salesianos
DOMINGO SAVIO

Tema 1. MANEJO DE FICHEROS

M07-Acceso a datos

REPASO DE TÉRMINOS

- **Registro:** Estructura para representar información. Consta de una serie de campos, en cada uno de los cuales se puede almacenar un dato particular.
- **Fichero.** Unidad fundamental de almacenamiento. Consiste en una secuencia de bytes. Conjunto de bits almacenados en un dispositivo.
- **Acceso aleatorio**: tipo de acceso al fichero que permite acceder directamente a los datos situados en cualquier posición del fichero.
- **Acceso secuencial**: tipo de acceso al fichero que única forma de acceder a una posición determinada es leer el contenido anterior desde el principio.

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 (hay ficheros almacenados en memoria RAM, temporales, existen mientras la RAM tiene electricidad, son los causantes de los problemas que se solucionan reiniciando)

Los ficheros se podrían definir por tres secciones:

- ruta: lugar dónde se encuentra ubicado
- 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 dos rutas distintas. Es decir, el conjunto de elementos tiene que ser único.

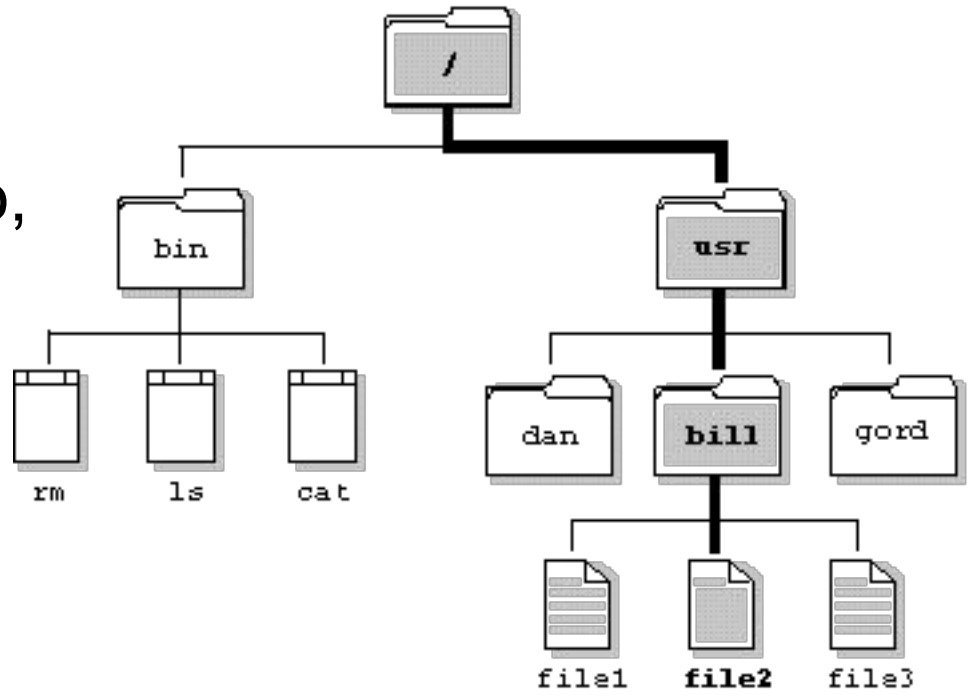
Las extensiones determinan qué tipo de fichero es y por tanto, cómo debe tratarlo el sistema operativo.

¿A quién corresponden estas extensiones?

- La mayoría de las extensiones suelen tener 3 letras, pero pueden ir de 2 a 4 caracteres. La extensión determina que tipo de fichero es y cómo se deben leer (la cantidad de bits que se leen)
  - .sh
  - .html
  - .pptx
  - .exe
  - .dmg

Fichero

- Se almacenan dentro del sistema de forma jerárquica.
- No existe una estructura determinada, la determina cada programador, por lo tanto, él determina cómo se debe de leer.
- Se organiza de registros de distintos tamaños 16, 32…
- Si abrimos un fichero con otro tipo de programa que no es el correspondiente veremos información rara.



Preguntas

- ¿Es necesaria la extensión en todos los sistemas Operativos?
- ¿Qué es una extensión MIME?

ORGANIZACIÓN INTERNA DE LOS FICHEROS

- No existe una forma predeterminada de cómo 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 ficheros y mostrarlo de forma correcta.

TIPOS DE FICHEROS DENTRO DE UN SO

- **Fichero estándar:** Fichero que contiene cualquier tipo de datos. Documentos, imágenes, audio, vídeo, etc.

