



UNIDAD DE TRABAJO 7

ENTRADA Y SALIDA DE INFORMACIÓN

Módulo de Programación



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Autor: Fran Gómez

1. Interfaces de entrada y salida
2. Flujo de datos
 - a. Tipos de flujos
 - b. Flujos predeterminados
 - c. Clase Scanner
 - d. Utilización de flujos mediante sus clases
3. Ficheros y directorios
 - a. Tipos
 - b. Creación y eliminación
 - c. Escritura y lectura
 - d. Otras operaciones
4. Serialización
5. XML en Java

Interfaces de entrada y salida



Los programas deben poder comunicarse con otros programas o bien con el usuario utilizando algún dispositivo



Para realizar esta comunicación se usan interfaces (en un sentido más general):

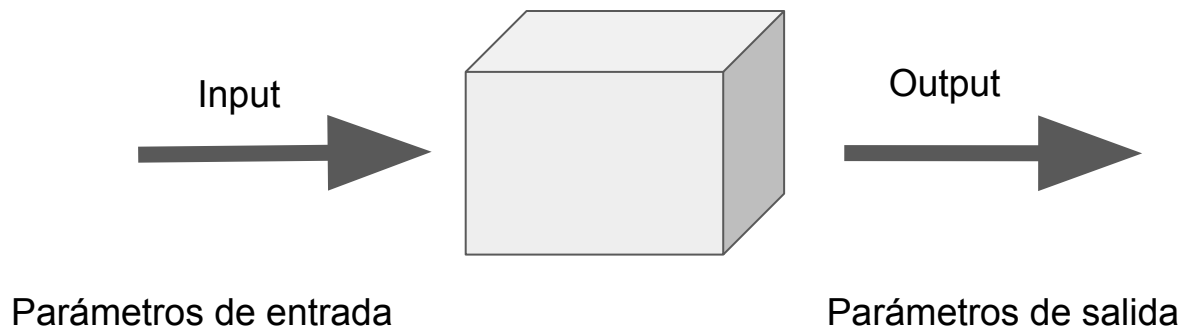
- **Interfaces de Entrada**
- **Interfaces de Salida**



🔍 Search Google or type a URL



Cualquier **programa** que desee interactuar o trabajar con datos debe tener la capacidad de leer y/o escribir datos. \Rightarrow **argumentos y parámetros**



¿Cómo introducimos datos y cómo los leíamos hasta ahora?

```
class HolaMundo {  
    public static void main (String[] args) {  
        String nombre = "Fran";  
        System.out.println("Hello, World: " + nombre);  
    }  
}
```

Hello, World: Fran

Process finished with exit code 0

Name: HolaMundo

☐ Store as project file

Build and run

[Modify options](#) Alt+M

java 17 SDK of '1daw-prog'

org.ieslosremedios.daw1.prog.ut7.HolaMundo

Fran

Press Alt for field hints

```
class HolaMundo {  
    public static void main (String[] args) {  
        System.out.println("Hello, World: " + args[0]);  
    }  
}
```

Hello, World: Fran

Process finished with exit code 0

En Java la entrada y salida de datos se lleva a cabo mediante flujos de datos ⇒ **Streams**

Flujo de datos ⇒ **Secuencia** ordenada de datos que se transmite en **serie**, es decir, uno detrás de otro, desde un origen hasta un destino

El flujo supone una capa de **abstracción** que nos permite olvidarnos de qué interfaz tenemos en el origen o en el destino, en ambos casos usamos el flujo de la misma manera, ya sea escribir en una impresora, leer de un fichero, mostrar algo en la pantalla, etc.



Según el sentido del flujo pueden ser:

- Flujo de **Entrada** ⇒ Se **LEE** información
- Flujo de **Salida** ⇒ Se **ESCRIBE** información

También existen flujos de entrada y salida si se usan en ambos sentidos

Según la forma de acceder al flujo pueden ser:

- Flujo de **acceso directo** \Rightarrow Se accede directamente al elemento deseado
- Flujo de **acceso secuencial** \Rightarrow Para acceder al elemento deseado hay que pasar por los anteriores

Según el tipo de datos del flujo pueden ser:

- Flujo de **caracteres** ⇒ los datos se codifican en caracteres. Ej: Unicode o ASCII
- Flujo de **bytes** ⇒ los datos no se codifican. Ejemplo los ficheros binarios.

