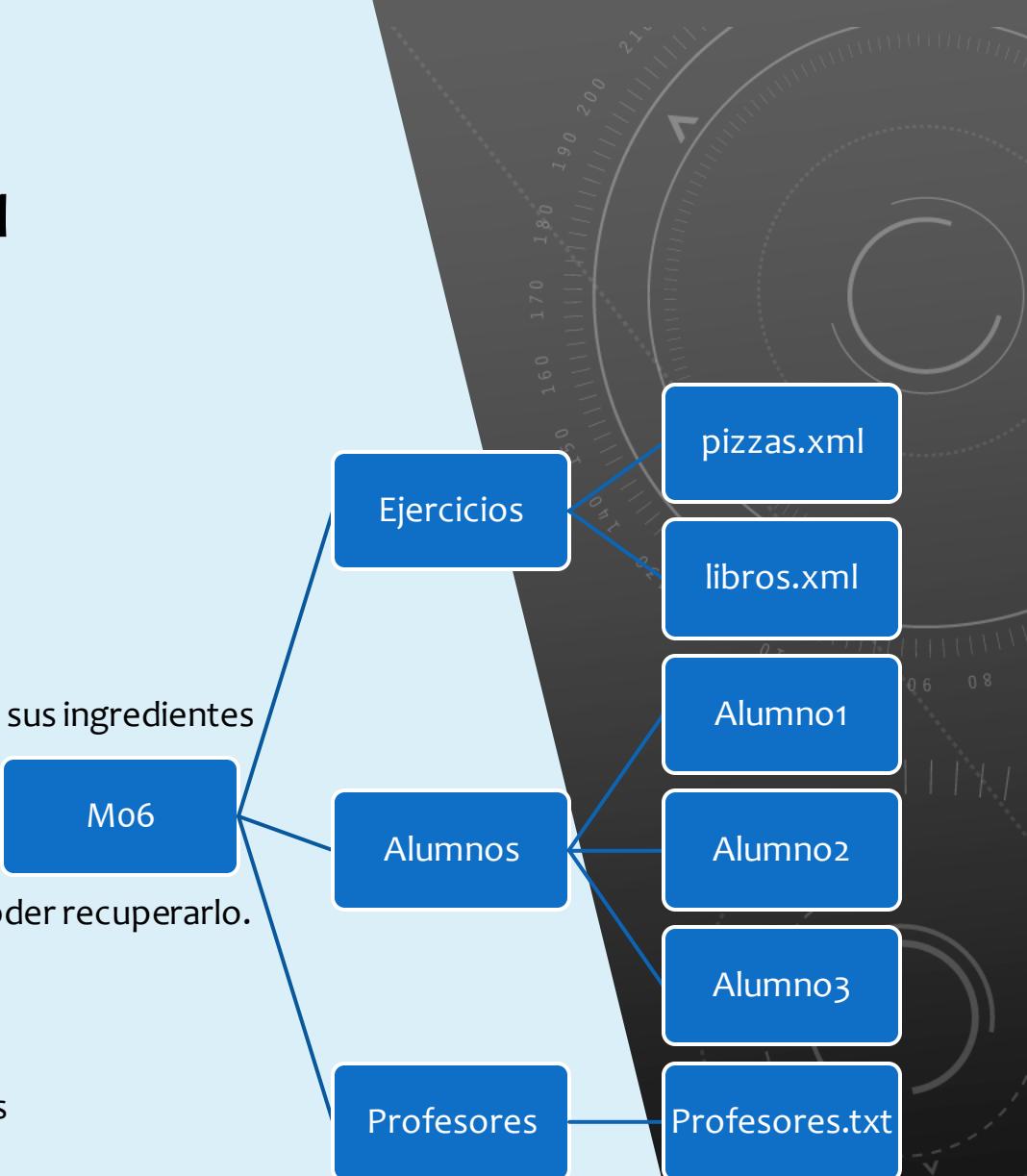
EJERCICIOS

- A partir del siguiente documento XML, [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms762271\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms762271(v=vs.85))
 12. Muestra por pantalla los diferentes id de cada libro utilizando la librería DOM
 13. Muestra por pantalla una lista de autores y los títulos de sus libros
 14. Muestra por pantalla los títulos de los libros y sus precios. Ordena de más económico a más caro.
 15. Muestra los libros por su genero
 16. Traduce todas las etiquetas del XML y guardarla en un fichero llamado libros.xml
 - Catalog → Catalogo
 - Book → Libro
 - Title → Título
 - Genre → Genero
 - Price → Precio
 - Public_date → Fecha de publicación
 - Description → Descripción



PRACTICA COMPLETA TEMA 1

- Realiza la siguiente practica, puedes preguntarme todas las dudas que tengas.
- Crea un programa que genere la siguiente estructura de directorios:
 - Directorio ejercicios
 - Crea el fichero pizzas.xml y rellénalo con diferentes pizzas junto con sus ingredientes
 - Lee el fichero libros.xml y muestra únicamente los autores
 - Directorio alumnos
 - Crea los 3 alumnos, guardando los objetos en ficheros para luego poder recuperarlo.
 - 1 alumno → 1 fichero
 - Directorio profesores
 - Crea un fichero de texto con el listado de nombres de los profesores



EJERCICIOS TEMA 1

EJERCICIOS

1. Crea un directorio llamado "ejercicios"
2. Crea un fichero llamado ejercicios, dentro del directorio ejercicios
3. Muestra por pantalla la longitud del fichero con nombre "ejercicios"
4. Crea un fichero llamado ejercicio_1, dentro del directorio ejercicios
5. Muestra todos los ficheros del directorio ejercicios
6. Elimina el fichero llamado ejercicios
7. Muestra todos los ficheros del directorio ejercicios
8. Elimina nuevamente el fichero llamado ejercicios.
 - ¿Has podido?

EJERCICIOS

1. Crea una clase clase Persona con los atributos (Nombre, Apellido, Edad).
 - a. Crea una función para guardar un objeto Persona en un fichero con el nombre persona.
 - b. Crea una función para recuperar un objeto Persona del fichero persona.
 - c. Modifica la propiedad nombre cuando se guarda en el fichero persona.
2. Crea un Archivo de texto utilizando la clase Nombre.
 - El Archivo debe contener la información:
"Este es un texto de prueba.
Estoy creando nuestra primera ficha de texto
Fichasdeprueba.txt"
3. Crea un Archivo de tipo Imagen y muestra su contenido por pantalla.
 - ¿Se puede leer de forma correcta?

EJERCICIOS

- A partir del siguiente documento XML, [Copia y Pega el siguiente código](#)
 1. Muestra por pantalla una lista de autores y los títulos de sus libros.
 2. Muestra por pantalla los títulos de los libros y sus precios. Ordena de más económico a más caro.
 3. Muestra los libros por su género.
 4. Trabaja todos los títulos del XML y guardálos en un fichero llamado libros.xml.
 - Catalog_M_Catalog
 - Book_M_Libro
 - Title_M_Titulo
 - Genre_M_Género
 - Price_M_Precio
 - PublicDate_M_Fecha_de_publicación
 - Description_M_Descripción

PRACTICA COMPLETA TEMA 1

Resuelve las siguientes prácticas y pídelas programadas todas las clases que tengas.

Crea un programa que genere el siguiente estructura de directorios:

- Directorio principal
 - Crea el directorio plazas con y relléntalo con diferentes plazas junto con sus ingredientes.
 - Llena el directorio libros con y muestra únicamente los autores.
- Crea Directorio alumnos
 - Crea los alumnos, guardando los objetos en Archivos para luego poder recuperarlos
 - Alumno_M_Alumno
- Directorio profesores
 - Crea un fichero de texto con el resultado de nombres de los profesores.

```
graph TD; Root[""] --> Plazas["Plazas"]; Root --> Libros["Libros"]; Root --> Alumnos["Alumnos"]; Root --> Profesores["Profesores"]; Plazas --> Plazas1["Plaza 1"]; Plazas --> Plazas2["Plaza 2"]; Plazas --> Plazas3["Plaza 3"]; Plazas --> Plazas4["Plaza 4"]; Plazas --> Plazas5["Plaza 5"]; Libros --> Libros1["Libro 1"]; Libros --> Libros2["Libro 2"]; Libros --> Libros3["Libro 3"]; Libros --> Libros4["Libro 4"]; Libros --> Libros5["Libro 5"]; Alumnos --> Alumno1["Alumno 1"]; Alumnos --> Alumno2["Alumno 2"]; Alumnos --> Alumno3["Alumno 3"]; Alumnos --> Alumno4["Alumno 4"]; Alumnos --> Alumno5["Alumno 5"]; Profesores --> Profesores1["Profesores 1"]; Profesores --> Profesores2["Profesores 2"]; Profesores --> Profesores3["Profesores 3"]; Profesores --> Profesores4["Profesores 4"]; Profesores --> Profesores5["Profesores 5"];
```

TEMA 2: MANEJO DE CONECTORES

- 
1. El desfase objeto-relacional
 2. Protocolos de acceso a bases de datos.
Conectores
 3. Ejecución de sentencias:
 - Descripción de datos
 - Modificación de datos
 - Consulta de datos

¿QUÉ ES UNA BASE DE DATOS?

- Es una colección de datos relacionados entre sí, que se almacenan para poder ser utilizados en un futuro.
- A diferencia de los ficheros, con las bases de datos podemos acceder a información repartida por el almacenamiento y agruparla para dar salida a un consulta muy concreta.
- Imaginemos, tenemos un fichero de Empleados y otro fichero de departamentos. Estos pese a poder ver que tienen relación (todo empleado tiene un departamento), no podemos unirlos y sacar un único resultado. Deberemos analizar cada fichero por separado y finalmente unir los datos para obtener la información, es un sistema poco eficiente.
 - Con las bases de datos esto no ocurre, ya que de por si la información está relacionada y permite acceder y recuperar a aquellos registros que deseamos (inclusive uniendo información entre ellos).
 - Si el ejemplo del fichero, lo extrapolamos a un uso intensivo con miles de usuarios el resultado puede ser catastrófico.
- Se podrían catalogar como la evolución lógica de los ficheros.
 - Se pueden restringir datos de entrada
 - Se puede evitar duplicidad
 - Puede ser consultado por miles de usuarios
 - Tiene un motor eficiente interpretando las consultas para obtener resultados muy precisos

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos
 - Selección.
 - Proyección
 - Combinación
 - División

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2



SELECCIÓN

SELECT * from table here d="D1"

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2



PROYECCIÓN

SELECT B,C,D from table

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2

D	E	F
D1	E1	F1
D2	E2	F2
D3	E3	F3
D4	E4	F4

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2

D	E	F
D1	E1	F1
D2	E2	F2
D3	E3	F3
D4	E4	F4



Unión

```
SELECT * from table t INNER JOIN table2 t2 ON t.D=t2.D where t.d="D1"
```

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B
A1	B1
A1	B2
A1	B3
A2	B1
A3	B1
A3	B2

B
B1
B2
B3

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B
A1	B1
A1	B2
A1	B3
A2	B1
A3	B1
A3	B2

B
B1
B2
B3

División

A
A1

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en álgebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos
 - Selección.
 - Proyección
 - Combinación
 - División



Gracias a este modelo matemático, si la BD está bien construida es muy eficiente y puede gestionar millones de registros

DESFASE OBJETO-RELACIONAL

- Las tecnologías, especialmente los lenguajes de programación han ido avanzando desde su origen, añadiendo más y más características. Dotando a los sistemas de más inteligencia y a su vez haciendo la programación más sencilla.
- Un claro ejemplo son los objetos donde ya no tenemos que trabajar con pseudo-estructuras de datos, si no que podemos crear estas estructuras y mantenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos encontramos otro objeto en su interior

DESFASE OBJETO-RELACIONAL

- Las tecnologías, especialmente los lenguajes de programación han ido avanzando desde su origen, añadiendo más y más características. Dotando a los sistemas de más inteligencia y a su vez haciendo la programación más sencilla.
- Un claro ejemplo son los objetos donde ya no tenemos que trabajar con pseudo-estructuras de datos, si no que podemos crear estas estructuras y mantenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos e

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

DESFASE OBJETO-RELACIONAL

- Las tecnologías de bases de datos han ido avanzando desde su origen, añadiendo más funcionalidades y más complejidad, las bases de datos de más inteligencia y a su vez haciendo más compleja la programación.
- Un clásico problema es que las bases de datos, para realizar esta operación debemos crear una segunda tabla donde unir estos datos y trabajar con pseudo-estructuras de datos, tenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos encontramos con que una columna de una tabla tiene más de un valor, por lo que por ejemplo no podríamos tener varias direcciones tal y como plantea la clase Persona.

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

OBJETO VS TABLA

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
  
    public Persona(int id, String nombre, int  
telefono, ArrayList<String> direcciones) {  
        this.id = id;  
        this.nombre = nombre;  
        this.telefono = telefono;  
        this.direcciones = direcciones;  
    }  
}
```

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

OBJETO VS TABLA

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
  
    public Persona(int id, String nombre, int telefono, ArrayList<String> direcciones) {  
        this.id = id;  
        this.nombre = nombre;  
        this.telefono = telefono;  
        this.direcciones = direcciones;  
    }  
}
```

¡De forma natural esto no puede cohesionar!



Nombre	telefono
Alvaro	712123123
_persona	direccion
1	C/ Falsa 123
2	C/Verdadera 123
3	C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

Unimos el ID de la clase
con el ID de la tabla

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

Unimos el ID de la clase
con el ID de la tabla

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direcciones;  
}
```

Unimos el ID de la clase
con el ID de la tabla

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

Deberemos crear unas consultas que nos consigan todos estos datos y actuar en consecuencia.

```
select p.id, p.nombre, p.telefono, d.direccion from personas  
p  
INNER JOIN direcciones d ON p.id = d.id_persona  
where p.id=1;
```