- **Directorio o carpeta:** Fichero que contiene otros ficheros. Sirve para organizar de forma jerárquica los diferentes ficheros

- **Ficheros especiales:** Ficheros que sirven para controlar los diferentes periféricos conectados al ordenador

TIPOS DE FICHEROS

- Desde el punto de vista del programador distinguiremos esencialmente dos ficheros con los que trabajaremos.
  - **Fichero de texto**: Se considera un fichero de texto si su único contenido son caracteres de texto y también espacios y separadores tales como tabuladores y retornos de carro. Este tipo de ficheros se puede leer con un simple editor de texto.
    - Extensiones como .txt (Fichero texto plano), .xml (Fichero XML), .json (Fichero de intercambio de información), .sql (Script SQL), .conf (Fichero de configuración)
  - **Fichero binario**: Son ficheros que no están compuestos exclusivamente de texto. Los programas almacenan su información en ficheros binarios

CODIFICACIONES PARA TEXTO

- Un texto es una secuencia de caracteres. Un texto se almacena como una secuencia de bytes. Una codificación es un método para representar cualquier texto como una secuencia de bytes. Dos textos iguales con distinta codificación se representa como una cadena de bytes distinta.
  - UTF-8 (Compatible con el código ASCII)

RUTA DE UN FICHERO

- Para acceder a un determinado fichero, se utiliza la ruta (path)
  - Una ruta indica la dirección del fichero en el sistema de archivos
    - En una ruta, cada nivel de la jerarquía se representa delimitado por el símbolo /. En Windows, el símbolo separador es \ . Además de /, existen 2 elementos especiales en la ruta
    - . Representa al directorio actual
    - .. Representa al directorio padre en la jerarquía
- Existen 2 tipos de rutas:
  - Absoluta: Ruta al fichero desde el directorio principal (root). Ej: /home/Documentos/ejemplo.txt
  - Relativa: Ruta al fichero desde el directorio actual. Ej: Estando en home, ./Documentos/ejemplo.txt

OPERACIONES BÁSICAS CON FICHEROS

Las operaciones básicas que admite un fichero son:

- **Creación de un fichero**
- **Apertura de un fichero:** no se puede utilizar un fichero que no está abierto en nuestro sistema
  - Lectura → se debe tener permisos de lectura sobre el fichero
  - Escritura → se debe tener permiso de escritura
- **Cierre de un fichero:** se debe cerrar un fichero para que lo puedan manejar otros programas. Si se queda abierto de manera indefinida puede haber situaciones de bloqueo.

PERMISOS SOBRE FICHEROS

- El usuario que crea el fichero tiene derecho a decidir quien y cómo accede a su fichero
- Existen 3 grupos para los que se les puede definir permisos
  - u. Propietario
  - g. Grupo
  - o. Resto de usuarios
- Los permisos que se pueden dar son los siguientes
  - r. Lectura
  - w. Escritura
  - x. Ejecución

FORMA 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.
  - 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.

FICHEROS EN JAVA

- Java sigue el concepto de fichero como un conjunto de bytes.
  - El fichero se “reserva” para poder operar con él.
  - Se establece un flujo de datos desde el fichero a una variable en Java, que representa el fichero.
  - Para cerrar el fichero hay que liberar la variable
- Para gestionar todas las operaciones con los ficheros, se utilizan dos librerías incluidas en el **jdk** de Java (no hay que instalar nada)
  - Java.io → Java Input Output
  - Java.nio → Java Non-blocking Input Output

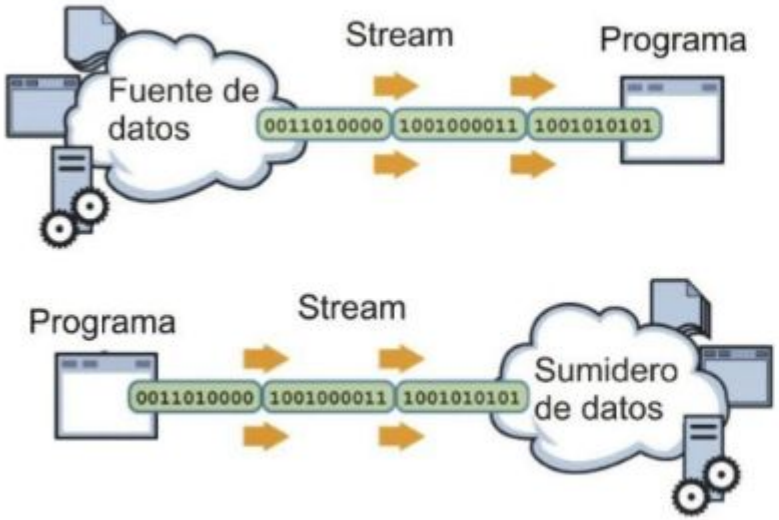
    Esta segunda librería supone una mejora en la forma en la que se realizan las operaciones. Ambas pueden ser utilizadas, pero java.nio corrigió muchas de las deficiencias de java.io