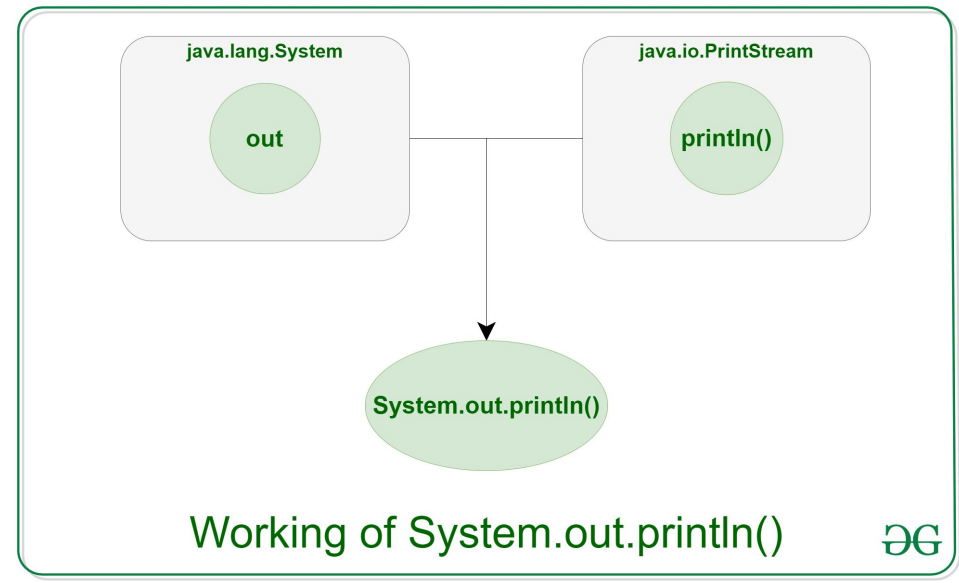
```
MainEscribirTexto.java fichero_texto.txt ×
1 Esto es la primera línea de texto
2 Esto es otra línea de texto
3 Esto es la última línea de texto
4
```

0000	FF	D8	FF	E1	1D	FE	45	78	69	66	00	00	49	49	2A	00
0010	08	00	00	00	09	00	0F	01	02	00	06	00	00	00	7A	00
0020	00	00	10	01	02	00	14	00	00	00	80	00	00	00	12	01
0030	03	00	01	00	00	00	01	00	00	00	1A	01	05	00	01	00
0040	00	00	A0	00	00	00	1B	01	05	00	01	00	00	00	A8	00
0050	00	00	28	01	03	00	01	00	00	00	02	00	00	00	32	01
0060	02	00	14	00	00	00	B0	00	00	00	13	02	03	00	01	00
0070	00	00	01	00	00	00	69	87	04	00	01	00	00	00	C4	00
0080	00	00	3A	06	00	00	43	61	6E	6F	6E	00	43	61	6E	6F
0090	6E	20	50	6F	77	65	72	53	68	6F	74	20	41	36	30	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	B4	00	00	00
00B0	01	00	00	00	B4	00	00	00	01	00	00	00	32	30	30	34
00C0	3A	30	36	3A	32	35	20	31	32	3A	33	30	3A	32	35	00
00D0	1F	00	9A	82	05	00	01	00	00	00	86	03	00	00	9D	82
00E0	05	00	01	00	00	00	8E	03	00	00	00	90	07	00	04	00

Mientras se ejecuta un programa, se mantienen abiertos algunos flujos de datos para ser usados sin necesidad de crearlos manualmente (como luego veremos).

Estos son:

- `System.out`
- `System.err`
- `System.in`



PrintStream

<code>System.out.write</code>	Escribe los bytes enviados como argumento en el flujo.
<code>System.out.flush</code>	Vacía el flujo por la salida (out). Es necesario hacerlo en el write si no está en automático o el byte no representa una nueva línea.
<code>System.out.print</code>	<code>System.out.write</code> + <code>System.out.flush</code> . Admite diferentes tipos de datos (no bytes como el write).
<code>System.out.println</code>	Como el print pero con retorno de carro
<code>System.out.printf</code>	Como el print pero con formato.