```
public class Persona {  
    int id;  
    String nombre;  
    int telefono;  
    ArrayList<String> direccio  
}
```



	...	Nombre	telefono
1		Alvaro	712123123
2		Tania	678987987

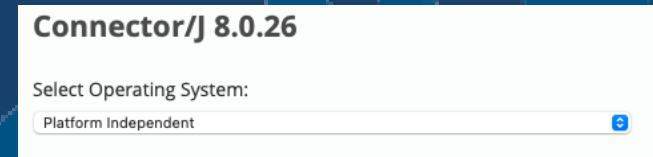
Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

CONNECTOR JDBC

- Para realizar las conexiones con la base de datos, utilizaremos JDBC (Java DataBase Connectivity), una librería encargada de darnos todas las funcionalidades de la base de datos
 - Conectarnos y Desconectarnos
 - Ejecutar consultas
 - Recuperar datos si fuese necesario
- No podemos utilizar JDBC directamente, ya que la programación interna se verá alterada según el motor de BD
 - Para utilizarlo deberemos buscar la librería de cada fabricante según el motor de base de datos utilizado.
 - Cambiar entre diferentes motores, serán tan sencillo como cambiar de librería, ya que utilizan la interfaz JDBC todas las peticiones se harán de la misma manera

INSTALAR JDBC PARA MYSQL - TRADICIONAL

- Buscar el connector de mysql
 - <https://dev.mysql.com/downloads/connector/j/>
- Seleccionar la opción de platform independent
- Descargar cualquiera de las dos opciones, la diferencia está en la compresión tar o zip.
- Por defecto os pedirá login, pero existe la opción de descargar sin loguearse “No thanks, just start my download”
- Una vez descargado deberemos añadirlo a nuestro IDE, este dependerá del mismo.



INSTALAR JDBC PARA MYSQL - TRADICIONAL

- Buscar el connector de mysql
 - <https://dev.mysql.com/downloads/connector/j/>
- Seleccionar la opción de platform independent
- Descargar cualquiera de las dos opciones, la diferencia es que una es para windows y otra para linux
- Por defecto os pedirá login, pero existe la opción “skip grant tables” en la configuración de “my download”
- Una vez descargado deberemos añadirlo a nuestro IDE, para ello tendremos que configurar la librería que dependerá del mismo.

¡JDBC NO ES UN MOTOR DE BASE DE DATOS ES UN CONECTOR!

DEBEREMOS TENER NUESTRO MOTOR DE BASE DE DATOS INSTALADO Y FUNCIONANDO

“no thanks, just start

INSTALAR JDBC - MAVEN

- Maven es una herramienta de software que entre otras utilidades puede gestionar las dependencias de nuestro proyecto.
- Esta herramienta contiene un fichero llamado pom.xml donde definiremos que librerías deseamos y nos las añadirá de forma automática.
 - No solo se pueden definir librerías, también podemos:
 - Pasos de compilación
 - Versión de Java utilizada
 - Autor del proyecto
 - Descripción del proyecto.
 - ...
 - Como podemos observar es realmente potente y junto a gradle y npm forman parte de las tres herramientas de software más utilizadas en la actualidad

INSTALAR JDBC - MAVEN

- Pasos para utilizar jdbc con maven
 1. Crear un proyecto Maven
 2. Editar el fichero pom.xml
 3. Añadir las dependencias (librerías necesarias)
 4. Actualizar fichero pom.xml

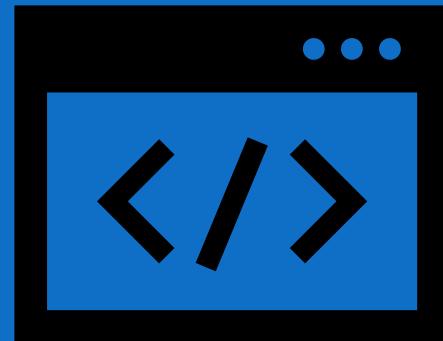
ARQUITECTURAS SOPORTADAS POR JDBC

- A diferencia de otras tecnologías como por ejemplo Android, JDBC es compatible con el modelo de 2 y 3 capas
 - **2 capas**, el frontend (lo ve el cliente) y el backend (la lógica) está en la misma capa, por lo que no hay separación alguna, la capa que contiene la aplicación se conecta directamente al Sistema Gestor De Bases de datos.
 - Es el modelo menos común a día de hoy ya que las webs y apps móviles por seguridad no soportan esta arquitectura.
 - **3 capas**, tenemos una clara diferenciación entre frontend, backend y SGDB. El Frontend se conecta mediante una api al backend y este utiliza JDBC para acceder al SGDB. Como podemos ver tenemos 3 capas diferenciadas entre ellas.
 - Es el modelo más común, podemos tener diferentes frontends (web, app) conectados al mismo backend ejecutando las mismas acciones. Es mucho más seguro ya que el frontend únicamente contiene la conexión con el backend y no contra el SGDB

ARQUITECTURAS SOPORTADAS POR IDBC

2 CAPAS

-



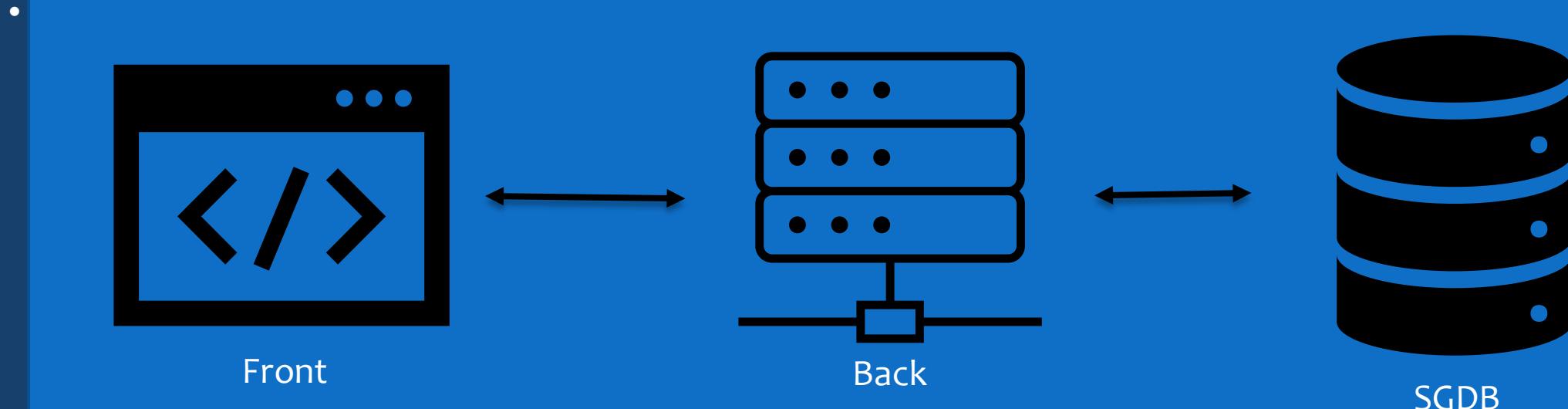
Front+Back
unificados



SGDB

ARQUITECTURAS SOPORTADAS POR IDRC

3 CAPAS



JDBC - PROGRAMACIÓN

- Para conectarnos a una base de datos mediante JDBC deberemos seguir unos pasos específicos o no funcionará. Estamos accediendo a información no gestionada por nosotros, la gestiona el SGDB y por tanto deberemos delegarle esa confianza.
- Pasos:
 1. Importar las librerías necesarias
 2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
 3. Establecer la conexión con el servidor
 4. Ejecutar las sentencias deseadas
 1. Las sentencias se ejecutan gracias a una clase llamada Statement
 2. Los resultados de las sentencias se recogen gracias a una clase llamada ResultSet
 5. Finalizar la conexión con el servidor

JDBC - PROGRAMACIÓN

2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
 - Para saber el driver deberemos mirar la documentación del fabricante

```
String driver = "com.mysql.cj.jdbc.Driver";
try {
    Class.forName(driver);
} catch (ClassNotFoundException ex) {
    System.err.println("No se ha encontrado el Driver: " +
driver);
    System.exit(-1);
}
```

JDBC - PROGRAMACIÓN

2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
 - Para saber el driver deberemos mirar la documentación del fabricante

```
String driver = "com.mysql.cj.jdbc.Driver";  
try {  
    Class.forName(driver);  
} catch (ClassNotFoundException ex) {  
    System.err.println("No se ha encontrado el Driver: " +  
        driver);  
    System.exit(-1);  
}
```



JDBC - PROGRAMACIÓN

3. Establecer conexión con el servidor

```
String bdName = "mo6";
String url = "jdbc:mysql://localhost/" + bdName;
String usuario = "root";
String password = "secret";
Connection connection = null;
try {
    connection = DriverManager.getConnection(url, usuario,
password);
} catch (SQLException ex) {
    System.err.println("Error al conectarse con la BD");
    System.exit(-2);
}
```

JDBC - PROGRAMACIÓN

3. Establecer conexión con el servidor

```
String bdName = "mo6";
String url = "jdbc:mysql://localhost/" + bdName;
String usuario = "root";
String password = "secret";
Connection connection = null;
try {
    connection = DriverManager.getConnection(url,usuario,
password);
} catch (SQLException ex) {
    System.err.println("Error al conectarse con la BD");
    System.exit(-2);
}
```



INSTALACIÓN DE H2 - TRADICIONAL

- Primero de todo es descargar la librería
 - Igual que descargamos el conector de MySQL descargaremos el conector de h2.
 - <http://www.h2database.com/html/download.html>
 - Una vez descargado debemos añadir el jar a nuestro proyecto igual que hicimos con el conector de Mysql
 - Una vez añadido deberemos establecer la ruta de conexión.

INSTALACIÓN DE H2 - TRADICIONAL

- Primero de todo es descargar la librería
 - Igual que descargamos el conector de MySQL
 - <http://www.h2database.com/html/download.html>

```
String bdName = "mo6";
Path path = Path.of(bdName);
String url = "jdbc:h2:" + path.toAbsolutePath();
String usuario = "admin";
String password = "secret";
Connection connection = null;
try {
    connection = DriverManager.getConnection(url, usuario, password);
} catch (SQLException ex) {
    System.err.println("Error al conectarse con la BD");
    ex.printStackTrace();
    System.exit(-2);
}
```

A diferencia de MySQL no hace falta establecer el Driver, y solo con la URL se conectará a la BD.

La primera vez si esta no existe la creará por nosotros con un username y password específico

JDBC - PROGRAMACIÓN

4. Crear objeto Statement para ejecutar sentencias DDL o DML

```
try{  
    Statement estado = connection.createStatement();  
    //sentencias  
    estado.close();  
}catch (SQLException ex){  
    System.err.println("Error al crear el Statement");  
    System.exit(-3);  
}
```

CLASE STATEMENT

- La clase Statement es la que nos permite ejecutar consultas desde Java contra la BD.
- Existen 3 tipos de funciones para lanzar operaciones:
 - `execute()` → Es la más general permite ejecutar cualquier tipo de consulta DDL o DML. Devuelve un booleano
 - CIERTO → La respuesta es un objeto de tipo ResultSet
 - FALSE → Si la respuesta es el número de filas actualizadas o no tiene resultado
 - `executeUpdate()` → Ejecución utilizada para operaciones del tipo DML, devuelve un entero con la cantidad de filas afectadas
 - `executeQuery()` → Ejecución utilizada en operaciones de tipo select, devuelve un objeto ResultSet con los datos de la Respuesta

JDBC – SENTENCIAS DDL O DML

- Las sentencias DDL (Data Definition Language) son aquellas utilizadas para modificar la estructura de la base de datos
 - Crear base de datos, tabla, vista, etc.
 - Actualizar estructuras de datos
 - Eliminar estructuras de datos
- Las sentencias DML (Data Manipulation Language) son aquellas utilizadas para modificar la información almacenada en la base de datos
 - SELECT
 - UPDATE
 - DELETE
 - INSERT

REPASO SQL DDL

```
CREATE TABLE personas
(
    id      INTEGER PRIMARY KEY AUTO_INCREMENT,
    nombre  VARCHAR(30) NOT NULL,
    telefono VARCHAR(9) NOT NULL
);
```

```
CREATE TABLE direcciones
(
    id      INTEGER PRIMARY KEY
    AUTO_INCREMENT,
    id_persona INTEGER NOT NULL,
    direccion VARCHAR(50) NOT NULL
);
```

```
DROP TABLE direcciones;
DROP TABLE personas;
```

```
ALTER TABLE personas
    ADD COLUMN dni varchar(10);

ALTER TABLE personas ADD COLUMN edad INTEGER DEFAULT
18;

ALTER TABLE direcciones
    ADD FOREIGN KEY (id_persona) REFERENCES personas (id);

ALTER TABLE direcciones
    ADD CONSTRAINT PERSONA_ID_FK FOREIGN KEY
(id_persona) REFERENCES personas (id);

ALTER TABLE direcciones DROP CONSTRAINT
PERSONA_ID_FK;
```

EJEMPLO CREAR STATEMENT

```
private static Statement crearEstado(Connection connection) {  
    try {  
        return connection.createStatement();  
    } catch (SQLException ex) {  
        System.out.println("Error al crear el Statement");  
        System.exit(-2);  
    }  
    return null;  
}
```

EJEMPLO CREATE TABLE

```
private static void createTable(Connection connection) {  
    Statement estado = crearEstado(connection);  
    try {  
        String query = "CREATE TABLE personas(id INTEGER PRIMARY KEY  
AUTO_INCREMENT," +  
            " nombre VARCHAR (30) NOT NULL, telefono VARCHAR (9) NOT NULL);";  
        estado.execute(query);  
        System.out.println("Tabla Personas creada correctamente");  
        estado.close();  
    } catch (SQLException ex) {  
        if (ex.getMessage().contains("already exists")) {  
            System.out.println("La tabla ya existe en la BD");  
        } else {  
            System.err.println(ex.getMessage());  
        }  
        System.exit(-3);  
    }  
}
```

EJEMPLO UPDATE TABLE

```
private static void updateTable(Connection connection) {  
    Statement estado = crearEstado(connection);  
    try {  
        String query = "ALTER TABLE personas ADD COLUMN edad int default 18;";  
        estado.execute(query);  
        System.out.println("Tabla personas actualizada");  
    } catch (SQLException ex) {  
        System.err.println("Error actualizar la tabla personas");  
        System.out.println(ex.getMessage());  
        System.exit(-3);  
    }  
}
```

EJEMPLO DELETE TABLE

```
private static void deleteTable(Connection connection) {  
    try {  
        Statement estado = connection.createStatement();  
        String query = "DROP TABLE personas;";  
        estado.execute(query);  
        System.out.println("Tabla personas eliminada  
correctamente");  
    } catch (SQLException ex) {  
        System.out.println("Error al eliminar la tabla personas");  
        System.out.println(ex.getMessage());  
        System.exit(-3);  
    }  
}
```

EJEMPLO DE DELETE TABLE

Como podemos observar es un proceso muy “tedioso”, debemos incluir las propias sentencias SQL dentro de Java por lo que estamos mezclando dos lenguajes en

```
private static void main(String[] args) {  
    try {  
        Statement estado = connection.createStatement();  
        String query = "DROP TABLE personas;";  
        estado.execute(query);  
        System.out.println("Tabla personas eliminada  
correctamente");  
    } catch (SQLException ex) {  
        System.out.println("Error al eliminar la tabla personas");  
        System.out.println(ex.getMessage());  
        System.exit(-3);  
    }  
}
```



EJERCICIOS

- A partir del siguiente ejemplo <https://dev.mysql.com/doc/employee/en/sakila-structure.html>
 1. Crea las tablas:
 1. Empleado
 1. (id, nombre varchar(20), fecha_nacimiento DATE, genero tinyint(1), departamento_id)
 2. Departamento
 1. (id, nombre varchar(20), ubicación varchar(100))
 2. Actualiza la tabla empleado
 1. Añade la columna departamento_id y crea la Foreign Key con departamento
 - 1 departamento muchos empleados
 - 1 empleado 1 departamento

REPASO SQL DML

```
INSERT INTO personas (nombre, telefono, dni)  
VALUES ('ALVARO', '612123123', '32211553-H'),  
      ('TANIA', '689789789', '98776654-J')
```

```
UPDATE personas SET nombre = 'Alvaro Ortega' WHERE id=1;
```

```
DELETE FROM personas WHERE id = 2;
```

```
SELECT * FROM personas;
```

```
select nombre from personas where edad>=18 order by edad;
```

EJEMPLO SELECCIÓN DATOS

```
private static void selectData(Connection connection){  
    int id = 1;  
    String query = "SELECT * from personas where id=" + id +  
";";  
    Statement estado = crearEstado(connection);  
    ResultSet result = null;  
    try{  
        result = estado.executeQuery(query);  
        while(!result.isLast()){  
            result.next();  
            Persona persona = new Persona(  
                result.getString("nombre"),  
                Integer.parseInt(result.getString("telefono")),  
                result.getString("dni"));  
            System.out.println(persona);  
        }  
    } catch(SQLException e){  
        System.out.println("Error al realizar la consulta de  
datos");  
    }  
}
```

```
Persona{ nombre='Alvaro Ortega', telefono=612123123, dni='32211553-H'}
```

RESULTSET

- ResultSet es una clase que contiene la información de toda la consulta de la BD.
- El funcionamiento de ResultSet es muy similar al de un iterador pero con algunas características
 - Un iterador es un objeto que permite recorrer una colección de elementos, no se puede utilizar un bucle tradicional ya que los elementos no están ordenados mediante un índice. El iterador nos proporciona los métodos para comprobar en cada vuelta del bucle si hay más datos (generalmente hasNext()) y recuperar el dato correspondiente (generalmente next()).
 - ResultSet tiene diferentes métodos para obtener los datos:
 - first() → Establece el apuntador en la primera fila recuperada de la BD
 - last() → Establece el apuntador en la última fila recuperada de la BD
 - isLast() → Nos dirá si el apuntador está en la última fila
 - next() → Obtiene los datos de la fila donde se encuentra el apuntador
 - getString(), getInt(), getDate(), etc... → Recupera la información en la columna enviada por parámetros
 - Se le puede enviar el índice de la columna a recuperar
 - Se le puede enviar el nombre de la columna a recuperar

!!!PROBLEMA!!! SQL INJECTION

- Existe una gran vulnerabilidad en SQL cuando se combina código + SQL, por ejemplo en el select anterior:

```
int id = 1;  
String query = "SELECT * from personas where id=" + id + ";";
```

- En cualquier momento, un atacante puede modificar el valor de id por cualquier otro valor, por ejemplo añadiendo más campos o información que no se debería de recuperar y ejecutando de esta manera queries que no deberían ser soportadas

```
String id = "1 OR 1=1";  
String query = "SELECT * from personas where id=" + id + ";";
```

- En este caso la consulta a ejecutar cambia totalmente, ya que $1=1$ es una condición que en SQL siempre es cierta. En el ejemplo es como eliminar la cláusula where ya que el resultado a ejecutar será where id=1 or 1=1 y este será where id=1 or true
- El resultado será recuperar todas las filas y no solo la del usuario con el id 1, rompiendo seguridad de nuestra bd, ya que quizás la consulta únicamente debe recuperar la información del usuario actualmente logueado

SOLUCIONAR SQL INJECTION

- Existe otra clase para evitar estos ataques de inyección de datos.
- En lugar de utilizar Statement deberemos utilizar PreparedStatement. Esta clase evitara que hagamos combinaciones de código Java + código SQL.
- Funciona de forma muy sencilla, las variables serán modificadas por “?” un placeholder que determina que allí habrá una variable que posteriormente será reemplazada
 - Cada uno de ellos tiene un índice, que se utilizará para así poder reemplazar de forma correcta. La primera aparición tendrá el índice 1, la segunda el 2, etc.

EJEMPLO CREAR PREPAREDSTATEMENT

```
private static PreparedStatement crearEstado(Connection connection, String query)
{
    try {
        return connection.prepareStatement(query);
    } catch (SQLException ex) {
        System.out.println("Error al crear el Statement");
        System.exit(-2);
    }
    return null;
}
```

EJEMPLO SELECCIÓN DATOS

```
private static void selectPreparedData(Connection connection){  
    String nombre = "Alvaro Ortega OR 1=1";  
    String query = "SELECT * FROM personas WHERE nombre = ?";  
    PreparedStatement estado = crearEstado(connection, query);  
    try{  
        estado.setString(1, nombre);  
        System.out.println(estado);  
        ResultSet result = estado.executeQuery();  
        int count = 0;  
        while(result.next()){  
            count++;  
            Persona persona = new Persona(  
                result.getString("nombre"),  
                Integer.parseInt(result.getString("telefono")),  
                result.getString("dni"));  
            System.out.println(persona);  
        }  
        if(count == 0)  
            System.out.println("No se han encontrado datos");  
        result.close();  
        estado.close();  
    } catch (SQLException ex){  
        System.out.println("Error en la consulta de los datos");  
        System.out.println(ex.getMessage());  
    }  
}
```

No se han encontrado datos

EJEMPLO SELECCIÓN DATOS

```
private static void selectPreparedData(Connection connection){  
    String nombre = "Alvaro Ortega";  
    String query = "SELECT * FROM personas WHERE nombre = ?";  
    PreparedStatement estado = crearEstado(connection, query);  
    try{  
        estado.setString(1, nombre);  
        System.out.println(estado);  
        ResultSet result = estado.executeQuery();  
        int count = 0;  
        while(result.next()) {  
            count++;  
            Persona persona = new Persona(  
                result.getString("nombre"),  
                Integer.parseInt(result.getString("telefono")),  
                result.getString("dni"));  
            System.out.println(persona);  
        }  
        if(count == 0)  
            System.out.println("No se han encontrado datos");  
        result.close();  
        estado.close();  
    } catch (SQLException ex) {  
        System.out.println("Error en la consulta de los datos");  
        System.out.println(ex.getMessage());  
    }  
}
```

```
Persona{ nombre='Alvaro Ortega', telefono=612123123, dni='32211553-H'}
```

EJEMPLO SELECCIÓN DATOS

- Un punto muy importante en las consultas del tipo PreparedStatement es como se usan los wildcard.
- Al inyectar un parámetro deberemos hacerlo ya con sus opciones prestablecidas, por ejemplo %Alvaro% para que el texto contenga Alvaro sin importar lo que exista al principio o al final del texto.
- Para ello al momento de inyectar el parámetro estableceremos el wildcard deseado.