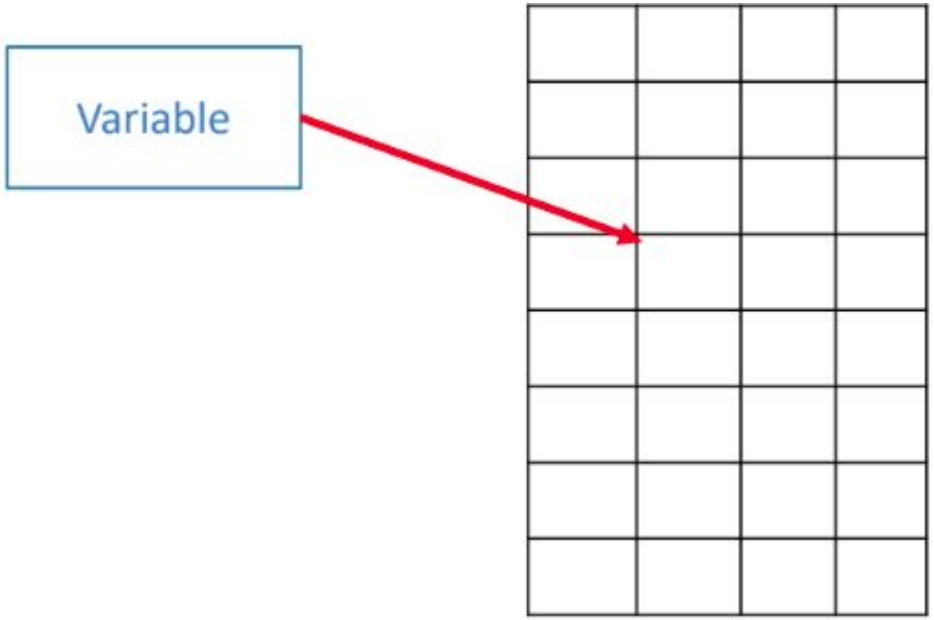
Tema 1.1 JAVA.IO

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 (otra clase)

STREAM DE DATOS





CLASE FILE EN JAVA

- La clase file permite obtener información relativa a directorios y ficheros dentro de un sistema de ficheros y realizar diversas operaciones con ellos como leer, escribir, borrar… Pero directamente con esta clase no se puede leer ni escribir.
  - Importar la clase file → import java.io.File;
  - Esta clase tiene tres constructores diferentes dependiendo de cómo le pasemos la información.
    - 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);
  - Crea ficheros en la RAM
  - Crear fichero (Ejemplo 1 y 2 constructor)
    - File nombreFile = new File("/carpeta/fichero.txt"); →La ruta puede ser relativa o absoluta
    - También se puede indicar la ruta y el nombre cómo dos parámetros separados → File fichero = new File ("/carpeta/ ","fichero.txt");
      - La variable nombreFile es un objeto con todos los datos de fichero que se encuentra en la ruta pasada por el parámetro
      - Después de crear el fichero, se invocará al método createNewFile nombreFile.createNewFile()
    - Se puede crear tanto ficheros como directorios, pero no se pueden crear al mismo tiempo. Se debe crear uno y después otro.
      - nombreFile.createNewFile
  - Eliminar fichero
    - nombreFile.delete();
  - 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
  - Todos los métodos de la clase File
    - https://docs.oracle.com/javase/7/docs/api/java/io/File.html

USO CONSTRUCTORES

```
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");
```

CAPTURAR LA EXCEPCIÓN (IOException)

- Cuando se produce una excepción **IOException** significa que se ha producido un error en la entrada/salida.
  - Casi todas las operaciones de entrada y salida pasan por IOException
  - Son excepciones controladas, es el programador quien controla el fragmento de código (En Java existe las controladas, Exception y las no controladas, RuntimeException)
- Importar la clase → import java.io.IOException;
- try { Aquí escribimos la operación de apertura del fichero }
- catch (IOException ioe) { ioe.printStackTrace(); }
- Podemos capturar el mensaje con ioe.getMessage()

CREAR FICHERO

```
try {
        //Path to create a File
        String path = "/Users/Beatriz/Desktop/fichero.txt";
        String content = "Contenido de ejemplo";
        //Create a file
        File file = new File(path);
        // Si el archivo no existe es creado
        //If this file doesn´t exist, the method createNewFile creates it
        if (!file.exists()) {
        file.createNewFile();
}
}catch(IOException ioe) {
     ioe.printStackTrace();
}
```

RESUMEN MÉTODOS CLASE FILE

Función	Explicación	Ejemplo
list()	Devuelve un listado con todos los ficheros y directorios del directorio utilizado → Devuelve un String	File ficheros = new File("ficheros"); ficheros.list();
listFiles()	Devuelve un listado con los ficheros del directorio → Devuelve un String	File ficheros = new File("ficheros"); ficheros.listFiles();
getPath()	Obtiene la ruta del fichero, el inicio es el punto de ejecución del programa. Es decir, devuelve la ruta desde la cual nos encontramos → Devuelve booleano	File ficheros = new File("ficheros"); System.out.println(ficheros.getPath());
getAbsolutePath()	Obtiene la ruta completa desde la raíz del sistema al fichero → Devuelve booleano	File ficheros = new File("ficheros"); System.out.println(ficheros. getAbsolutePath());
getParent()	Obtiene el nombre del directorio padre → Devuelve booleano	File ficheros = new File("ficheros"); System.out.println(ficheros. getParent());
canRead()	Devuelve true o false, dependiendo si se puede o no leer (Permisos) → Devuelve booleano Devolverá falso hasta que esté creado	File ficheros = new File("ficheros"); System.out.println(ficheros. canRead());
canWrite()	Devuelve true o false, dependiendo si se puede o no escribir (Permisos) → Devuelve booleano Devolverá falso hasta que esté creado	File ficheros = new File("ficheros"); System.out.println(ficheros. canWrite());
mkdir()	Crea un nuevo directorio → Devuelve booleano Se crea de forma física en el disco	File ficheros = new File("ficheros"); System.out.println(ficheros. mkdir());
createNewFile()	Crea un nuevo fichero → Devuelve booleano Se crea de forma física en el disco	File ficheros = new File("ficheros"); System.out.println(ficheros. createNewFile());
delete()	Elimina un fichero o directorio, si es un directorio debe estar vacío. → Devuelve booleano	File ficheros = new File("ficheros"); System.out.println(ficheros. delete());
renameTo()	Renombra un fichero o directorio → Devuelve booleano	File ficheros = new File("ficheros"); System.out.println(ficheros.renameTo(new File("eliminados")));