InputStream

<code>System.in.read ()</code>	Devuelve el siguiente byte del flujo. Hay que pasarlo al tipo que queramos mediante casting.
<code>System.in.read (byte[] b)</code>	Almacena en b los bytes del flujo

Ejercicio 1



```
// Pedir al usuario que introduzca 4 caracteres por teclado
```

```
// 1. Imprimir el primero con el write
```

```
// 2. Imprimir el segundo con print
```

```
// 3. Imprimir el tercero con println
```

```
// 4. Imprimir el cuarto con printf
```

Ejercicio 2



Permitir al usuario introducir un número indeterminado de caracteres.

Primero habrá que imprimir las instrucciones para el usuario: "Introduzca varios caracteres"

Luego habrá que añadir qué condición debe introducir el usuario para indicar que ya ha terminado de introducir caracteres, con lo que quedaría algo así: "Introduzca varios caracteres y después pulse intro para finalizar"

Lo último que haremos es un hola mundo donde se pida el nombre al usuario, utilizando el código anterior.

Ej: output:"Introduzca su nombre"

input: Fran

output: Hola Fran!

Ejercicio 3



¿Qué hace el siguiente código? Piénsalo y luego pruébalo para confirmarlo.

```
public static void main(String args[]) {  
    byte b[] = new byte[5];  
    try {  
        System.in.read(b);  
    } catch (IOException ioe) {  
        System.out.println(ioe);  
    }  
    String s = new String(b);  
    System.out.println(s);  
}
```

Más sencilla de usar que los métodos de `InputStream` y `PrintStream`

- Es capaz de usar tipos de datos primitivos
- Es capaz de leer a nivel de línea (tokens)

Los tokens por defecto serán grupos de caracteres alfanuméricos separados por espacios. Aunque también podrían definirse qué queremos que sea un token, por ejemplo una expresión regular.

Ejercicio 4



1º DAW
Programación

UT 6:
Excepciones

Hacer un *hola mundo*, pidiendo al usuario su nombre completo y edad. Controlar que la edad sea un número entero.

java.io.Console extends Object implements Flushable

- Alternativa a System.out que añade algunos métodos interesantes
- No se puede ejecutar desde el IDE sino desde consola.
- Para crearla: *Console console = System.console();*
- Métodos:
 - String readLine()
 - char[] readPassword()
 - Console printf(String format, Object... args)

Ejercicio 5



1º DAW
Programación

UT 6:
Excepciones

Escribe un programa que pida al usuario su nombre de usuario y contraseña y los compare con el valor de una constante. En caso de que introduzca mal sus credenciales, permitirle 3 intentos.

1. Crear el Stream (En los estándar es automático)
2. Leer-escribir
3. Cerrar el Stream (En los estándar es automático)

Nos permiten **abstraernos** de los detalles del dispositivo donde queramos leer o escribir → Usamos objetos en lugar de acceder directamente a los drivers.

Estos objetos son los flujos → Definidos mediante las clases del paquete:

java.io

Según el tipo de información que manejan los flujos, usaremos unas clases u otras:

- Flujos de bytes → InputStream y OutputStream (Clases abstractas)
- Flujos de caracteres → Reader y Writer (Clases abstractas)

- Entrada → [InputStream](#)

<code>int read()</code>	Lee el siguiente byte desde el <code>InputStream</code> . Si retorna -1 no se pueden leer más bytes
<code>int available()</code>	Devuelve el número de bytes que se pueden leer del <code>InputStream</code> sin un bloqueo
<code>void close()</code>	Cierra el <code>InputStream</code>

- Salida → [OutputStream](#)

<code>int write(int b)</code> <code>int write (byte[] a)</code>	Escribe uno o varios bytes en el <code>OutputStream</code>
<code>int flush()</code>	Vacía el flujo de salida actual del <code>OutputStream</code>
<code>void close()</code>	Cierra el <code>OutputStream</code> y escribe lo que haya en ese momento

- `FileOutputStream`
- `BufferedOutputStream`
- `DataOutputStream`
- `ByteArrayOutputStream`
- `PrintStream`
- `PipedOutputStream`
- `FileInputStream`
- `BufferedInputStream`
- `DataInputStream`
- `ByteArrayInputStream`
- `SequenceInputStream`
- `PipedInputStream`

Ejercicio 6:

Investiga cada una de estas clases y crea un ejemplo para explicar su funcionamiento al resto de la clase.