```
estado.setString(1, "%" + nombre + "%");
```

EJEMPLO SELECCIÓN DATOS

- Un punto muy importante es el uso del wildcard.
- Al injectar un parámetro en la consulta: %Alvaro% para que el resultado sea “Alvaro” o “Alvaro123” o cualquier otro texto.
- Para ello al momento de declarar el parámetro se usa el tipo de dato String.



...estado.setString(1,

Reaso SQL:

Los wildcard son caracteres especiales (comodín) para substituir uno o más caracteres, por ejemplo el "%" es para cualquier texto.

Podeís encontrar más información en:

https://www.w3schools.com/sql/sql_wildcards.asp

EJEMPLO INSERTAR DATOS

```
private static void insertData(Connection connection) {
    Persona persona = new Persona("Alvaro", 612123123, "32211553-H");
    Persona persona2 = new Persona("TANIA", 689789789, "98877654-J");
    Statement estado = crearEstado(connection);
    try {
        String query = "INSERT INTO personas (nombre, telefono, dni) " +
                      "VALUES (" + persona.getNombre() + "," + persona.getTelefono() + ", " + persona.getDni() +
                      "), " +
                      "      (" + persona2.getNombre() + "," + persona2.getTelefono() + ", " + persona2.getDni() +
                      ");";
        estado.executeUpdate(query);
    } catch (SQLException ex) {
        System.out.println("Error al insertar datos");
    }
}
```

EJEMPLO ACTUALIZAR DATOS

```
private static void updateData(Connection connection) {  
    String query = "UPDATE personas SET nombre = ? where id=?";  
    PreparedStatement estado = crearEstado(connection, query);  
    try {  
        estado.setString(1, "Alvaro");  
        estado.setInt(2, 1);  
        estado.executeUpdate();  
        estado.close();  
    } catch (SQLException ex) {  
        System.out.println("Error al acutalizar los datos de la tabla");  
        System.out.println(ex.getMessage());  
    }  
}
```

EJEMPLO ELIMINAR DATOS

```
private static void deleteData(Connection connection){  
    String query = "DELETE FROM personas where id = ?";  
    PreparedStatement estado = crearEstado(connection,query);  
    try{  
        estado.setInt(1, 1);  
        estado.executeUpdate();  
        estado.close();  
    }catch(SQLException ex){  
        System.out.println("Error al eliminar de la BD");  
        System.out.println(ex.getMessage());  
    }  
}
```

EJERCICIOS

- Siguiendo el ejemplo anterior <https://dev.mysql.com/doc/employee/en/sakila-structure.html>
- 3. Inserta datos en las diferentes tablas, todos los datos deben ir por parámetros y debe utilizarse el PreparedStatement para evitar SQLInjection
 1. Tabla departamento (el id se debe de autogenerar)
 1. 1, contabilidad, PBo
 2. 2, marketing, PBo,
 3. 3, RRHH, PB1
 2. Tabla Empleado (el id se debe de autogenerar)
 1. 1, Alvaro, 25/11/1979, 1, 1
 2. 2, Juan, 07/07/2001, 1, 2
 3. 3, Marta, 25/12/1997, 0, 3
 4. 4, Silvia, 01/04/1992, 0, 2
- 4. Recupera los datos de los empleados junto con el nombre del departamento al cual pertenecen
- 5. Recupera todos los usuarios ordenados por fecha de nacimiento, de mayor a menor
- 6. Edita la fecha de nacimiento del empleado con nombre Marta y establece “04/03/2002”
- 7. Elimina el usuario con el id 1

PRACTICA COMPLETA TEMA 2

- Realiza la siguiente practica, puedes preguntarme todas las dudas que tengas.

1. Crea un programa que genere la siguiente estructura de packages y clases:

- Tema2
 - Common
 - Column
 - DDL
 - DML
 - Entities
 - Persona
 - Direccion
 - H2
 - Main
 - MySQL
 - Main

2. Define las clases Persona y Direccion:

- Persona (id, nombre, teléfono,dirección,ArrayList<Direccion>)
- Direccion(id, idPersona,dirección)

3. Define la clase Column, esta representa una columna de BD. Decide que variables debe contener para generar una nueva columna en la creación de una nueva tabla

- Ejemplo: nombre VARCHAR(30) NOT NULL

4. Define la clase DDL, debe contener las funciones:

- Crear tabla, debe recibir por parámetros los datos de la tabla (nombre y columna)

- Crear ambas tablas

- Actualizar tabla

- Añade la FK entre Dirección y Persona

- Eliminar tabla

- Eliminar ambas tablas

5. Define la clase DML, debe contener las funciones

- Consultar datos

- Consultar 1 persona por id

- Consultar todas las personas

- Consultas 1 persona por id junto a sus direcciones

- Insertar datos

- Insertar persona

- Insertar dirección

- Actualizar datos

- Modificar una persona

- Modificar una dirección

- Eliminar datos

- Eliminar una persona y su dirección

6. Debe poder ejecutarse en MySQL y H2, por ello tenemos 2 Main diferentes

TEMA 3: HERRAMIENTAS DE MAPEO OBJETO RELACIONAL

- Concepto de mapeo objeto relacional.
- Características de las herramientas ORM. Herramientas ORM más utilizadas.
- Instalación de una herramienta ORM.
- Estructura de un fichero de mapeo. Elementos, propiedades.
- Clases persistentes.
- Sesiones; estados de un objeto.
- Carga, almacenamiento y modificación de objetos.
- Consultas SQL.

CONCEPTO MAPEO OBJETO RELACIONAL

- En el tema anterior hemos podido observar que vincular los datos recuperados de la base de datos con una clase o insertar un objeto en la base de datos es una tarea muy tediosa y manual
 - Se debe de crear la consulta correspondiente en SQL
 - Se deben vincular las columnas y datos para que todo funcione de manera correcta.
- El mapeo objeto-relacional es una técnica de programación en la cual mediante orientación a objetos podemos hacer las consultas y vinculaciones anteriores de forma muy sencilla.
- Para hacer estas acciones se utilizan unas herramientas llamadas ORM, Object Relational Mapping



CARACTERÍSTICAS ORM

- Los ORM actualmente son muy utilizados y en muchísimas ocasiones han reemplazado a las BD00. Ya que gracias a estas herramientas, conseguimos una orientación a objetos en nuestras BD relacionales.
- Existen muchos ORM en diferentes lenguajes, por ejemplo:
 - Java → Hibernate
 - Android → Room
 - NodeJS → Sequelize
 - ...

CARACTERÍSTICAS ORM

Ventajas

- Orientación a Objetos
- Reducen el tiempo de desarrollo
- Permite reutilizar código de forma muy simple
- Evitan mezclar lenguajes en el mismo código (Java + SQL)
- Sirven para muchos motores de BD
- Lenguajes propios para la realización de consultas

Inconvenientes

- Complejos de aprender
- Múltiples soluciones en el mercado
- En ocasiones la documentación es escasa o incompleta
- Complejidad para realizar consultas avanzadas
- Aplicaciones más pesadas para el procesador por el tratamiento de los datos

HIBERNATE

- Hibernate es el ORM más utilizado de Java
 - Inicialmente funcionaba mediante una configuración XML en la que se definían los mapeos de los datos y así el podía funcionar de forma “automática”
 - Actualmente funciona mediante el uso de anotaciones ”@” en las diferentes clases que se deben utilizar simplificando enormemente esta tarea.
 - En pocos minutos podemos tener nuestra aplicación funcionando de forma correcta.
 - Hibernate tiene su propio lenguaje de programación llamado HQL (Hibernate Query Language), una especie de SQL reducido.

INSTALACIÓN Y CONFIGURACIÓN DE HIBERNATE

- Antes de nada aclarar que hibernate no es un motor de BD es solo un ORM, un traductor de datos.
- 1. Instalar la librería del conector de la base de datos
- 2. Instalar la librería de hibernate, la última es la 5.5 <https://hibernate.org/orm/releases/5.5/>
- 3. Creación de clases entidad (aquellas clases que deseamos que se guarden en la BD)
- 4. Crear fichero de conexión XML. → https://docs.jboss.org/hibernate/orm/5.5/userguide/html_single/Hibernate_User_Guide.html#configurations
- 5. Configurar el fichero de mapeo XML / Las anotaciones para las clases
- 6. Iniciar hibernate
 - 1. Crear un objeto del tipo StandardServiceRegistry
 - 2. Crear un objeto del tipo SessionFactory
 - Este utiliza la clase anterior para funcionar
 - 3. Crear un objeto de tipo Session
 - Se crea mediante el SessionFactory
 - 4. Iniciar transaction
 - Se crea mediante la Session
 - Ejecutar consultas
 - 5. Finalizar transaction
 - 6. Finalizar session

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password"></property >
    <property name ="hibernate.dialect"></property >

    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS

OPCIONALES
(entre otras)

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password</property >
    <property name ="hibernate.dialect"></property >
    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS
OPCIONALES
(entre otras)

jdbc:motor:url

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property>
    <property name ="connection.password</property>
    <property name ="hibernate.dialect"></property >
    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS

OPCIONALES
(entre otras)

Nombre de usuario

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password"></property >
    <property name ="hibernate.dialect"></property >
    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS
OPCIONALES
(entre otras)

Contraseña

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password"></property >
    <property name ="hibernate.dialect"></property >
    <!--
      <property name ="hibernate.hbm2ddl.auto"></property >
      <property name ="hibernate.show_sql"></property >
    -->
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS
OPCIONALES
(entre otras)

Lenguaje que debe
hablar hibernate

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password"></property >
    <property name ="hibernate.dialect"></property >
    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS

OPCIONALES
(entre otras)

Cómo se debe comportar
Hibernate al iniciarse

FICHERO DE CONFIGURACIÓN XML

- El fichero principal de hibernate se llama hibernate.cfg.xml (Hibernate Configuration XML)
- En este fichero se debe establecer la conexión contra el motor de la bd
- Estructura del fichero:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name ="connection.url"></property >
    <property name ="connection.username"></property >
    <property name ="connection.password"></property >
    <property name ="hibernate.dialect"></property >
    <property name ="hibernate.hbm2ddl.auto"></property >
    <property name ="hibernate.show_sql"></property >
  </session-factory>
</hibernate-configuration>
```

OBLIGATORIAS

OPCIONALES
(entre otras)

Mostrar las consultas a la
BD por consola

FICHERO DE CONFIGURACIÓN XML

- El fichero de configuración es un fichero XML.
- En este fichero se definen las configuraciones para la aplicación.
- Estructura:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:h2:m06</property>
        <property name="connection.username">root</property>
        <property name="connection.password">secret</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <property name="hibernate.show_sql">true</property>
        <mapping class="com.codigonline.entities.Persona"/>
    </session-factory>
</hibernate-configuration>
</session-factory>
</hibernate-configuration>
```

HIBERNATE.HBM2DDL.AUTO

- Es una opción que determina como debe comportarse hibernate al iniciarse.
 - Create → Elimina todas las tablas y vuelve a crearlas
 - Create-drop--> Crea todas las tablas al iniciarse y las elimina al detenerse.
 - Update → Actualiza las tablas en caso de ser necesario
 - Validate → Comprueba si las tablas son correctas.
- En otros sistemas como por ejemplo Spring existe un valor adicional “none” consiste en que no haga nada, pero en hibernate nativo no existe.

CREACIÓN DE CLASES ENTIDAD

- Las clases entidad (entity) son aquellas que se convertirán en tablas de la base de datos, por lo que deberemos discernir muy bien entre clases de datos y clases de funcionalidad.
 - Para hacerlo más sencillo las introduciremos en un package llamado entities.
- Las clases entidad deben tener unas propiedades muy concretas:
 - La existencia de un campo que será PRIMARY KEY
 - Dependerá si usamos fichero de mapping xml o anotaciones
 - Constructor sin parámetros
 - Getters y Setter para todas las propiedades. Deben estar bien escritos
 - Getter → get+propiedad
 - `getId()`
 - `obtenerId()`
 - Setter → set+propiedad

EJEMPLO CLASE ENTITY PERSONA

```
public class Persona{  
    int id;  
    String nombre;  
    int edad;  
    Date fechaNacimiento;  
    public Persona(){ }  
    public Persona(String nombre, int edad, Date fechaNacimiento){  
        this.nombre = nombre;  
        this.edad = edad;  
        this.fechaNacimiento = fechaNacimiento;  
    }  
    public int getId(){ return id; }  
    public void setId(int id) { this.id = id; }  
    public String getNombre(){ return nombre; }  
    public void setNombre(String nombre){ this.nombre = nombre; }  
    public int getEdad(){ return edad; }  
    public void setEdad(int edad) { this.edad = edad; }  
    public Date getFechaNacimiento(){ return fechaNacimiento; }  
    public void setFechaNacimiento(Date fechaNacimiento){ this.fechaNacimiento = fechaNacimiento; }  
}
```

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="" table="">
  <id name="" type="" column="">
    <generator class="identity"/>
  </id>
  <property name="" type="" column="" length="" />
</class>
</hibernate-mapping>
```

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="" table="">
  <id name="" type="" column="">
    <generator class="identity"/>
  </id>
  <property name="" type="" column="" length="" />
</class>
</hibernate-mapping>
```

DTD

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="" table="">
    <id name="" type="" column="">
      <generator class="identity"/>
    </id>
    <property name="" type="" column="" length="" />
  </class>
</hibernate-mapping>
```

ETIQUETA PRINCIPAL

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="" table="">
    <id name="" type="" column="">
      <generator class="identity"/>
    </id>
    <property name="" type="" column="" length="" />
  </class>
</hibernate-mapping>
```

Clase que se desea
mapear

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="" table="">
    <id name="" type="" column="">
      <generator class="identity"/>
    </id>
    <property name="" type="" column="" length="" />
  </class>
</hibernate-mapping>
```

Atributo que hará de PK

FICHEROS DE MAPING

- Originalmente hibernate utilizaba unos ficheros de mapeo de archivos con la extensión hbm.xml (Hibernate Mapping XML)
- En estos se deben definir las diferentes propiedades que vincularán la clase con la tabla.
 - Se establecerá la relación atributo a atributo
- Estructura fichero mapping

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="" table="">
  <id name="" type="" column="">
    <generator class="identity"/>
  </id>
  <property name="" type="" column="" length="" />
</class>
</hibernate-mapping>
```

Columnas de la tabla

GENERATOR

- El generator en el id es como se va a crear esta PK en la base de datos. Existen diferentes tipos
 - AUTO → Establecerá un valor basado en el tipo del atributo que puede ser numérico o UUID (Universally Unique Identifier), para valores numéricos utilizará una secuencia, para valores UUID utilizará un UUIDGenerator
 - IDENTITY → Quiere decir que el id será AUTO_INCREMENT
 - SEQUENCE → Seguirá una secuencia que se debe proporcionar, es similar al AUTO_INCREMENT pero nos permite configurar la secuencia.
 - TABLE → Utiliza una tabla oculta que contiene valores de identificadores y se combinan con el nombre de la tabla, es el más ineficiente de todos.

EJEMPLO MAPEO CLASE PERSONA