RESUMEN MÉTODOS CLASE FILE

Método	Descripción
getName()	Devuelve un String con el nombre del fichero
exists()	Boolean que nos indicará si el fichero existe
isDirectory()	Boolean que indica si el fichero es un directorio
lastModified()	Devuelve la última hora de modificación del archivo
length()	Devuelve la longitud del archivo

OPERACIONES BÁSICAS DE FICHEROS EN JAVA

- Existen cuatro operaciones sobre ficheros que son esenciales para la gestión de los mismos:
  - Creación
  - Eliminación
  - Copia
  - Movimiento
- No todas ellas se pueden realizar con el API básico de Java.io. En esa API están disponibles las operaciones de
  - Creación
    - File fichero = new File(ruta);
    - fichero.createNewFile();
  - Eliminación (No lanza excepciones, por lo que, como mkdir podemos controlarla con un condicional)
    - File fichero = new File(ruta);
    - fichero.delete;
  - Hay una operación relacionada con las anteriores que es rename
    - File file = new File("teoria/operaciones_basicas");
      File rename = new File("teoria/operaciones_basicas1");
      file.renameTo(rename);

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?

EJERCICIOS

1. Crea un fichero que se llame fichero1.txt, verifica que existe.

2. Una vez que hayas verificado que existe, renombrarlo a ficheroCopia con la operación rename.

FILEUTILS

- Java.io no permite las operaciones de copia y movimiento de ficheros, por lo que, vamos a utilizar la librería de FileUtils.
- Para ello nos debemos descargar de Apache Commons el jar para incluirlo en eclipse.
- Esta librería nos ofrece una serie de mecanismos para controlar la copia y el movimiento de ficheros de manera sencilla.
- Nos proporciona funciones estáticas (no hay que crear ningún objeto), pero si hay que controlar las excepciones.
- Esta librería contiene muchas operaciones, entre ellas nos permitiría escribir.
- Las operaciones de copia y movimiento arrojan errores hay que utilizar try y catch.
- https://commons.apache.org/proper/commons-io/apidocs/org/apache/commons/io/FileUtils.html
- Importar librería en eclipse
  - https://www.youtube.com/watch?v=z5Dq8-L-23g

EJEMPLO DE COPIA DE UN FICHERO

```
try {
        // Dentro de fichero1 metemos ficheroCopia
        FileUtils.copyFile(ficheroCopia,fichero1);
    }
    catch (IOException e)
    {
        System.err.println("Error a copiar un fichero dentro de otro");
        e.getStackTrace();
    }
```