Elabora la jerarquía de clases e indica los paquetes a los que pertenecen.

¿Eres capaz de encontrar más flujos a parte de los de la lista?

Clases para manejar flujos de caracteres



1º DAW
Programación

UT 6:
Excepciones

- Entrada → **Reader**

<code>int read()</code>	Lee el siguiente byte desde el Reader. Si retorna -1 no se pueden leer más bytes
<code>long skip(n)</code>	Hace que se omitan los n primeros caracteres
<code>void close()</code>	Cierra el InputStream

- Salida → **Writer**

<code>int write(int c)</code> <code>int write (String s)</code> <code>int write (char[] c)</code>	Escribe uno o varios bytes en el OutputStream
<code>int flush()</code>	Vacía el flujo de salida actual del Writer
<code>void close()</code>	Vacía el flujo de salida actual del Writer y lo cierra

- `FileWriter`
- `BufferedWriter`
- `CharArrayWriter`
- `PrintWriter`
- `StringWriter`
- `PipedWriter`
- `FileReader`
- `BufferedReader`
- `LineNumberReader`
- `CharArrayReader`
- `StringReader`
- `PipedReader`

Ejercicio 7:

Investiga cada una de estas clases y crea un ejemplo para explicar su funcionamiento al resto de la clase.

Elabora la jerarquía de clases e indica los paquetes a los que pertenecen.









Añade al ejemplo la utilización de los métodos `mark` y `reset`, explicando para qué los usas.

Ficheros y Directorios



- Permiten persistencia de datos →

Almacenar los datos para recuperarlos en sucesivas ejecuciones del programa.

 .git	11/04/2023 9:57	File folder	
 .idea	27/03/2023 19:18	File folder	
 org	27/03/2023 17:24	File folder	
 out	27/03/2023 17:30	File folder	
 .gitattributes	27/03/2023 17:24	Git Attributes Sour...	1 KB
 .gitignore	27/03/2023 17:30	Git Ignore Source ...	1 KB
 1daw-prog-22_23.iml	27/03/2023 17:26	IML File	1 KB
 README.md	27/03/2023 17:24	Markdown Source...	1 KB

Ficheros y Directorios: Tipos de ficheros



1º DAW
Programación

UT 6:
Excepciones

Texto	Binario
Legibles para un humano	Legibles para los programas
Acceso secuencial	Acceso directo

Ficheros y Directorios: Registros



1º DAW
Programación

UT 6:
Excepciones

Conjunto de datos que deben ser accedidos juntos.

Ej: un objeto.

Ficheros y Directorios: Creación y eliminación



Clase java.io.**File**

- Constructores
 - *File (String pathname)*
- Métodos
 - *boolean createNewFile()*
 - *boolean mkdir()*
 - *boolean delete()*

La clase File representa un fichero en Java. No lo abre. No lo crea. Sólo lo representa. Sí podemos obtener información sobre él.

- Acceso secuencial
 - FileReader, FileInputStream
 - (String **path**)
 - (File **file**)
 - FileWriter, FileOutputStream
 - (String **path** [, boolean **append**])
 - (File **file** [, boolean **append**])
- Acceso directo
 - RandomAccessFile
 - (String **path** ,String **modo**)
 - (File **file** ,String **modo**)

En el constructor se puede indicar tanto la ruta como el objeto File.

En el caso de escritura, también si añadimos o sobrescribimos.

El modo de acceso:

- r → lectura
- rw → lectura y escritura

Ficheros y Directorios: Escritura y lectura secuencial



Se realiza como habíamos hecho hasta ahora con las clases relativas a los flujos.

Ejercicio 8: Crea un nuevo fichero de texto utilizando un editor de texto plano del sistema operativo. Después, desde Java, añade algún contenido al final del fichero. Finalmente muestra el contenido del fichero por consola desde Java.