```
public class Persona {  
    int id; ←  
    String nombre; ←  
    int edad; ←  
    Date fechaNacimiento; ←  
    ...  
}
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<!DOCTYPE hibernate-mapping PUBLIC  
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
    <class name="com.codigonline.entities.Persona" table="personas">  
        <id name="id" type="integer" column="id">  
            <generator class="identity"/>  
        </id>  
        <property name="nombre" type="string" column="nombre" length="20"/>  
        <property name="edad" type="integer" column="edad" length="3"/>  
        <property name="fechaNacimiento" type="date" column="fecha_nacimiento"/>  
    </class>  
</hibernate-mapping>
```

ANOTACIONES MAPPING

- Una vez acostumbrados al uso de ficheros de mapeo, es muy sencillo crear nuestra primera aplicación pero tiene ciertos inconvenientes
 - Si se modifica la clase debemos modificar el fichero (doble trabajo)
 - Se deben definir de forma correcta todos los atributos
 - Es fácil dejarse un atributo sin mapear (dará error)
 - Si la aplicación va creciendo mucho es cada vez más complejo mantener los ficheros.
- Es por ello que hibernate se actualizó, eliminando los ficheros de mapeo y utilizando anotaciones.
- Las anotaciones son unas pequeñas instrucciones que se ponen dentro del código dotando a nuestra clase Entity de la información necesaria para ser mapeada. Las anotaciones más importantes son:
 - `@Entity`
 - `@Table`
 - `@Id`
 - `@GeneratedValue`
 - `@Column`

EJEMPLO MAPEO CLASE PERSONA

```
public class Persona{  
    int id;  
    String nombre;  
    int edad;  
    Date fechaNacimiento;  
    ...  
}
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<!DOCTYPE hibernate-mapping PUBLIC  
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
        "http://www.hibernate.org/dtd/hibernate-mapping-  
        3.0.dtd">  
<hibernate-mapping>  
<class name="com.codigonline.entities.Persona"  
table="personas">  
    <id name="id" type="integer" column="id">  
        <generator class="identity"/>  
    </id>  
    <property name="nombre" type="string"  
column="nombre" length="20"/>  
    <property name="edad" type="integer" column="edad"  
length="3"/>  
    <property name="fechaNacimiento" type="date"  
column="fecha_nacimiento"/>  
</class>  
</hibernate-mapping>
```

```
@Entity  
 @Table(name = "personas")  
 public class Persona{  
     @Id  
     @GeneratedValue(strategy = GenerationType.IDENTITY)  
     int id;  
     @Column(length = 20)  
     String nombre;  
     @Column(length = 3)  
     int edad;  
     @Column(name = "fecha_nacimiento")  
     Date fechaNacimiento;  
     ...  
 }
```

FICHEROS DE MAPING

- Finalmente una vez mapeadas las clases/tablas ya sea en formato fichero o con anotaciones deberemos añadir el mapeo al fichero hibernate.cfg.xml
 - Mapeo con fichero:
`<mapping resource="persona.hbm.xml"/>`
 - Mapeo con anotaciones:
`<mapping class="com.codigonline.entities.Persona"/>`

INICIAR HIBERNATE

- Una vez creada toda la configuración de Hibernate ya podemos inicializarlo en nuestra aplicación.
- Recordando los pasos:
 - Crear un StandardServiceRegistry el cual determinará la ubicación del fichero hibernate.cfg.xml
 - Crear un SessionFactory el cual incluirá la conexión contra la BD
 - Crear una Session contra la BD (abrir la conexión contra la BD)
- Para todos estos pasos deberemos ir a la documentación oficial, ya que son pasos muy sencillos pero hay que entenderlos

EJEMPLO INICIALIZACIÓN HIBERNATE

```
final StandardServiceRegistry registro = new StandardServiceRegistryBuilder().configure().build();
final SessionFactory sessionFactory = new MetadataSources(registro).buildMetadata().buildSessionFactory();
final Session session = sessionFactory.openSession();
```

EJEMPLO INICIALIZACIÓN HIBERNATE

```
final StandardServiceRegistry registro = new StandardServiceRegistryBuilder().build();
final SessionFactory sessionFactory = new MetadataSources(registro).buildMetadataSources().get();
final Session session = sessionFactory.openSession();
```

CÓDIGO LOCURA

Solo con ver ese código
me mareo...



CLASES DE INICIALIZACIÓN

- **StandardServiceRegistry**: Clase encargada de buscar el fichero de configuración hibernate, analizarlo y comprobar que sea correcto.
- **SessionFactory**: Clase que gracias al StandardServiceRegistry puede crear una conexión contra la BD, además incluye todas las propiedades opcionales para que la aplicación se comporte como nosotros deseamos.
- **Session**: Clase que gracias al SessionFactory nos permite abrir las conexiones y enviar/recibir datos de la BD

INSERCIones CON HIBERNATE

- Una vez realizado todo el mapeo y configuración inicial realizar inserciones es muy sencillo.
- Para guardar deberemos utilizar el método save del objeto session, este se encargará de guardar toda la información necesaria en la BD.
- Un punto muy importante es que todas las operaciones deben ir en transacciones por lo que tendremos 3 pasos
 1. Abrir transacción
 2. Realizar operaciones
 3. Finalizar transacción

```
//ABRIMOS TRANSACCION
session.beginTransaction();

Persona persona = new Persona("Alvaro", 27, Persona.toDate("25/11/1993"));
//GUARDARMOS
session.save(persona);

//CERRAMOS TRANSACCION
session.getTransaction().commit();
```

INSERCIones CON HIBERNATE

- Una vez realizado todo el mapeo y configuración inicial realizar inserciones es muy sencillo.
- Para guardar deberemos utilizar el método save del objeto session, este se encargará de guardar toda la información necesaria en la BD.
- Un punto muy importante es que todas las operaciones deben ir en transacciones por lo que

Un elemento muy curioso es que pese a que no le pasamos ningún id al guardar el objeto persona, si después de guardarla lo mostramos por pantalla este tendrá el id actualizado

```
//ABRIMOS TRANSACCION
session.beginTransaction();

Persona persona = new Persona("Alvaro", 27, Persona.toDate("25/11/1993"));
//GUARDARMOS
session.save(persona);

//CERRAMOS TRANSACCION
session.getTransaction().commit();
```

COMPARATIVA JDBC VS HIBERNATE

COMPARATIVA JDBC VS HIBERNATE

```
//ABRIMOS TRANSACCION  
session.beginTransaction();  
  
Persona persona = new Persona("Alvaro", 27, Persona.toDate("25/11/1993"));  
//GUARDARMOS  
session.save(persona);  
  
//CERRAMOS TRANSACCION  
session.getTransaction().commit();
```

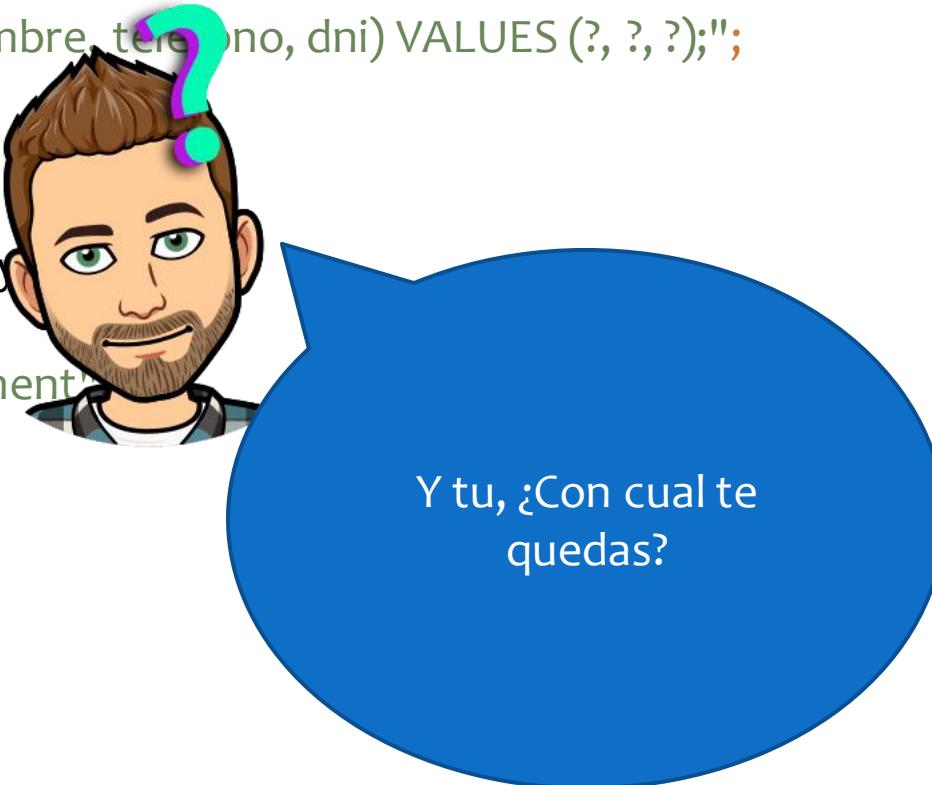
COMPARATIVA JDBC VS HIBERNATE

```
String query = "INSERT INTO personas (nombre, telefono, dni) VALUES (?, ?, ?);";

PreparedStatement estado;
try {
    estado = connection.prepareStatement(query);
} catch (SQLException ex) {
    System.out.println("Error al crear el Statement");
    ex.printStackTrace();
    System.exit(-2);
}
try {
    estado.setString(1, "ALVARO");
    estado.setString(2, "123456789");
    estado.setString(3, "65544334-J");
    estado.executeUpdate();
} catch (SQLException ex) {
```

COMPARATIVA JDBC VS HIBERNATE

```
String query = "INSERT INTO personas (nombre, telefono, dni) VALUES (?, ?, ?);";  
  
PreparedStatement estado;  
try {  
    estado = connection.prepareStatement(query);  
} catch (SQLException ex) {  
    System.out.println("Error al crear el Statement");  
    ex.printStackTrace();  
    System.exit(-2);  
}  
try {  
    estado.setString(1, "ALVARO");  
    estado.setString(2, "123456789");  
    estado.setString(3, "65544334-J");  
    estado.executeUpdate();  
} catch (SQLException ex) {
```



Y tu, ¿Con cual te quedas?

ACTUALIZACIONES CON HIBERNATE

- Para realizar las actualizaciones disponemos de dos mecanismos diferentes
 - Utilizar la función update
 - Será el mecanismo utilizado de forma “normal”
 - Utilizar la función save
 - Esta función en el update está reservada a cuando hemos recuperado un objeto previamente de la BD ya sea porque hemos hecho un select o una inserción nueva. Si nosotros le ponemos el ID de forma manual no funcionará
 - Si recuperamos y actualizamos podremos utilizar el save.
 - Mi recomendación es utilizar siempre el update

```
session.beginTransaction();
Persona p = new
Persona(1,"asd",12,Persona.toDate("01/01/2001"));
session.update(p);
//CERRAMOS TRANSACCION
session.getTransaction().commit();
```

ELIMINACIONES CON HIBERNATE

- Para realizar las eliminaciones deberemos recurrir al método delete

```
session.beginTransaction();
Persona p = new
Persona(1,"asd",12,Persona.toDate("01/01/2001"));
session.delete(p);
//CERRAMOS TRANSACCION
session.getTransaction().commit();
```

HQL

- Hemos visto que las inserciones y eliminaciones con hibernate son increíblemente sencillas, solo debemos pasarle el objeto a eliminar.
- ¿pero que pasa con las búsquedas?
 - Para la realización de consultas “select” deberemos hacer uso del lenguaje propio de Hibernate: HQL (Hibernate Query Language)
 - Este lenguaje es muy similar a SQL pero es un poco más reducido. La gran mayoría de consultas veremos que son iguales

HQL

- Las sentencias HQL se ejecutan creando una query desde el objeto Session de hibernate:
`session.createQuery("")`
- Aunque aquí vemos un solo argumento, el String, esta operación puede recibir dos argumentos distintos:
 1. La query HQL a ejecutar
 2. La clase en la que se desea transformar los datos.

HQL – RECUPERAR TODOS LOS DATOS

```
Query<Persona> queryPersonas = session.createQuery("FROM Persona ", Persona.class);
```

Clase a la cual se
deben de transformar
los resultados

Consulta HQL correspondiente a:
SELECT * FROM personas

HQL – RECUPERAR TODOS LOS DATOS

```
Query<Persona> queryPersonas = session.createQuery("FROM Persona ", Persona.class);
```



Pero con esto no hay suficiente,
deberemos recuperar los
resultados de esta consulta

Clase a la cual se
deben de transformar
los resultados

Consulta HQL correspondiente a:
SELECT * FROM personas

HQL – RECUPERAR TODOS LOS DATOS

```
Query<Persona> queryPersonas = session.createQuery("FROM Persona ", Persona.class);
```

```
List<Persona> personas = queryPersonas.getResultList();
```



Pero con esto no hay suficiente,
deberemos recuperar los
resultados de esta consulta

Clase a la cual se
deben de transformar
los resultados

Consulta HQL correspondiente a:
SELECT * FROM personas

HQL – RECUPERAR UN RESULTADO

```
Query<Persona> queryPersonas = session.createQuery("FROM Persona WHERE id = :id", Persona.class);
```

Consulta HQL correspondiente a:
SELECT * FROM personas where id = ?

Clase a la cual se
deben de transformar
los resultados

¡Parámetro que se debe
inyectar!
Recordad el SQL INJECTION

HQL – RECUPERAR UN RESULTADO

```
Query<Persona> queryPersonas = session.createQuery("SELECT p FROM Persona p WHERE id = :id", Persona.class);
```

El nombre del parámetro
debe de coincidir

```
queryPersonas.setParameter("id", p.getId());  
queryPersonas.getSingleResult();
```



Ahora debemos realizar 2 operaciones:
1. Inyectar parámetro
2. Recuperar un solo resultado

RELACIONES 1-N N-M

- Hasta ahora hemos visto como realizar consultas simples, pero, ¿qué pasa con relaciones de varias tablas?
- De la misma forma que SQL soporta relaciones entre tablas, hibernate también debe soportarlo.
 - Relaciones 1-N se llaman **OneToMany/ManyToOne** (dependerá de donde estemos observando la relación)
 - Relaciones N-M se llaman **ManyToMany**.
- Deberemos de configurar el mapeo utilizando la que necesitemos.

1-N

- Vamos a observarlo con un ejemplo (1-N).
- Tenemos Personas y Empresas. Una persona solo puede trabajar en una empresa, una empresa tiene N Personas
 - Persona
 - ManyToOne, muchas personas 1 empresa
 - En este lado de la relación también podrían ser OneToOne, 1 Persona 1 Empresa
 - Empresa
 - OneToMany, 1 empresa muchas personas

1-N

- A continuación deberemos:
 1. Crear las entidades en nuestra aplicación
 2. Crear las variables correspondientes a la unión
 1. La clase Persona tendrá una variable de tipo Empresa
 2. La clase Empresa tendrá una lista de tipo Persona
 3. Definir el mapeo
 - Puede ser con anotaciones o XML dependiendo como lo estemos realizando.