EJEMPLO DE MOVIMIENTO FICHERO

```
try {
        FileUtils.moveFile(fichero1,new File (ruta,"fichero3.txt"));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

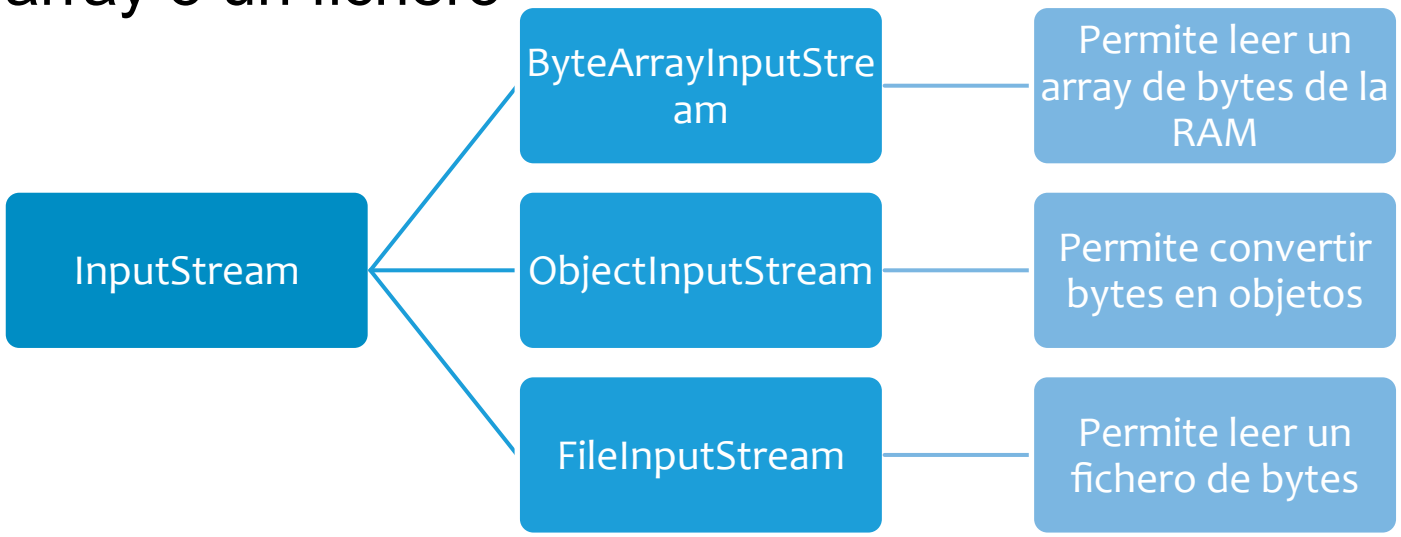
FICHEROS EN JAVA

En Java podemos diferenciar dos tipos de ficheros:

- **Binario (stream de 8 bits)**: se basan en enviar la información en bloques de 8 bits, 1 bytes.
  - Podemos decir que se consideran binarios todos aquellos ficheros que no son legibles por una aplicación como notepad o block de notas.
- **Caracteres (stream de 16 bits)**: Se basan en enviar la información por bloques 16 bits, 2 bytes, esto es debido a que la codificación Unicode (UTF-8) utiliza dos bytes para representar cada carácter.

FICHEROS BINARIOS

- 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

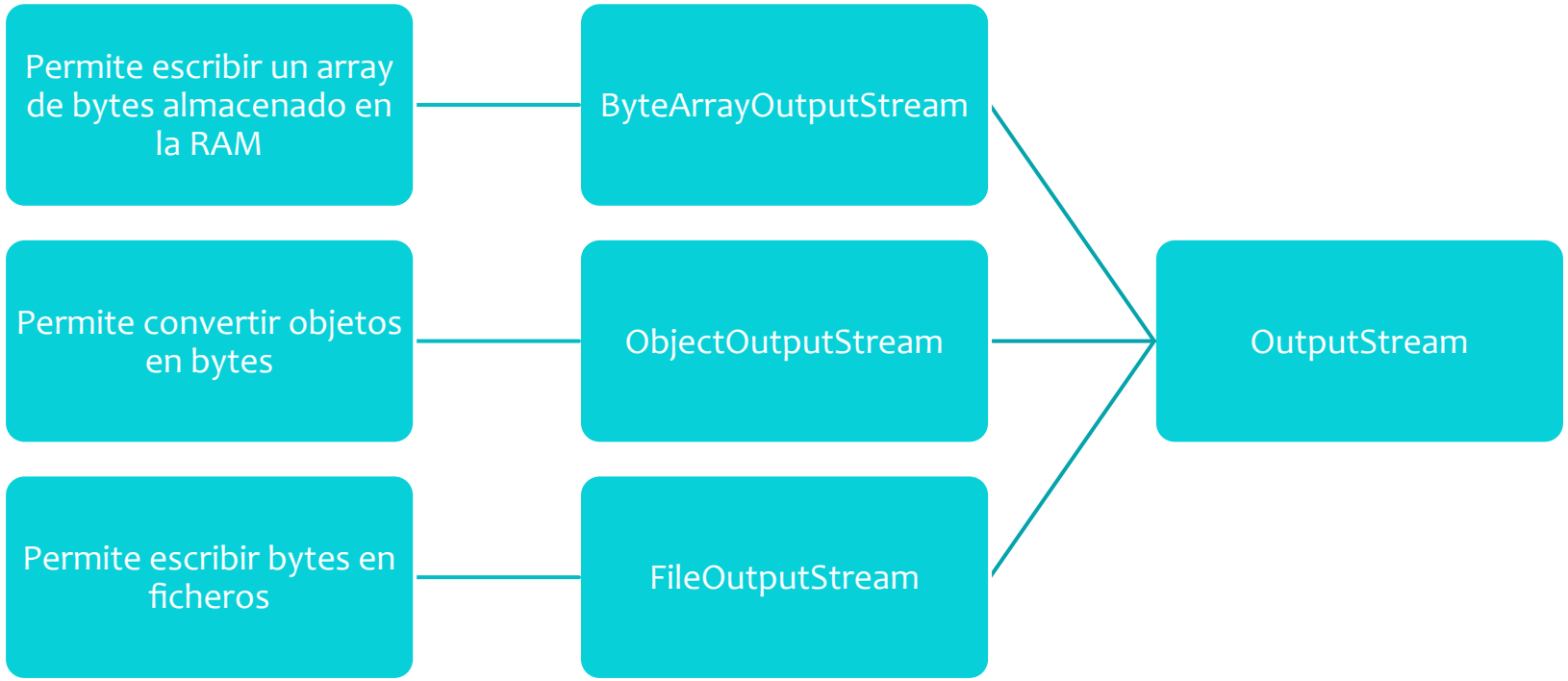


FICHEROS BINARIOS (II)

Para escribir un objeto en un fichero, éste debe implementar la interfaz serializable. Sin esta interfaz no se va a poder escribir el objeto en un fichero.

Convertir Objetos en bytes nos acercará a nuestro objetivo que sería generar una base de datos. Ya que los objetos son recuperables.

CLASE OUTPUTSTREAM



CONSTRUCTORES: *INPUTSTREAM

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

EXCEPCIONES

Hay algunas excepciones 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.

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;
}
```

FICHEROS DE CARACTERES

Los ficheros de caracteres o streams de 16 bits (2 bytes) utilizan unas clases preparadas para la lectura y escritura que son Reader y Writer.