Ficheros y Directorios: Escritura y lectura directa



Se utiliza la clase `java.io.RandomAccessFile`

Métodos:

- `seek` → Desplaza puntero del archivo a la posición indicada (empezando en 0 como los arrays)
- `read` → Lee un byte
- `write` → Escribe un byte
- `readTipo/writeTipo` → Lee/Escribe un dato del tipo primitivo indicado.
- `length` → Longitud del fichero

Nota: cada tipo de dato ocupa un número de bytes distinto en Java. Ej: `char` ocupa 2 bytes en Unicode.

Ficheros y Directorios: Escritura y lectura directa

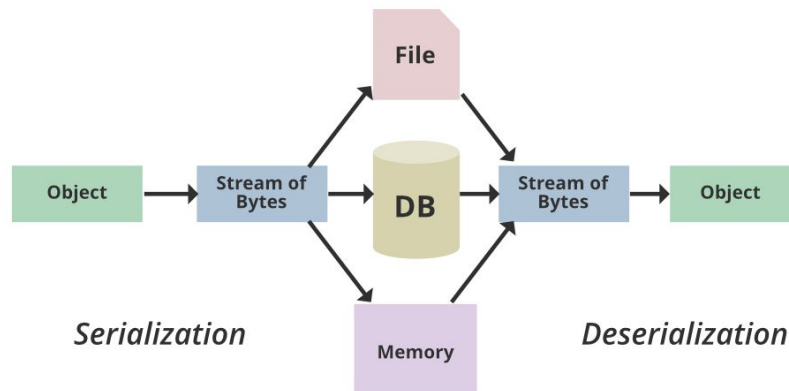


Ejercicio 9:

1. Crea un nuevo fichero eliminando el que ya existía en su caso.
2. Abre un flujo de acceso aleatorio al fichero e introduce la cadena "defg".
3. Imprime por consola el contenido del fichero
4. Añade al principio la cadena "abc" y al final la cadena "hij"
5. Sustituye las vocales por "*"

Proceso mediante el que convertimos objetos a secuencias de bytes para así poder transmitirlos (por ejemplos a un fichero binario) y luego poder volver a transformarlo en un objeto mediante un proceso de deserialización.

Serialization & Deserialization in Java



Para que un objeto se serializable, su clase tiene que cumplir:

- Implementar la interfaz Serializable
- Sus atributos deben ser de tipos primitivos o que implementen Serializable
- Declarar el atributo estático privado serialVersionUID

- Los atributos que no queramos serializar deben marcarse con el modificador “transient”
- Los atributos estáticos tampoco se serializan, su valor es el mismo para todos los objetos de la clase.

- Serializar → `ObjectOutputStream`
 - `writeObject` → Escribe un objeto en el flujo
- Deserializar → `ObjectInputStream`
 - `readObject` → Lee un objeto (como tipo `Object`) del flujo

Ejercicio 10:

1. Crea un código donde se serialize un objeto de la clase Persona y se escriba en un fichero. Esta persona deberá tener el nombre de algún personaje famoso.
2. Pasa el fichero a tu compañero para que intente deserializarlo y desvelar de qué personaje se trata.