```
private Empresa empresa;  
private List<Persona> personas;
```

MAPEO ANOTACIONES

CLASE PERSONA

```
@OneToOne()  
private Empresa empresa;
```

```
@ManyToOne()  
private Empresa empresa;
```

CLASE EMPRESA

```
@OneToMany()  
private List<Persona> personas;
```

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MappedBy

- Indica cual es el atributo que representa la FK en la otra entidad

Cascade

- Como debe comportarse al actualizar o eliminar un elemento.

Fetch

- Como debe de recuperarse la información
 - EAGER → Obtener los datos al momento
 - LAZY → Obtener los datos bajo demanda

OrphanRemoval

- Que hacer con los hijos de una relación cuando esta se elimina

MAPEO XML

Persona.hbm.xml

```
<many-to-one name="empresa" cascade="all" column="empresa_id" class="com.codigonline.entities.Empresa"/>
```

Empresa.hbm.xml

```
<set name="personas" table="personas">
  <key>
    <column name="empresa_id" not-null="true"/>
  </key>
  <one-to-many class="com.codigonline.entities.Persona"/>
</set>
```

MAPEO XML

Persona.hbm.xml

```
<many-to-one name="empresa" cascade="all" column="empresa_id" class="com.codigonline.entities.Empresa"/>
```

Empresa.hbm.xml

```
<set name="personas" table="personas">
  <key>
    <column name="empresa_id" not-null="true"/>
  </key>
  <one-to-many class="com.codigonline.entities.Persona"/>
</set>
```



Los ficheros XML son
mucho más rígidos y no
admiten OneToOne con
ManyToOne

HQL –

RECUPERAR TODOS LAS PERSONAS DE UNA EMPRESA RECUPERA LA EMPRESA DE UNA PERSONA CONTAR LOS TRABAJADORES DE UNA EMPRESA

```
List<Persona> personas = session.createQuery(  
    "select e.personas " +  
        "from Empresa e " +  
        "where e.id=1").getResultSet();  
System.out.println(personas);
```

```
Empresa empresa = session.createQuery(  
    "select p.empres" +  
        "from Persona p left join Empresa e " +  
        "on p.empres.id = e.id " +  
        "where e.id=1", Empresa.class).getSingleResult();  
System.out.println(empresa);
```

```
Long trabajadores = session.createQuery(  
    "select count(*) as Trabajadores " +  
        "FROM Persona p left join " +  
        "Empresa e ON p.empres.id = e.id " +  
        "where e.nombre='CodigOnline'",  
    Long.class).getSingleResult();  
System.out.println(trabajadores);
```

RELACIONES N-M

- Las relaciones N-M son más complicadas ya que deberemos de definir la tabla intermedia y sus relaciones
- Este paso solo se hará una vez en la entidad que nosotros deseemos.
 - Generalmente elegiremos la más importante de las dos

MAPEO ANOTACIONES

CLASE PERSONA

```
@ManyToMany(mappedBy = "personas")
private Set<Empresa> empresas;
```

CLASE EMPRESA

```
@ManyToMany
@JoinTable(
    name="Persona_Empresa",
    joinColumns = {@JoinColumn(name="empresa_id")},
    inverseJoinColumns = {@JoinColumn(name="persona_id")})
private Set<Persona> personas;
```

MAPEO ANOTACIONES

CLASE PERSONA

```
@ManyToMany(mappedBy = "personas")
private Set<Empresa> empresas;
```

Donde se define la relación (indica el atributo de la otra relación)

CLASE EMPRESA

```
@ManyToMany
@JoinTable(
    name="Persona_Empresa",
    joinColumns = {@JoinColumn(name="empresa_id")},
    inverseJoinColumns = {@JoinColumn(name="persona_id")})
private Set<Persona> personas;
```

MAPEO ANOTACIONES

CLASE PERSONA

```
@ManyToMany(mappedBy = "personas")  
private Set<Empresa> empresas;
```

Donde se define la relación (indica el atributo de la otra relación)

CLASE EMPRESA

```
@ManyToMany  
 @JoinTable(  
     name="Persona_Empresa",  
     joinColumns = {@JoinColumn(name="empresa_id")},  
     inverseJoinColumns = {@JoinColumn(name="persona_id")}  
)  
private Set<Persona> personas;
```

Cómo se llamará la tabla intermedia

Nombre de la columna correspondiente a empresa

Nombre de la columna correspondiente a persona

MAPEO XML

- Es muy similar al mapeo OneToMany, pero unos matices correspondientes a:
 - La nueva relación
 - Las columnas resultantes

Empresa.hbm.xml

```
<set name="personas" table="Persona_Empresa">
  <key>
    <column name="empresa_id" not-null="true"/>
  </key>
  <many-to-many class="com.codigonline.entities.Persona">
    <column name="persona_id"/>
  </many-to-many>
</set>
```

Persona.hbm.xml