- De estas clases utilizaremos FileReader y FileWriter.
  - 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
  - public FileWriter(File file) throws IOException
  - public FileWriter(FileDescriptor fd)
  - public FileWriter(File file, boolean append) throws IOException
  - public FileWriter(String fileName) throws IOException
  - public FileWriter(String fileName, boolean append) throws IOException

LECTURA DE UN FICHERO

- Clase File: Para representar un fichero que se quiere leer
  - File fichero = new File (“ruta fichero”);
- Clase FileReader: establece un stream de datos de lectura del fichero. Recibe un objeto FIle
  - FileReader reader = new FileReader(fichero)
- Clase BufferedReader: crea un buffer a través del FileReader, que permite leer más de un carácter. El constructor recibe el FileReader como parámetro
  - BufferedReader buffer =new BufferedReader (reader)
- La función BufferedReader puede utilizar la llamada readline() que devuelve la siguiente línea de texto si existe y si no devuelve null

CLASES BUFFEREDREADER Y BUFFEREDWRITER

Estas clases utilizan un buffer para realizar las lecturas y las escrituras de manera más eficiente, es decir, optimizan las funciones anteriormente mencionadas.

Se crean igual que **FileReader y FileWriter**, pero como parámetro insertamos un objeto **FileReader** para **BufferedReader** y un objeto **FileWriter** para **BufferedWriter**.

La principal ventaja es que se puede leer e insertar líneas completas en vez de caracter a caracter.

ESCRIBIR EN UN FICHERO

- La clase **FileWriter** permite escribir, primero me debemos utilizar la clase File.
  - Este método puede usar como parámetro un String con lo que queremos escribir o un número que corresponderá un carácter de la tabla ASCII.
  - Este método si el fichero no existe lo crea en la ruta que hayas especificado en el parámetro
  - Si el fichero tiene contenido, éste se sobrescribe.
  - Se puede activar la opción append, que escribe los datos al final del fichero, poniendo la opción a true.
    - **FileWriter(String fileName, boolean append)**:
    - FileWriter fw = new FileWriter(file,true);
  - Después del uso del fichero, es conveniente cerrar el fichero con un close().

EJEMPLO DE ESCRIBIR 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 DE ESCRITURA EN UN FICHERO

```
Try{

    FileWriter fw = new FileWriter("/Users/Beatriz/Desktop/fichero1.txt");
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write("Hello World");
    bw.close();

}catch(IOException ioe) {
    ioe.printStackTrace();
}
```

LECTURA DEL CONTENIDO DE UN FICHERO

- Para la lectura de un fichero utilizaremos FileReader, este método no tiene parámetros pero devuelve un número que se le hacemos casting a char este será legible por nosotros. Este número lo podemos mostrar o inclusive pasarlo a otro fichero.
- Cuando termina el fichero devuelve -1.

EJEMPLO DE LECTURA DE UN FICHERO

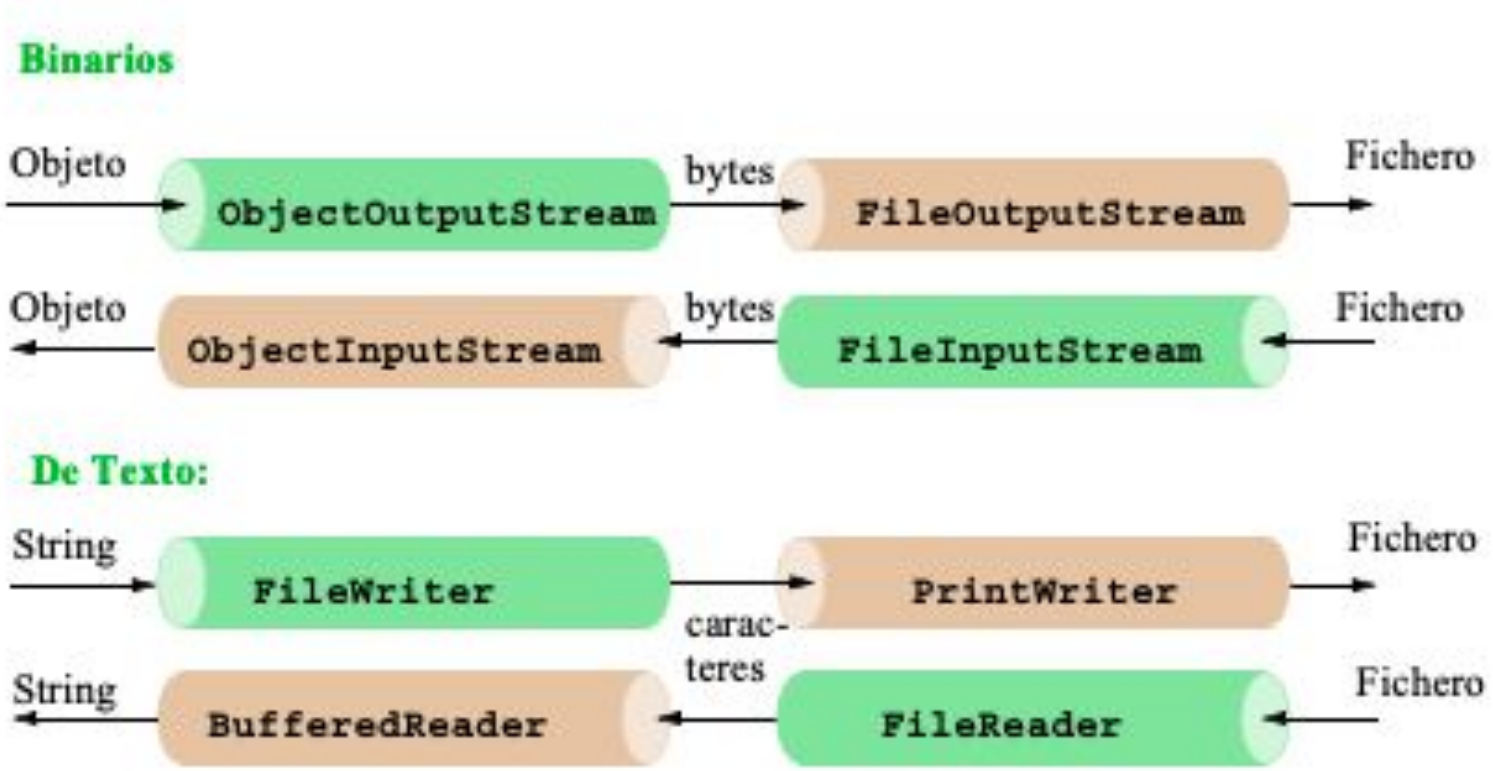
```
Try{
    FileReader fr=new FileReader("/Users/Beatriz/Desktop/fichero1.txt");
    //Leemos el fichero y lo mostramos por pantalla
    int valor=fr.read();
    while(valor!=-1){
        System.out.print((char)valor);
        valor=fr.read();
    }
    //Cerramos el fichero
    fr.close();
}catch(IOException ioe) {
ioe.printStackTrace();

}
```