- Datos + Estructura
- Human readable
- Almacenar y Transmitir información fácilmente

```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

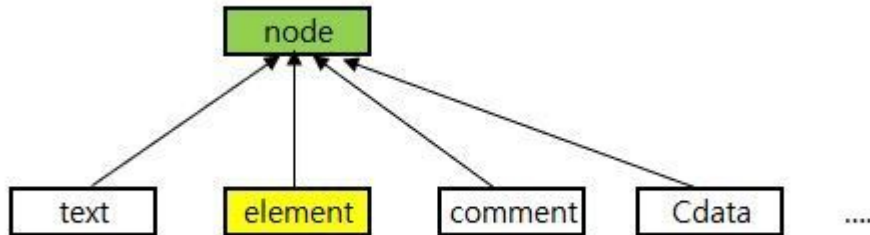
XML



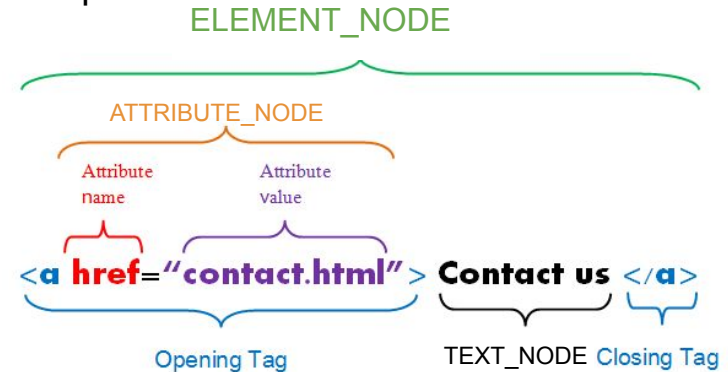
DOM: Document Object Model

- Modelo utilizado para representar ficheros tipo XML o derivados como HTML.
- No depende del lenguaje, se utiliza en cualquier lenguaje
- Existen diferentes librerías o API's para manejarlo

Node → Cualquier objeto en el documento



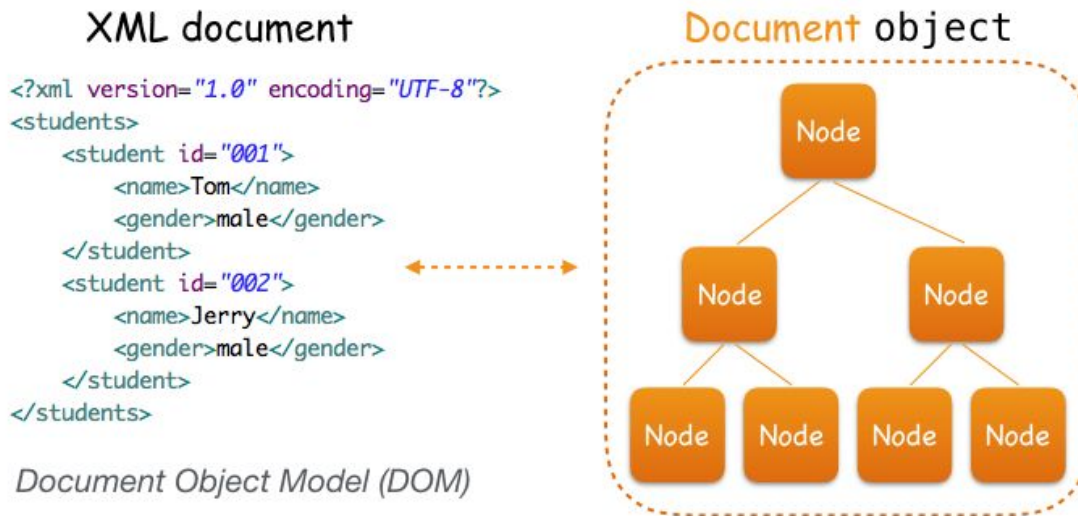
Element → Un tipo de node que representa a un tag completo.



DOM: Document Object Model



- Los nodos guardan una estructura jerarquizada, en forma de árbol.
- Por tanto tenemos: nodo raíz, nodo padre, nodo hijo, nodo hoja.



XML: Transmitir información

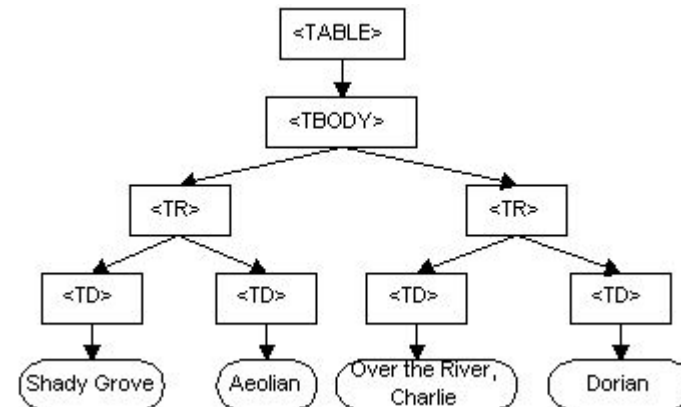


XML

Deserializar o Parsear

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

DOM



Serializar

Principales APIs utilizadas en Java para manejo del DOM:

- SAX
- DOM → paquete org.w3c.dom

Veremos la API DOM → Se trata de una API, es decir, una colección de interfaces que aporta una especificación (el qué), y que son implementadas por una librería de clases (el cómo), normalmente Xerces.

Principales interfaces

Tipos Node	Qué representa	Posibles nodos hijo
Document	Representa todo el documento XML. Conceptualmente es la raíz del árbol	Element, Comment
Node	Cualquier objeto en el XML	
Element	Representa un elemento (delimitado por etiquetas de principio y fin)	Element, Text, Comment
Attr	Atributo de un objeto Element	Text
Comment	Comentarios	--
Text	Texto	--
NodeList	TAD que representa una lista de nodos	

Principales métodos

Interfaz	Método	Descripción
Element	getDocumentElement()	Accede al nodo raíz del documento. Devuelve un Element.
Element	getElementsByTagName(String tagname)	Accede a todos los elementos de la etiqueta indicada. Devuelve un NodeList.
Node	getNodeType()	Devuelve un número entero indicando el tipo de nodo: 1 → ELEMENT_NODE 2 → ATTRIBUTE_NODE 3 → TEXT_NODE
NodeList	getLength()	Número de nodos
NodeList	item(int index)	Accede a un node por su índice

Principales métodos

Interfaz	Método	Descripción
Element	getAttribute(String name)	Devuelve String con el valor del atributo
Node	getTextContent()	Devuelve String con el texto del nodo tipo Element
Document	createElement(String name)	Crea un nodo de tipo elemento
Text	createTextNode(String name)	Crea un nodo de tipo texto
Node	appendChild(Node node)	Añade un nodo como hijo del nodo del parámetro implícito