```
<set name="empresas" table="Persona_Empresa">
  <key>
    <column name="persona_id" not-null="true"/>
  </key>
  <many-to-many class="com.codigonline.entities.Empresa">
    <column name="empresa_id"/>
  </many-to-many>
</set>
```

TEMA 5: BASES DE DATOS XML.

- Bases de datos nativas XML.
- Estrategias de almacenamiento.
- Establecimiento y cierre de conexiones.
- Colecciones y documentos.
- Creación y borrado de colecciones; clases y métodos.
- Añadir, modificar y eliminar documentos; clases y métodos.
- Realización de consultas; clases y métodos.
- Tratamiento de excepciones.

BASES DE DATOS XML

- Hasta el momento hemos trabajado con BD relacionales con el lenguaje SQL, pero no son las únicas que existen.
- Las bases de datos XML en lugar de almacenar datos en una estructura de tabla, almacenan XMLs y dentro de estos se almacena la información.
 - Las BD XML aparte de gestionar los ficheros almacenados tienen total acceso a su contenido y por lo tanto se pueden realizar operaciones sobre sus datos.
- Un motor de base de datos XML debe admitir las tecnologías propias de XML como por ejemplo:
 - Xquery → Lenguaje de consulta de ficheros XML
 - Xpath → Lenguaje que permite recorrer y procesar el documento XML
 - XSLT → Lenguaje de transformación de XML, permite transformar documentos XML en otros documentos XML (cambiar su estructura)
- Adicionalmente debe cumplir con las propiedades ACID

ACID

- Atomicity (Atomicidad): Debe tratar los elementos de forma atómica, por ejemplo una transacción la trata como un todo.
- Consistency (Concurrencia): Debe partir de un estado de la base de datos correcto (consistente) y dejar la base de datos en el mismo estado. Sin error alguno.
- Isolation (Aislamiento): Una operación no debe afectar a otra operación que se esté ejecutando al mismo tiempo (concurrencia).
- Durability (Durabilidad): Una operación finalizada de forma correcta no debe poder ser sobrescrita por otra operación en ese mismo momento.

ACID

- Atomicity (Atomicidad): Debe tratar los elementos de forma atómica, por ejemplo una transacción la trata como un todo.
- Consistency (Concurrencia): Debe partir de un estado de la base de datos correcto (consistente) y dejar la base de datos en el mismo estado. Sin error alguno.
- Isolation (Aislamiento): Una operación no debe afectar a otra operación que se esté ejecutando al mismo tiempo (concurrencia).
- Durability (Durabilidad): Una operación finalizada de forma correcta no debe poder ser sobrescrita por otra operación en ese mismo momento.

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

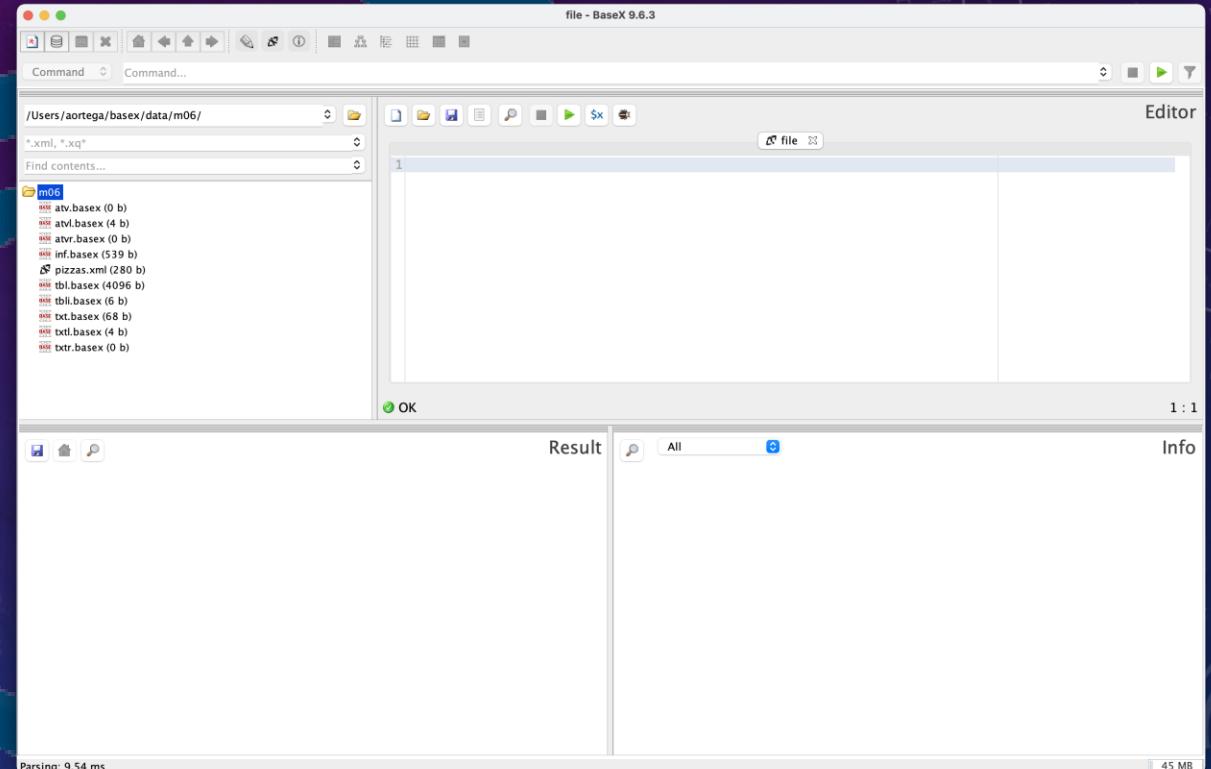
INSTALACIÓN MOTOR BD XML

- Existen diferentes motores. Lo más utilizados son BaseX y eXist siendo el primero mucho más completo. Ambos están escritos en Java y permiten la utilización de XPath, XQuery y XSLT
- Instalación de BaseX → <https://baseX.org/download/>
- Instalación de eXist → <http://exist-db.org/exist/apps/homepage/index.html>
- El motor utilizado no va a afectar al transcurso de este tema dado que las instrucciones serán las mismas

Denominación	Licencia	Lenguaje	XQuery 3.0	XQuery Update	XQuery Full Text	Extensiones EXPath	Extensions EXQuery	XSLT 2.0
BaseX	Licencia BSD	Java	✓ Sí	✓ Sí	✓ Sí	✓ Sí	✓ Sí	✓ Sí
eXist	Licencia LGPL	Java	✗ Parcial	Propietario	Propietario	✓ Sí	✓ Sí	✓ Sí
MarkLogic Server	Comercial	C++	✗ Parcial	Propietario	Propietario	✗ No	✗ No	✓ Sí
Sedna	Licencia Apache	C++	✗ No	✓ Sí	✓ Sí	✗ No	✗ No	✗ No
plataforma 28msec	Motor: Licencia Apache , Almacenamiento: Comercial	C++	✗ Parcial ¹⁰	✓ Sí	✓ Sí	✓ Sí	✓ Sí	✗ No
MonetDB/XQuery	Licencia MonetDB (tipo Mozilla)	C	?	?	?	?	?	?

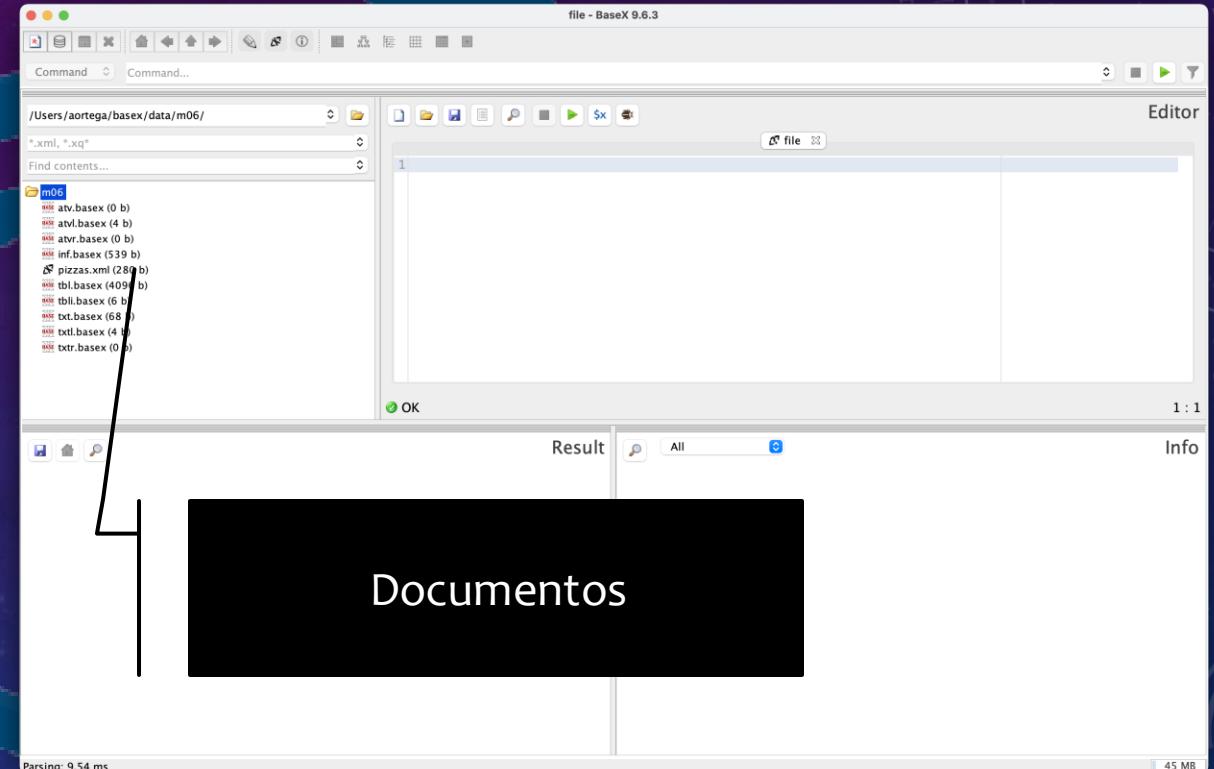
BASEX

- Es un software muy sencillo. Para ejecutarlo únicamente deberemos ejecutar el fichero .jar descargado.
 - ¡Atención! Debemos de tener Java instalado.
- Tiene una UI muy sencilla y en el podremos ejecutar las consultas necesariasnuestro para propósito.



BASEX

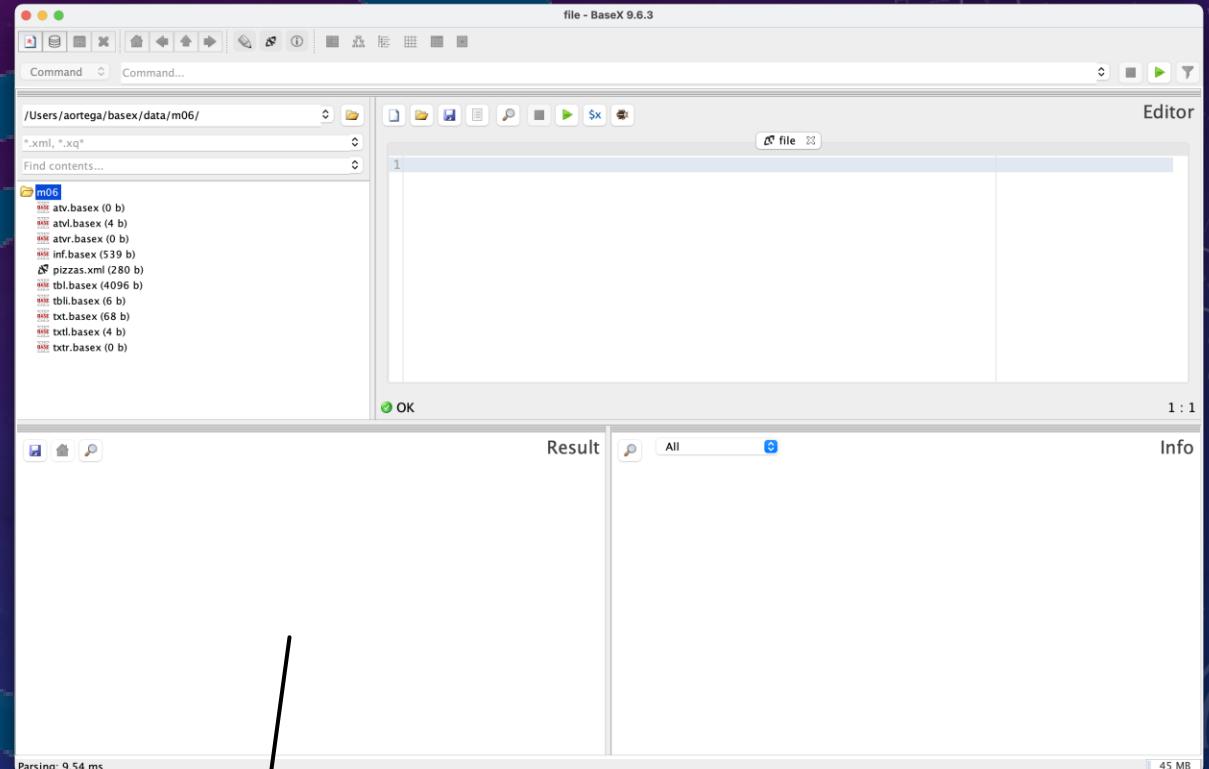
- Es un software muy sencillo. Para ejecutarlo únicamente deberemos ejecutar el fichero .jar descargado.
 - ¡Atención! Debemos de tener Java instalado.
- Tiene una UI muy sencilla y en el podremos ejecutar las consultas necesariasnuestro para propósito.



Documentos

BASEX

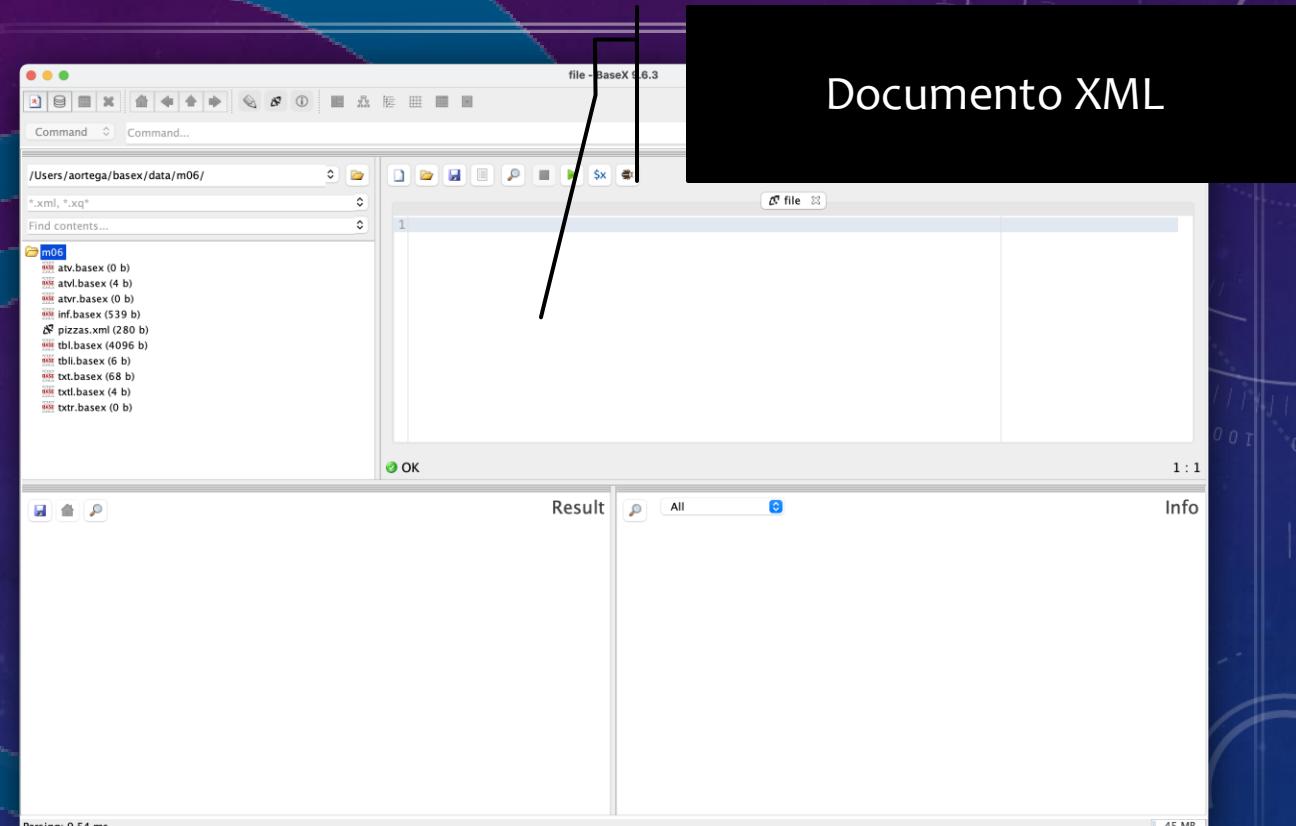
- Es un software muy sencillo. Para ejecutarlo únicamente deberemos ejecutar el fichero .jar descargado.
 - ¡Atención! Debemos de tener Java instalado.
- Tiene una UI muy sencilla y en el podremos ejecutar las consultas necesariasnuestro para propósito.



Resultados

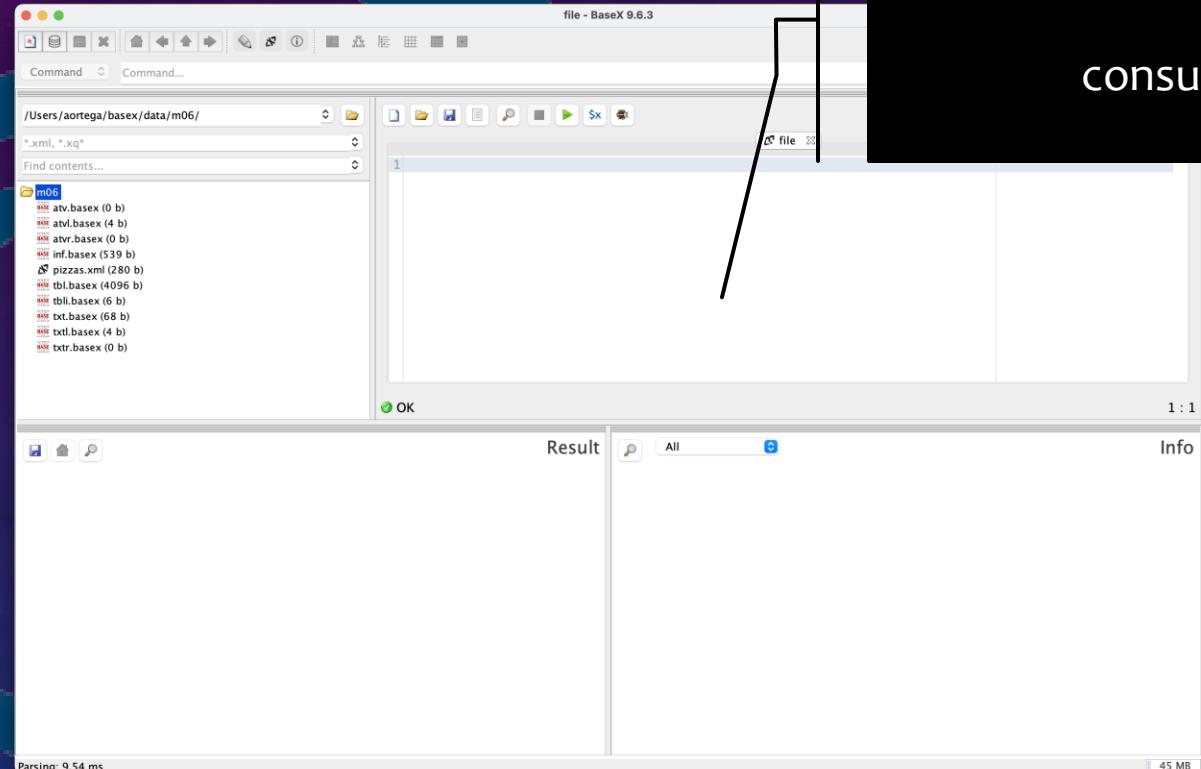
BASEX

- Es un software muy sencillo. Para ejecutarlo únicamente deberemos ejecutar el fichero .jar descargado.
 - ¡Atención! Debemos de tener Java instalado.
- Tiene una UI muy sencilla y en el podremos ejecutar las consultas necesariasnuestro para propósito.



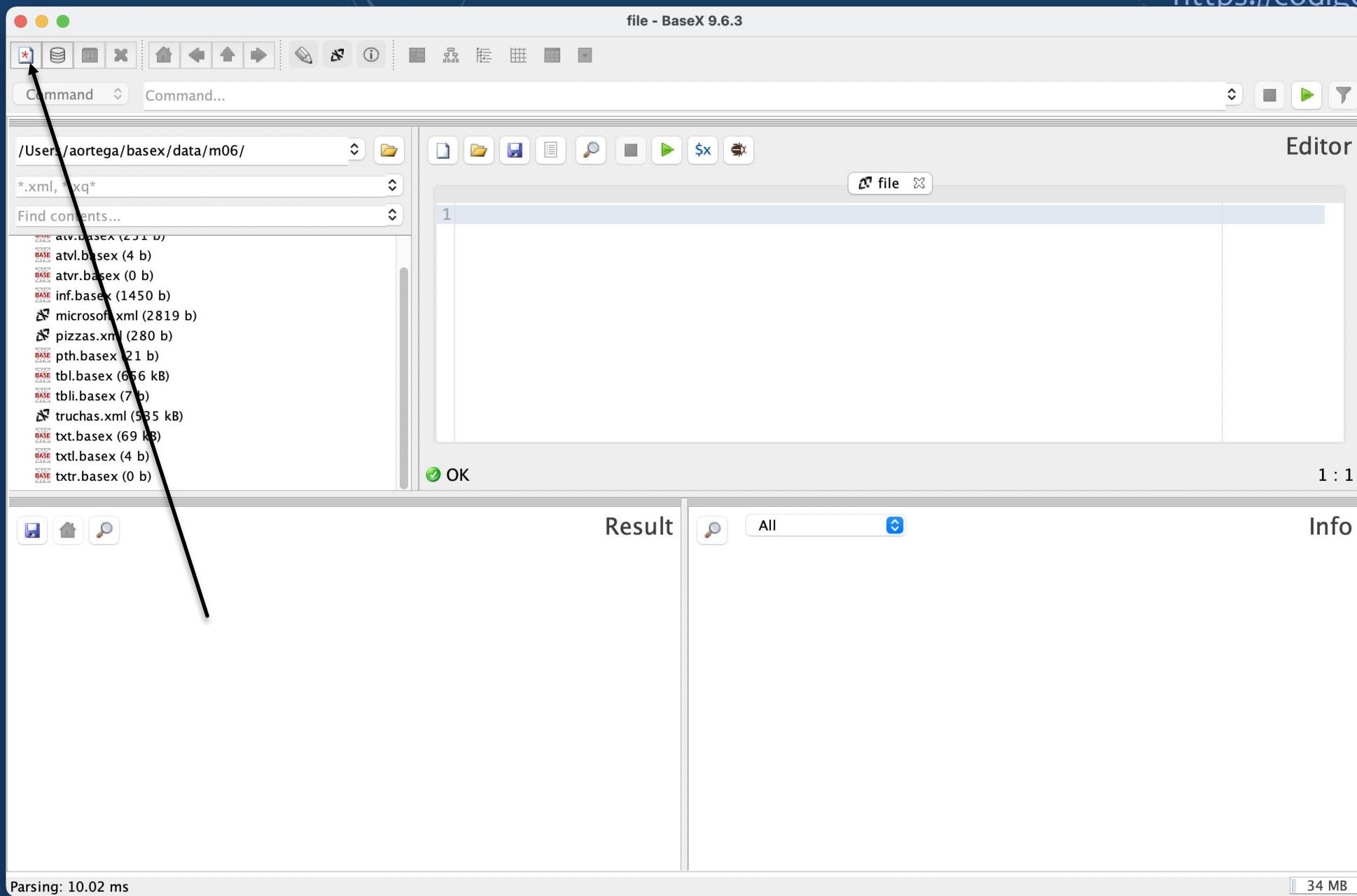
BASEX

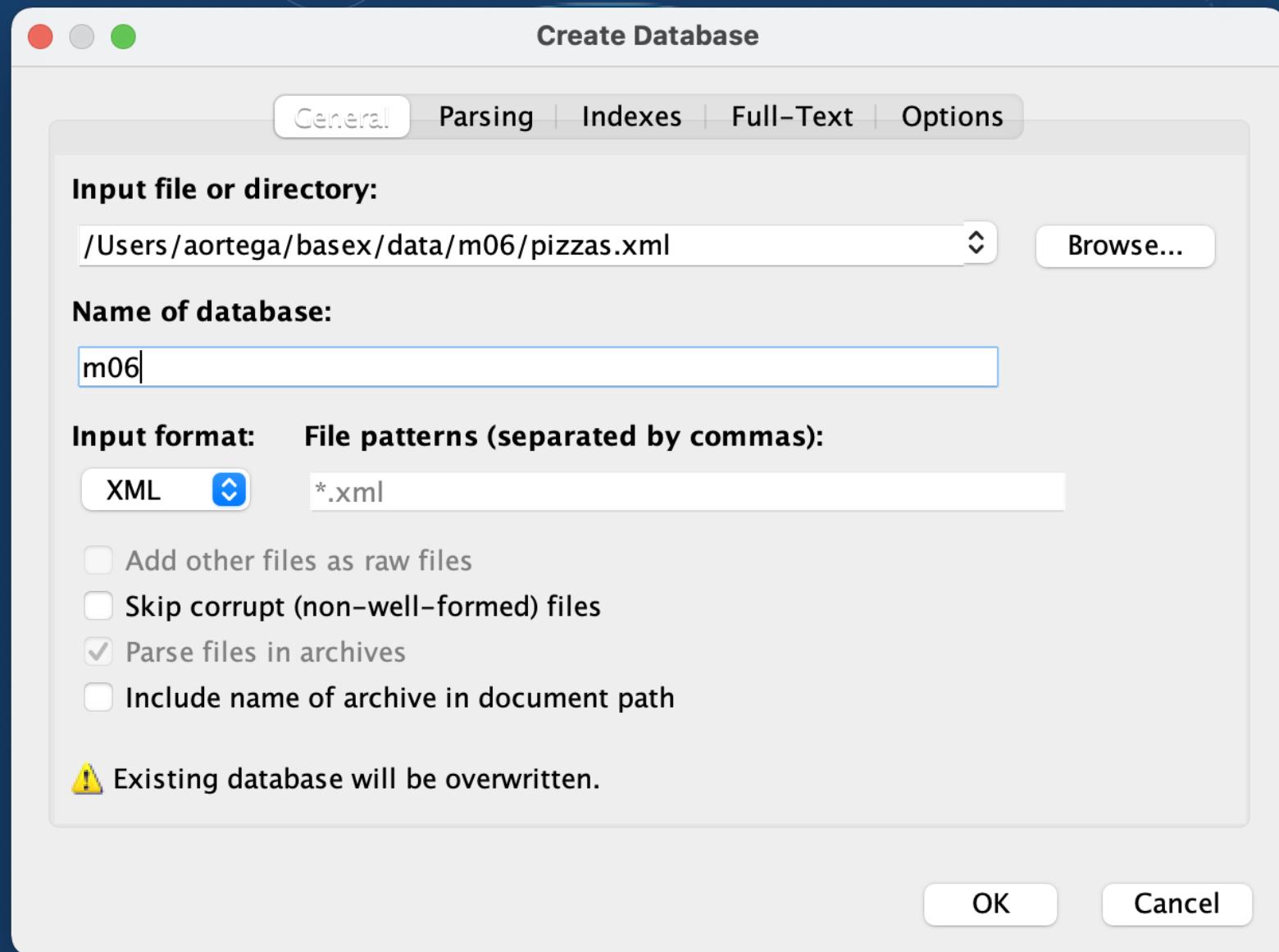
- Es un software muy sencillo. Para ejecutarlo únicamente deberemos ejecutar el fichero .jar descargado.
 - ¡Atención! Debemos de tener Java instalado.
- Tiene una UI muy sencilla y en el podremos ejecutar las consultas necesariasnuestro para propósito.

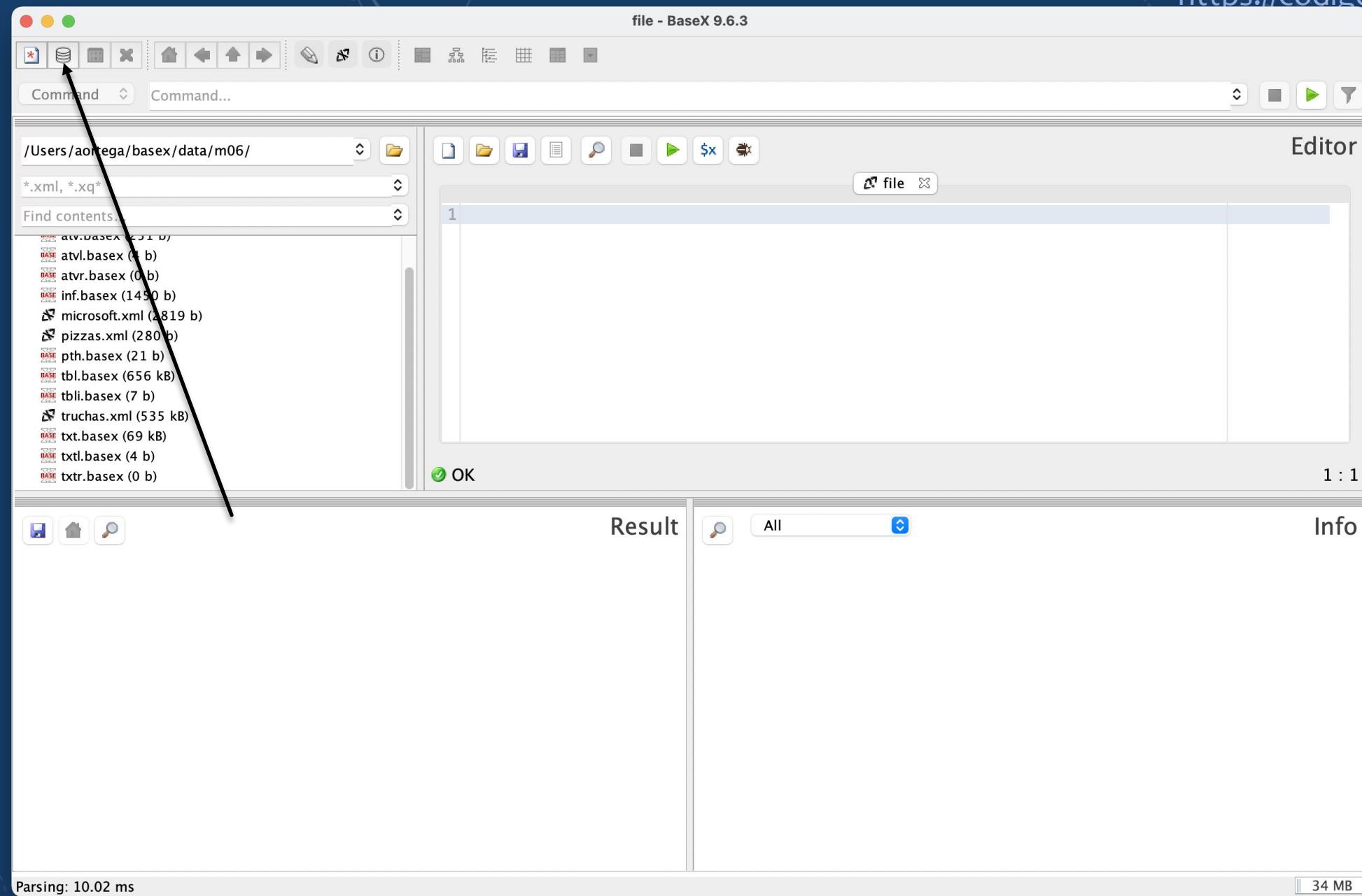


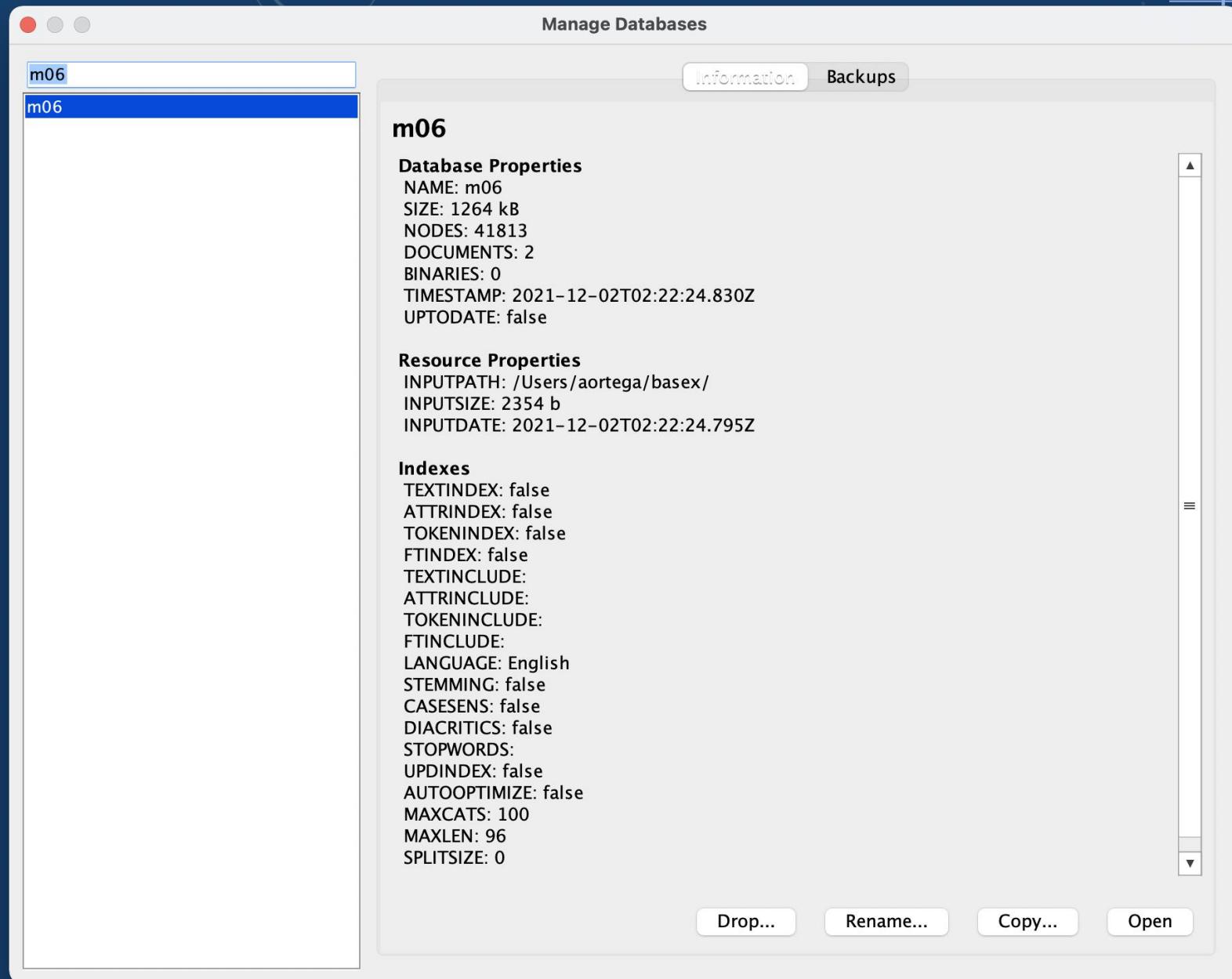
CONFIGURAR BASEX

- Hay que realizar unos pasos muy sencillos:
 - Crear base de datos
 - Añadir documentos
 - Realizar consultas









file [m06] - BaseX 9.6.3

2 Results

/Users/aortega/baseX/data/m06/

Find... Context: db:open("m06")

Editor

OK

1 : 1

2 Results, 584 kB

Result