EJERCICIOS

- En Java además de poder utilizar BufferedWriter, se puede utilizar PrintWriter. Investiga cuales son las diferencias entre una forma y otra.
- En los apuntes tienes un ejemplo del uso de BufferedWriter, ¿cómo sería la lectura con BufferedReader? Realiza un ejemplo.

RESUMEN FICHERO ACCESO SECUENCIAL



FICHEROS SEGÚN SU ACCESO

Los ficheros según hemos visto podemos acceder a su contenido de dos formas:

- **Acceso secuencial:** todas las funciones de lectura que hemos visto anteriormente. Por eso, recorremos el fichero desde el inicio hasta el final.
  - Fichero binario → 1 byte
  - Fichero aleatorio → 2 bytes
- **Acceso aleatorio**: para el acceso aleatorio se simplifica porque en vez de dos clases sólo tenemos que utilizar una **RandomAccessFile**, que nos proporciona los mecanismos necesarios para recorrer el fichero utilizando un apuntador.
  - Se enumeran con un índice que empieza en cero
  - Si escribimos al final del fichero el tamaño se amplia

CONSTRUCTORES RANDOMACCESSFILE

RandomAccessFile(String name, String accesMode)	• NAME: Nombre del fichero a abrir • AccesMode: Mode de acceso al fichero • r:lectura • rw: lectura y escritura
RandomAccessFile(File file, String accesMode)	• File: fichero a abrir • AccesMode: Mode de acceso al fichero

FUNCIONES RANDOMACCESSFILE

Descripción	Declaración
Intenta leer un array de bytes. Retorna el número de bytes leídos, o -1 si no quedan más	`int read(byte[] b) throws IOException`
Lee repetidamente hasta rellenar el array de bytes completo. Lanza `EOFException` si se acaba el fichero y no se ha podido leer todo	`void readFully(byte[] b) throws IOException`
Lee un double. Lanza `EOFException` si se acaba el fichero	`double readDouble() throws IOException`
Lee un int Lanza `EOFException` si se acaba el fichero	`int readInt() throws IOException`
Lee bytes convirtiéndolos a caracteres hasta encontrar un final de línea	`String readLine() throws IOException`

FUNCIONES RANDOMACCESSFILE

Descripción	Declaración
Escribe un array de bytes	`void write(byte[] b) throws IOException`
Escribe un double	`void writeDouble(double d) throws IOException`
Escribe un int	`void writeInt(int i) throws IOException`
Escribe los caracteres de un string convirtiéndolos primero a bytes (sólo vale para caracteres de 8 bits)	`void writeBytes(String s) throws IOException`
Cerrar el fichero	`void close() throws IOException`

OTRAS FUNCIONES REPRESENTATIVAS

- **getFilePointer()**: devuelve un puntero a una posición
- **seek(long pos)**: Establece la posición al puntero
- **length():** Devuelve el tamaño del fichero en bytes
- **skipBytes(int salto):** Desplaza el puntero un salto de posiciones

Tema 1.2 JAVA.NIO

JAVA.NIO

- Es una nueva biblioteca de E/S
- Funciona con canales y buffers, y por otro lado, funciona con streams.
- Incluye mejoras de permitir la navegación entre directorios, soporte para reconocer enlaces simbólicos, leer atributos de ficheros como permisos e información como última fecha de modificación, soporte de entrada/salida asíncrona y soporte para operaciones básicas sobre ficheros como copiar y mover ficheros.

CLASES PRINCIPALES

- **Path:** es una abstracción sobre una ruta de un sistema de ficheros, su principal función es manejar rutas de ficheros. No tiene porque existir en el sistema de ficheros pero si si cuando se hacen algunas operaciones como la lectura del fichero que representa. Puede usarse como reemplazo completo de java.io.File pero si fuera necesario con los métodos File.toPath() y Path.toFile() se ofrece compatibilidad entre ambas representaciones.
  a. Con la clase *Path* se pueden hacer operaciones sobre rutas como obtener la ruta absoluta de un *Path* relativo o el *Path* relativo de una ruta absoluta, de cuanto elementos se compone la ruta, obtener el *Path* padre o una parte de una ruta.
- **Files**: es una clase de utilidad con operaciones básicas sobre ficheros.
- **FileSystems**: otra clase de utilidad como punto de entrada para obtener referencias a sistemas de archivo.