```
// Cargamos fichero que vamos a leer
File file = new File( pathname: "org/ieslosremedios/daw1/prog/ut7/xml/ejemplo.xml");
```

```
// Parseamos el fichero al Document
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(file);
```

```
// Accedemos a todos los nodos con el tag "estudiante"
NodeList estudiantes = document.getElementsByTagName("estudiante");
// Recorremos todos esos nodos
for (int i = 0; i < estudiantes.getLength(); i++) {
    Node nodeEstudiante = estudiantes.item(i);
    // Filtramos todos los que son nodos de tipo elemento
    if (nodeEstudiante.getNodeType() == Node.ELEMENT_NODE) {
        Element elementEstudiante = (Element) nodeEstudiante;
        System.out.println("Nombre del estudiante: " + elementEstudiante.getTextContent());
    }
}
```



Escritura de XML

```
// Creamos el documento vacío para añadirle a continuación los nodos
// En este caso lo hago todo en una sola línea
Document document = DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();

// Creamos el nodo raíz
Element estudiantes = document.createElement( tagName: "estudiantes");
// Hacemos que cuelgue del documento (estructura de árbol)
document.appendChild(estudiantes); // Creamos el primer nodo y lo colgamos de su padre, el nodo raíz. -->
Element estudianteFran = document.createElement( tagName: "estudiante");
estudiantes.appendChild(estudianteFran);

// Creamos un nodo de texto que será el valor del elemento anterior
Text fran = document.createTextNode( data: "Fran");
// y lo colgamos del nodo anterior --> <estudiante>Fran</estudiante>
estudianteFran.appendChild(fran);

// Clases necesarias para finalizar la creación del archivo XML
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(new File( pathname: "org/ieslosremedios/daw1/prog/ut7/xml/otro.xml"));

// Se realiza la transformación, de Document a Fichero.
transformer.transform(source, result);
```


Ejercicio 11

Vamos a crear una lista de contactos. Crearemos para ello una lista de Personas donde almacenaremos al menos: nombre, dirección y teléfono.

Pasaremos esa lista a un fichero XML.

Subiremos este fichero a una carpeta compartida, para igual que en el ejercicio anterior, compartirlo con un compañero, para que este pueda leer estos contactos y añadirlos a su propia lista.

Carpeta compartida:

https://drive.google.com/drive/folders/1SbpTlmsvao8kNNmruz1GhY_B1Hx3UZTz?usp=sharing

Actividades finales



1º DAW
Programación

UT 6:
Excepciones

1. Práctica
2. Portfolio
3. Examen
4. Feedback