```
<PurchaseOrders>
  <PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
    <Address Type="Shipping">
      <Name>Ellen Adams</Name>
      <Street>123 Maple Street</Street>
      <City>Mill Valley</City>
      <State>CA</State>
      <Zip>10999</Zip>
      <Country>USA</Country>
    </Address>
    <Address Type="Billing">
      <Name>Tai Yee</Name>
      <Street>8 Oak Avenue</Street>
      <City>Old Town</City>
      <State>PA</State>
    </Address>
  </PurchaseOrder>
</PurchaseOrders>
```

Info

Total Time: 21.26 ms

Command:
OPEN m06

Result:
Database 'm06' was opened in 17.11 ms.

OPEN m06: 21.26 ms

173 MB

The screenshot shows the BaseX 9.6.3 application window. The toolbar at the top has several icons, with the fourth one from the left highlighted by a black arrow pointing down to the search results below. The main pane displays a list of XML files in the 'm06' database. The 'Result' pane shows the XML query results, which include two purchase orders. The 'Info' pane at the bottom provides performance metrics: 'Total Time: 21.26 ms' and 'Database 'm06' was opened in 17.11 ms.' The status bar at the bottom right indicates a memory usage of '173 MB'.

Database Properties

Resources Names Paths Indexes Full-Text Options Information

Add Resources

General Parsing

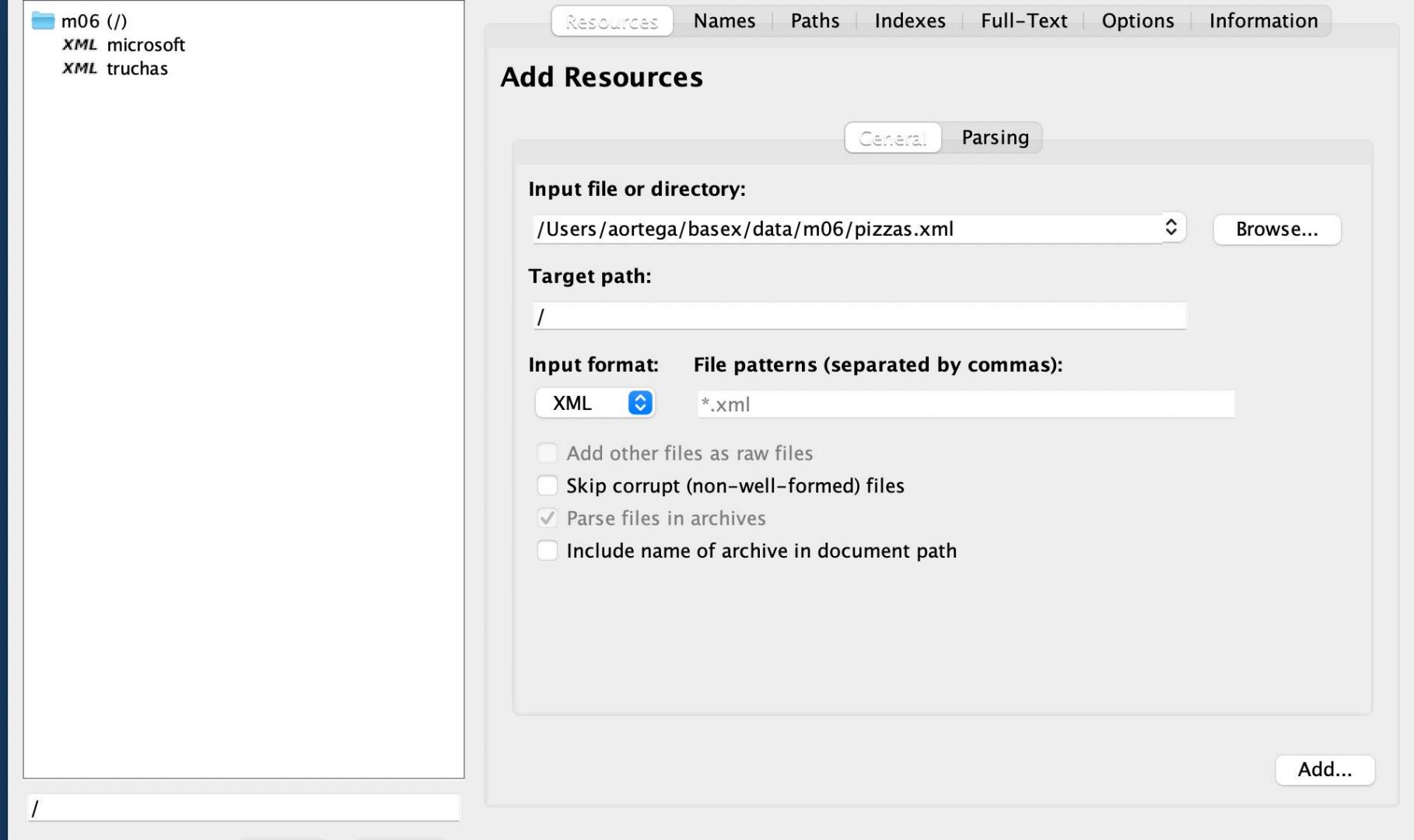
Input file or directory:
/Users/aortega/basex/data/m06/pizzas.xml

Target path:
/

Input format: File patterns (separated by commas):
XML

Add other files as raw files
 Skip corrupt (non-well-formed) files
 Parse files in archives
 Include name of archive in document path

/



XPATH

- Xpath es el lenguaje de rutas de XML, es utilizado para moverse y recuperar datos dentro de la estructura arbórea de un documento XML (Recordemos que una estructura arbórea es aquella en la que un nodo solo tiene 1 padre)
- Al trabajar con XPath deberemos recordar los diferentes elementos de un documento
 - Ráiz → primer elemento
 - Nodo → Cualquier elemento de la estructura dentro de los caracteres <>
 - Atributo → Acompañan al elemento dentro de los caracteres <> tienen un nombre un valor
 - Texto → Contenido de un elemento final (un elemento puede contener otro elemento)
 - Comentarios/Instrucciones de proceso



NODO

- En Xpath tenemos 6 tipos de nodos diferentes:
 - Elemento
 - Atributo
 - Texto
 - Namespace
 - Processing-instruction
 - Comentario

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:openDrawer="start">

    <include
        layout="@layout/app_bar_main"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!--LO NECESITAMOS APRA EDITAR LOS DATOS DEL USUARIO A MOSTRAR-->
    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header_main"
        app:menu="@menu/activity_main_drawer" />
</androidx.drawerlayout.widget.DrawerLayout>
```

RELACIONES ENTRE NODOS

- Parent → Todo nodo (excepto la raíz tiene un parent)
- Children → Un nodo puede tener o no tener hijos (0/N)
- Siblings → Nodos con un mismo parent
- Ancestors → Padres superiores de un nodo (parent, abuelo, bisabuelo)
- Descendants → Hijos inferiores de un nodo (hijo, nieto, bisnieto)

COMANDOS BÁSICOS DE XPATH

- / → Raíz
- * → Cualquier elemento
- /nodos/nodo → navegación
- //nodo → navegación express
- //nodo/node() → Recupera el elemento al completo
- //nodo/text() → Recupera el texto del nodo
- //nodo/comment() → Recupera el comentario
- . → Selecciona el nodo actual
- .. → Selecciona el nodo padre del nodo actual

- @→ Para acceder a un atributo
- @* → Para acceder a cualquier atributo

BUSCAR ELEMENTOS EN XPATH

- Se pueden utilizar comparadores para obtener consultas más precisas, por ejemplo recuperar alguna información en concreto.
- Los comparadores son:

<	<=	>	>=	=	!=	or	and	not
---	----	---	----	---	----	----	-----	-----
- Para utilizarlos pondremos la condición dentro de []

<https://docs.microsoft.com/es-es/visualstudio/xml-tools/how-to-create-an-xml-document-based-on-an-xsd-schema?view=vs-2022>

EJERCICIOS

1. Devuelve todos los PurchaseOrder
2. Devuelve el PurhcaseOrder con OrderDate="1999-10-22"
3. Devuelve el PurchaseOrder el ProductName Computer Keyboard
4. Devuelve todos los Items del purchase Order del ejercicio 2
5. Devuelve todos los PurchaseOrder donde contengan un Item con una cantidad superior a 1
6. Devuelve todos los PurchaseOrder donde tengan un Item con un USPrice inferior a 30
7. Muestra el nombre de todas las direcciones que sean del tipo Billing
8. Muestra los valores de todos los PurchaseOrderNumber

FUNCIONES XPATH

- last() → selecciona el último elemento
- position() → Selecciona el elemento en la posición especificada, puede ser substituido por las llaves de array [] pero empiezan en 1.
- count() → Devuelve el número de elementos
- sum() → Devuelve la suma de los elementos
- max() /min () / avg()
- start-width(propiedad,valor)
- ends-width(propiedad,valor)
- contains(propiedad,valor)

EJERCICIOS

1. Devuelve el número de Item dentro de cada Item
2. Devuelve el Item más caro de la colección
3. Devuelve el Item más caro de cada PurchaseOrder
4. Devuelve el Item más barato de la colección
5. Devuelve el precio medio de todos los Item
6. Devuelve el precio medio de cada PurchaseOrder
7. Devuelve todos los Item que acaben con la letra "r"
8. Devuelve todos los Items que contengan una "w"
9. Recupera el nombre de cada producto, su cantidad y precio y muéstralos por pantalla. El resultado debe ser similar a:
 - Articulo: Lawnmower - 1 - 148.95\$
 - Articulo: Baby Monitor - 2 - 39.98\$
 - Articulo: Power Supply - 1 - 45.99\$
 - Articulo: Computer Keyboard - 1 - 29.99\$
 - Articulo: Wireless Mouse - 1 - 14.99\$
10. Muestra de cada PurchaseOrder el número de pedido, el nombre de la persona a la cual se le envía la factura (@Type=Billing) y la cantidad que debe pagar
 - 99503 -> Tai Yee 188.93\$
 - 99505 -> Cristian Osorio 45.99\$
 - 99504 -> Jessica Arnold 44.98\$

EJERCICIOS – CUSTOMERS.XML

- Utilizando el fichero customers.xml realiza los siguientes ejercicios:
 1. Muestra todos los order del CustomerID “GREAL”
 2. Muestra todos los order del EmployeeID 6
 3. Muetra

XQUERY

- Xquery es la evolución de Xpath, nos permite de un XML devolver otro XML modificado con los datos que deseamos.
- Es el lenguaje que nos permite realizar consultas complejas en el XML
- Xquery nos permite:
 1. Buscar la información en los documentos
 2. Seleccionar la información deseada
 3. Manipular los datos a nuestro antojo (a nivel de nodos, texto u operaciones matemáticas)
 4. Reestructurar el documento de salida



XQUERY - CARACTERÍSTICAS

- Es case sensitive, distingue mayúsculas y minúsculas
- Los String pueden ir en ' o "
- Las variables se definen mediante el símbolo \$ seguido de un nombre
- Para mostrar una variable se podrán entre llaves { }
- Los comentarios están delimitados por los símbolos (: y :)

XQUERY – IF / SWITCH

- En Xquery se pueden utilizar los condiciones IF y Switch para mostrar unos datos u otros, siempre dependiendo de una condición

```
if() then  
  ""  
else ""
```

```
switch()  
  case "0" return ""  
  case "1" return ""  
  case "2" return ""  
  default return ""
```

FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR

Xquery	SQL
For	SELECT
Let	
Where	WHERE
Order By	Order By
Return	

FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR

Xquery	SQL
For	SELECT
Let	
Where	
Order By	
Return	

Consulta Xpath con los resultados a recuperar,
estos se guardarán en una variable

FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR

Xquery	SQL
For	SELECT
Let	Definimos las variables deseadas, resultado de la anterior ejecución
Where	
Order By	
Return	

FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR

Xquery	SQL
For	
Let	
Where	<code>SELECT</code> Filtramos resultados no deseados
Order By	
Return	

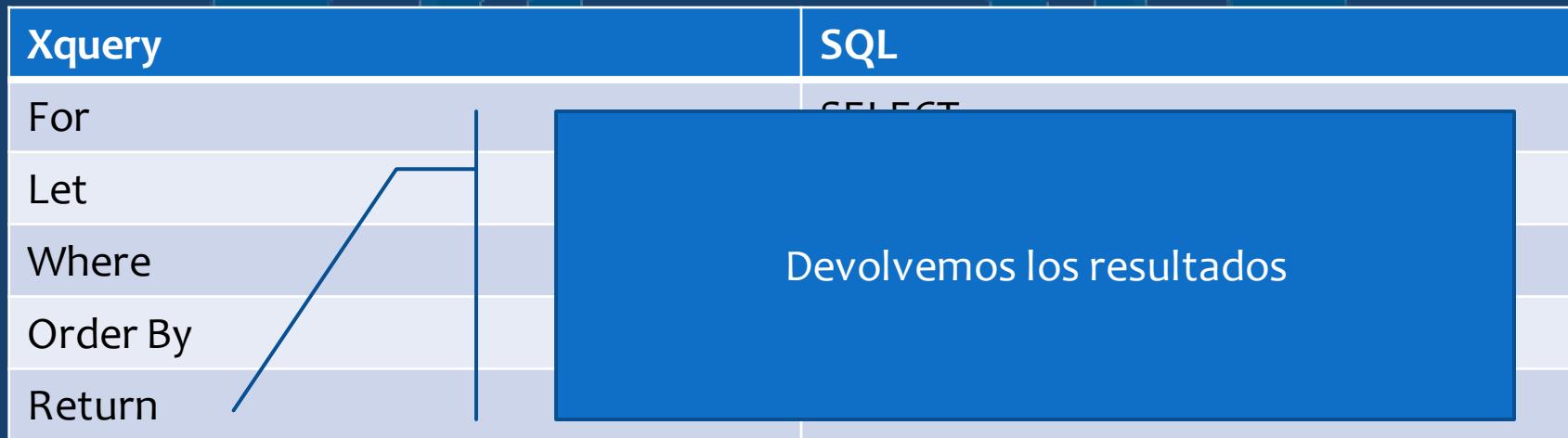
FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR

Xquery	SQL
For	
Let	
Where	
Order By	SELECT
Return	Ordenamos los resultados según nuestras necesidades

FLWOR

- Las consultas utilizan un patrón muy similar a SQL siguen la estructura FLWOR



CONSIDERACIONES

- Podemos tener tantos let como hagan falta, estos a su vez pueden ser una subconsulta Xpath.
- Let / Where / Order By no son obligatorios
- En el order by podemos poner ascending (predeterminado) o descending
- Para obtener datos pre calculados utilizaremos los let

EJERCICIO

- Crea una consulta Xquery tal que:
 - Nos devuelva estructura de la imagen de la derecha.
 - Debemos recuperar el número de pedido
 - El nombre de la persona compradora (@Type=“Shipping”)
 - El nombre de la persona pagadora (@Type=“Billing”)
 - La suma de todos los precios redondeado a dos cifras
 - La cantidad de artículos
 - Mostraremos solo aquellos que tengan una suma total superior a 45
 - Ordenaremos por fecha de forma descendente (mas nuevos al principio)

```
<Pedido num="">
  <Comprador></Comprador>
  <Pagador></Pagador>
  <Precio></Precio>
  <Cantidad></Cantidad>
</Pedido>
```

EJERCICIO

- Utilizando el fichero de truchas.xml
- 1. Mostrar todas las truchas modificando la etiqueta row por etiqueta trucha
- 2. Mostrar solo las truchas que pesen más de 3000
- 3. Mostrar solo las truchas que pesen menos de 3000
- 4. Mostrar solo las truchas que pesen 3000
- 5. Mostrar las truchas que midan más de 600, únicamente deseamos los datos id, tanque, qr y mida
- 6. Mostrar el estado reproductivo de las truchas, utilizar la instrucción switch para:
 1. Estado reproductivo 0 → NO ESTABLECIDO
 2. Estado reproductivo 1 → VERDE
 3. Estado reproductivo 2 → MADURA
 4. Estado reproductivo 3 → BLANCA
- 7. Mostrar si la trucha está genotipada o no, utilizando la instrucción if
 1. Genotipada 0 -> NO
 2. Genotipada 1 -> SI
- 8. Muestra solo las truchas vivas
- 9. Muestra las truchas muertas

EJERCICIOS HTML - OPCIONALES

- Deberemos de crear un documento html para mostrar la información correspondiente.
1. Genera un documento HTML que nos muestre el listado de todas las truchas ordenado por id simplemente utiliza la estructura `id`
 2. Genera un documento HTML que incorpore el toolkit Bootstrap. Genera un documento donde la estructura sea como la imagen siguiente
 1. Solo se mostrarán las vivas
 2. Solo se mostrarán las que tengan un estado_reproductivo != o
 1. Los estados pueden ser:
 1. BLANCA
 2. VERDE
 3. MADURA

ID:925	TANQUE: R8.2	QR: 2241
	MADURA	
PESO: 2200		TAMAÑO: 538

XSLT

- XSLT es una tecnología que nos permite transformar documentos XML en otros documentos, Extensible Stylesheet Language Transformations.
- Para ello hay que definir cual es la transformación que queremos realizar
 - Necesitaremos dos cabeceras
 - `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">` → Para definir cual será el estilo
 - `<xsl:template match="/">` → Para definir donde realizaremos la transformación
 - XSLT tiene muchas funciones pero vamos a explorar alguna de ellas:

XSLT - FUNCIONES

- `<xsl:value-of select="" />`
- `<xsl:variable name="" select ="" />`
- `<xsl:sort select="" />`
- `<xsl:if test=""></xsl:if>`
- `<xsl:choose><xsl:when test=""></xsl:when><xsl:when test=""></xsl:when><xsl:otherwise></xsl:otherwise></xsl:choose>`
- `<xsl:for-each select="" /></xsl:for-each>`

EJECUCIÓN DE XSLT

```
let $origin := /Root  
let $xslt := fn:doc('/Users/aortega/basex/data/mo6/queries/customers.xslt')  
return xslt:transform($origin,$xslt)
```

EJERCICIOS

- Utilizando el fichero de customers.xml
 - En la siguiente diapositiva tienes un ejemplo del resultado final
1. Traduce todo el documento
 2. Dentro de cada customer añade sus orders de forma que quede todo 100% ordenado
 3. Elimina la etiqueta ContactTitle
 4. Ordena los pedidos por el ShippedDate
 5. Elimina todos los pedidos que no tengan ShippedDate

```
<Cliente>
  <Compañia>Great Lakes Food Market</Compañia>
  <Contacto>Howard Snyder</Contacto>
  <Telefono>(503) 555-7555</Telefono>
  <DireccionCompleta>
    <Direccion>2732 Baker Blvd.</Direccion>
    <Ciudad>Eugene</Ciudad>
    <Region>OR</Region>
    <CP>97403</CP>
    <Pais>USA</Pais>
  </DireccionCompleta>
  <Pedidos>
    <Pedido>
      <Empleado>6</Empleado>
      <FechaPedido>1997-05-06T00:00:00</FechaPedido>
      <FechaNecesitado>1997-05-20T00:00:00</FechaNecesitado>
      <Envio fecha="1997-05-09T00:00:00">
        <Via>2</Via>
        <Transporte>3.35</Transporte>
        <Nombre>Great Lakes Food Market</Nombre>
        <Direccion>2732 Baker Blvd.</Direccion>
        <Ciudad>Eugene</Ciudad>
        <Region>OR</Region>
        <CP>97403</CP>
        <Pais>USA</Pais>
      </Envio>
    </Pedido>
  </Pedidos>

```

PARAMETRIZAR CONSULTA

```
declare variable $datos as xs:string external;
declare variable $transform as xs:string external;
let $origin := doc($datos)
let $xslt := doc($transform)
return xslt:transform($origin,$xslt)
```


TEMA 6: PROGRAMACIÓN DE COMPONENTES.

- Concepto de componente y características
- Propiedades y atributos
- Eventos; asociación de acciones a eventos
- Persistencia del componente.
- Herramientas para desarrollo de componentes no visuales.
- Empaquetado de componentes.

¿QUÉ ES UN COMPONENTE?

- Un componente más conocido como (en inglés) bean, consiste en una clase con dos finalidades muy clara.
 - Almacenar datos de en nuestras aplicaciones
 - Crear un sistema de reutilización de código en nuestras aplicaciones de forma mucho más extensa.
- Sin saberlo, prácticamente hemos estado trabajando con representaciones de componentes, como por ejemplo en el tema 3-Hibernate.

ESTRUCTURA DE UN COMPONENTE

- Un componente debe cumplir con una estructura muy clara para que se le pueda identificar como tal.
 - Debe ser una clase Serializable (implementando la interfaz) o extendiendo de otra clase que la implemente.
 - Todas sus variables deben de ser privadas.
 - Deben contar con todos sus atributos getter y setter.
 - Debe contar con un constructor por defecto (sin parámetros).

REQUISITOS EN LOS GETTER Y SETTER

Getter

Deben empezar por get seguido del nombre de la variable, a excepción de los booleanos que pueden empezar por is

Deben devolver el tipo de la variable

NO deben recibir ningún parámetro

Deben ser públicos

```
private String nombre;
```

```
/** CORRECTO */
public String getNombre() {
    return nombre;
}
```

```
/** INCORRECTO */
public String obtenerNombre(){
    return nombre;
}
```

```
public String getName(){
    return nombre;
}
```

```
public String getnombre(){
    return nombre;
}
```

```
public String dameNombre(){
    return nombre;
}
```

REQUISITOS EN LOS GETTER Y SETTER

Setter

Deben empezar por set seguido del nombre de la variable

Deben devolver void

Deben recibir el mismo tipo de variable a la que se desea guardar

Deben ser públicos

```
private String nombre;
```

/** CORRECTO */

```
public void setNombre(String nombre){  
    this.nombre = nombre;  
}
```

/** INCORRECTO */

```
public String setNombre(String nombre){  
    this.nombre = nombre;  
    return nombre;  
}
```

```
public void ponerNombre(String nombre){  
    this.nombre=nombre;  
}
```

```
public void setNombre(char[] nombre){  
    this.nombre = new String(nombre);  
}
```

CARACTERÍSTICAS DE UN BEAN

- Introspección: Proceso mediante el cual los propios bean expresan su comportamiento.
 - Esto se realiza gracias a las pautas de diseño anteriormente comentadas.
- Propiedades: Permite cambiar el valor de sus atributos (mediante setter) para personalizarlo.
- Personalización: Se pueden alterar la apariencia y conducta del bean en un momento concreto.
 - Este apartado afecta especialmente a la construcción de UI, donde por ejemplo un botón puede ser muy diferente a otro botón.
- Eventos: Tienen la capacidad de informar de los eventos sobre los que puede estar atento.
- Persistencia: Un bean debe permitir guardar su estado y recuperarlo posteriormente (por ello debe implementar la interfaz Serializable).

PROPIEADES DE LOS BEAN

- Los bean pueden tener 4 tipo diferente de propiedades
 - Simples
 - Indexadas
 - Ligadas (bound)
 - Restringidas (constrained)

PROPIEDADES SIMPLES

- Son las más fáciles de todas, ya que son aquellas que únicamente representan un único valor en un momento concreto.
- Están representadas por un tipo y un nombre.

```
private String nombre;  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getNombre() {  
    return nombre;  
}
```

```
private Persona persona;  
  
public void setPersona(Persona persona) {  
    this.persona = persona;  
}  
  
public Persona getPersona() {  
    return persona;  
}
```

PROPIEDADES INDEXADAS

- Son iguales que las simples, pero en lugar de contener un único valor contienen un conjunto de valores a los que se puede acceder mediante un índice.
 - Al tener un conjunto de valores debe tener 2 getter y 2 setter uno para todo el conjunto de datos y otro para datos individuales

```
private Rueda[] ruedas;

public Rueda[] getRuedas() {
    return ruedas;
}

public void setRuedas(Rueda[] ruedas) {
    this.ruedas = ruedas;
}

public Rueda getRueda(int idx) {
    return ruedas[idx];
}

public void setRueda(Rueda rueda, int idx) {
    this.ruedas[idx] = rueda;
}
```

PROPIEDADES LIGADAS

- Cuando una propiedad es ligada quiere decir que tiene la capacidad de notificar a otros objetos (*listeners*) que se ha producido un cambio en su valor. Esto permite al objeto actuar en consecuencia, por lo que podemos observar que un objeto está ligado a una propiedad.
- Para poder hacer esta unión podemos utilizar la clase **PropertyChangeSupport**, especialmente creada para dar soporte a las propiedades ligadas y poder realizar la vinculación con los *listeners* de forma correcta

EJEMPLO PANTALLA_TURNO-PERSONA

- Imaginemos el caso donde disponemos de una pantalla que va mostrando turnos y una lista de personas, cada persona tiene asignado un turno y deberá comprobar cada vez que en la pantalla se cambia el turno si le toca o debe seguir esperando.
 - Cuando se crea la persona debe ligarse con la pantalla, cuando sea su turno debe desligarse
- Para todo esto utilizaremos la clase `PropertyChangeSupport`, `PropertyChangeListener` y `PropertyChangeEvent`
 - `PropertyChangeSupport` → Definiremos el controlador sobre la lista de objetos ligados
 - `PropertyChangeListener` → Es el objeto que se liga a la propiedad
 - `PropertyChangeEvent` → Es el evento emitido por la clase escuchada

EJEMPLO PANTALLA TURNO-PERSONA

```
public class Pantalla {  
  
    private int turno;  
    private PropertyChangeSupport controladora;  
  
    public Pantalla() {  
        this.controladora = new PropertyChangeSupport(this);  
    }  
  
    public void nuevoObservador(PropertyChangeListener listener) {  
        this.controladora.addPropertyChangeListener(listener);  
    }  
    public void eliminarObservador(PropertyChangeListener listener) {  
        this.controladora.removePropertyChangeListener(listener);  
    }  
    public void setTurno(int turno) {  
        controladora.firePropertyChange("turno", this.turno, turno);  
        this.turno = turno;  
    }  
  
    public int getTurno() {  
        return turno;  
    }  
}
```

EJEMPLO PANTALLA_TURNO-PERSONA

```
public class Persona implements PropertyChangeListener {  
  
    @Override  
    public void propertyChange(PropertyChangeEvent evt){  
        if ((int) evt.getNewValue() == turno){  
            System.out.println("Me toca!");  
            Pantalla pantalla = (Pantalla) evt.getSource();  
            pantalla.eliminarObservador(this);  
  
        } else{  
            System.out.println("Sigo esperando...");  
        }  
    }  
  
    private String nombre;  
    private int edad;  
    private int turno;  
    ...  
}
```

PROPIEDADES RESTRINGIDAS

- Son muy similares a las ligadas pero en este caso los observadores puede “juzgar” el valor que se ha añadido y determinar si es correcto o no es correcto.



EJEMPLO PANTALLA_TURNO-PERSONA

```
public class Pantalla {  
  
    private int turno;  
    private VetoableChangeSupport controladora;  
  
    public Pantalla(){  
        this.controladora = new VetoableChangeSupport(this);  
    }  
  
    public void nuevoObservador(VetoableChangeListener listener){  
        this.controladora.addVetoableChangeListener(listener);  
    }  
    public void eliminarObservador(VetoableChangeListener listener){  
        this.controladora.removeVetoableChangeListener(listener);  
    }  
    public void setTurno(int turno) throws PropertyVetoException {  
        controladora.fireVetoableChange("turno", this.turno, turno);  
        this.turno = turno;  
    }  
  
    public int getTurno() {  
        return turno;  
    }  
}
```

EJEMPLO PANTALLA_TURNO-PERSONA

```
public class Persona implements VetoableChangeListener {  
  
    @Override  
    public void vetoableChange(PropertyChangeEvent evt) throws  
PropertyVetoException {  
    int newTurno = (int) evt.getNewValue();  
    System.out.println(newTurno);  
    if (newTurno < 1) {  
        throw new PropertyVetoException("El turno no puede ser inferior a 0", evt);  
    }  
    if ((int) evt.getNewValue() == turno) {  
        System.out.println("Me toca!");  
        Pantalla pantalla = (Pantalla) evt.getSource();  
        pantalla.eliminarObservador(this);  
    } else {  
        System.out.println("Sigo esperando...");  
    }  
}
```

ARQUITECTURA DE COMPONENTES

- Hasta el momento hemos hablado únicamente de Bean o JavaBean y que como hemos visto no son nada más que unas directrices para crear clases de una forma concreta.
 - Pero no acaba aquí...
- Enterprise JavaBean y Spring
 - Ambos son contenedores de componentes, que vendría a ser como un programa superior donde residen los componentes, este es el encargado de administrarlos.