CLASE PATH

- static: un método estático es aquel que se puede acceder sin que se tiene se tenga que crear un objeto de la clase
  - of(String path)
  - of(String path, String … more)
- no-static (se tiene que crear un objeto de la clase path)
  - getParent()
  - normalize()
  - toAbsolutePath()
  - toFile()
  - toString()
  - toUri()

CLASE FILES

- Todas las funciones de la clase file son estáticas- NO HAY QUE CREAR OBJETOS
- copy (Path source, Path target)
- copy(Path source, OutputStrean out)
- copy(InputStream in, Path target)
- createDirectoy(Path dir)
- createFile(Path path)
- delete(Path path)
- deleteExists(Path path)
- exists(Path path)
- readAllLines(Path path)
- readAllBytes(Path path)
- write(path path, byte[] Bytes)
- writeString(Path path, CharSequence chr)

EXTRA EN LA CLASE FILES

- La clase files nos permite copiar y mover archivos (esta opción no se encontraba en la clase File)
- No necesitamos el uso de FileUtils

CREAR UN FICHERO DEFINIDO EN LA RUTA

```
package ListarArchivos;
import java.io.*;
import java.nio.*;
import java.nio.file.Path;
public class Main {
	public static void main(String[] args) {

		//Stream es una forma de Java dónde utilizamos las colecciones como ETL
		//Extraemos la información, transformamos y la cargamos de forma modificada

		//Definimos una ruta que no existe
		Path path = Path.of("fichero.txt");
		System.out.println(path.normalize().toString());
		System.out.println(path.toAbsolutePath());


		//Crear un fichero
		File file = path.toFile();



		try {
			file.createNewFile();
		}
		catch(IOException ex) {
			System.out.println("Error al crear el fichero en el path: "+path.toString());;
		}

		//Collect coge todos los elementos del string y lo crea una lista
	}
}
```

LISTAR ARCHIVOS

```
private static void listarArchivos(Path path) {
	try {

		Stream<Path> ficheros =Files.list(path);
		//Error porque hay que castear el resultado
		ArrayList<Path> lista = (ArrayList<Path>) ficheros.collect(Collectors.toList());

		for(int i =0; i<lista.size();i++) {
			System.out.println(lista.get(i));
			if(lista.get(i).toFile().isDirectory()) {

				listarArchivos(Path.of(lista.get(i).toString()));
			}
		}

	}catch(IOException ex) {
		System.err.println("No se ha podido recuperar la lista del directorio: "+path.toString());
		System.err.println(ex.getMessage());
	}
}
```

EJEMPLO DE 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();
}
```

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");
}
```

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");
}
```

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());
}
```

LEER 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());
}
```

FICHERO XML

- XML: Lenguaje extendido de marcado
- Permite crear estructuras anidando elementos uno dentro de otro y definir el contenido de cada elemento.
- Este tipo de fichero siempre se organiza mediante una estructura de forma de árbol.
  - No se pueden dar elementos cíclicos
  - Un elemento no puede ser padre e hijo de otro elemento

FICHERO XML (2)

- Son ficheros muy fáciles de leer, ya que siguen una estructura concreta y tienen una sintaxis sencilla.
  - Los elementos se definen mediante una etiqueta y ésta puede 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 0..* 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.

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>
```

FICHERO XML (3)

- 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 leyende 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

- La primera forma para poder leer los ficheros es a través de DOM:
  - **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

FICHEROS XML- LECTURA A TRAVÉS DE DOM

- Antes de empezar a leer hay que tener en cuenta que los espacios en blanco dentro de una etiqueta los lee como texto, por ello, debemos tener cuidado al tratar el fichero.
  a. Se recomienda eliminar los espacios en blanco: https://codebeautify.org/xmlviewer
- Pasos para leer un documento:
  a. Crear el DocumentBuilderFactory
  b. Crear el DocumentBuilder
  c. Crear el Document, este punto es muy importante ya que cargará todo el documento en memoria.
  d. Leer las propiedades del documento deseadas

FICHEROS XML- ESCRITURA A TRAVÉS DE DOM

- 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

FICHERO XML- LECTURA 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 lemento
  - DefaultHandler → Clase abstracta que debemos implementar, contiene las llamadas a los eventos

FICHERO XML- LECTURA 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 podemos leer los atributos si los contiene
  - endElement()
  - characters() Caracteres que contiene el elemento

FICHERO XML- LECTURA 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

EJERCICIOS

- A partir del siguiente documento XML, 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 género
16. Traduce todas las etiquetas del XML y guardarlo en un fichero llamado libros.xml
    - Catalog →Catálogo
    - Book → Libro
    - Title → Título
    - Genre → Género
    - Price → Precio
    - Public_date → Fecha de publicación
    - Description → Descripción

MATERIAL ÚTIL

- Importar/Exportar proyectos de Java en Eclipse → https://www.youtube.com/watch?v=dFDxJI34R98