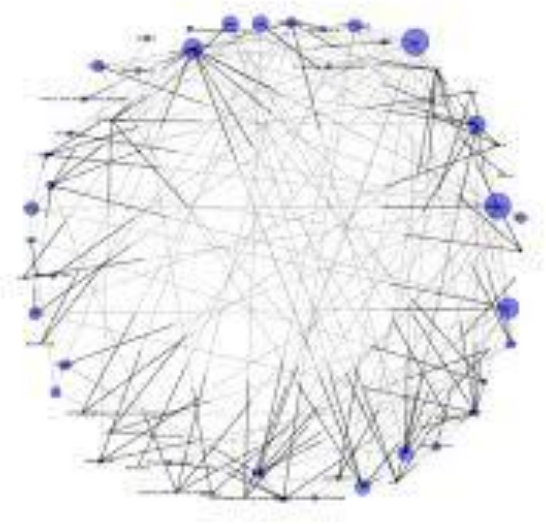
DAM
ACCESO A DATOS



UD1
MANEJO DE FICHEROS EN JAVA
3. ACCESO A FICHEROS XML CON DOM

Introducción

Un metalenguaje, es un lenguaje que se utiliza para definir otros lenguajes y sus gramáticas.

- XML (*eXtensible Markup Language – Lenguaje de Etiquetado Extensible*) es un **metalenguaje**.
- **XML** fue creado por el W3C (*Consorcio World Wide Web*) para jerarquizar e estructurar la información así como para describir los contenidos dentro de un documento elaborado con un lenguaje de marcas, por ejemplo, HTML.
- Actualmente, XML es un **estándar para el intercambio de información estructurada entre distintas plataformas**. XML puede ser utilizado para almacenar información de bases de datos, hojas de cálculo, ficheros

Ficheros XML

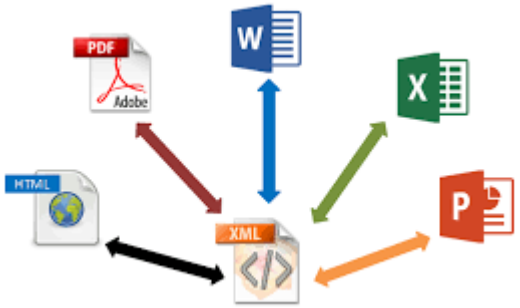
- Un fichero XML está compuesto por una serie de **etiquetas limitadas por los símbolos <** y **>**. El significado de las etiquetas se define en un fichero específico que aporta el significado y la estructura de un documento XML concreto.
- Las etiquetas XML pueden tener información adicional en forma de atributos.
- En el ejemplo siguiente podemos observar como se almacenan información de alumnos con etiquetas como <alumno> que posee un atributo denominado curso.

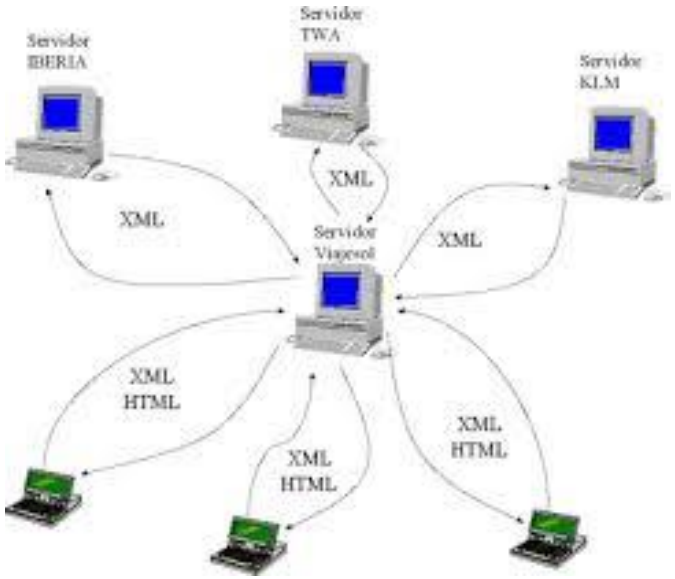
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<alumnos>
    <alumno curso="C90">
      <nombre>Pilar</nombre>
      <apellidos>Pérez Sousa</apellidos>
    </alumno curso="C80">
    <alumno>
      <nombre>Carmen</nombre>
      <apellidos>Novoa Real</apellidos>
    </alumno>
</alumnos>
```

alumnos es el elemento **raíz** que contiene información sobre distintos alumnos, para cada uno almacena el **curso** (atributo) y el **nombre** y **apellidos**

Ficheros XML: usos

- Los archivos XML se pueden utilizar para:
  - proporcionar datos a una base de datos,
  - almacenar copias de partes del contenido de la base de datos,
  - almacenar la configuración de programas,
  - para ejecutar comandos en servidores remotos, como en el protocolo SOAP *(Simple Object Access Protocol), etc.*





Procesadores XML

- Los procesadores XML, también llamados ***parser***, ponen a disposición de las aplicaciones los contenidos de un documento XML, a la vez que detecta errores.
- Entre los procesadores o ***parser*** más utilizados están:
  - **DOM** (*Modelo de Objetos de Documento*).
  - **SAX** (*API Simple para XML*).
- Los procesadores son independientes del lenguaje de programación donde se utilizan y existen versiones particulares para Java, VisualBasic, C, etc.
- Un procesador XML lee el archivo XML y lo convierte en una representación interna disponible para ser usado por otros programas.

Se denomina **parsear** al análisis que se hace de un documento o dato para ponerlo a disposición de un sistema para su proceso.

DOM: introducción

- El **W3C** ha estandarizado la API de manejo de ficheros XML desde un lenguaje de programación en el **Modelo de Objeto de Documento (DOM)**, basada en una estructura en forma de árbol jerárquico.
- **Los procesadores DOM**:
  - Proporcionan una interfaz (API) al programador para poder crear y acceder de forma sencilla y homogénea a todos los elementos de un fichero XML.
  - Almacenan la estructura del documento en memoria en forma de árbol con **nodos padre**, **nodos hijos** y **nodos finales**. Una vez creado el árbol, se van recorriendo los diferentes nodos para poder ser procesador desde el lenguaje específico de programación.

DOM: ejemplo de XML

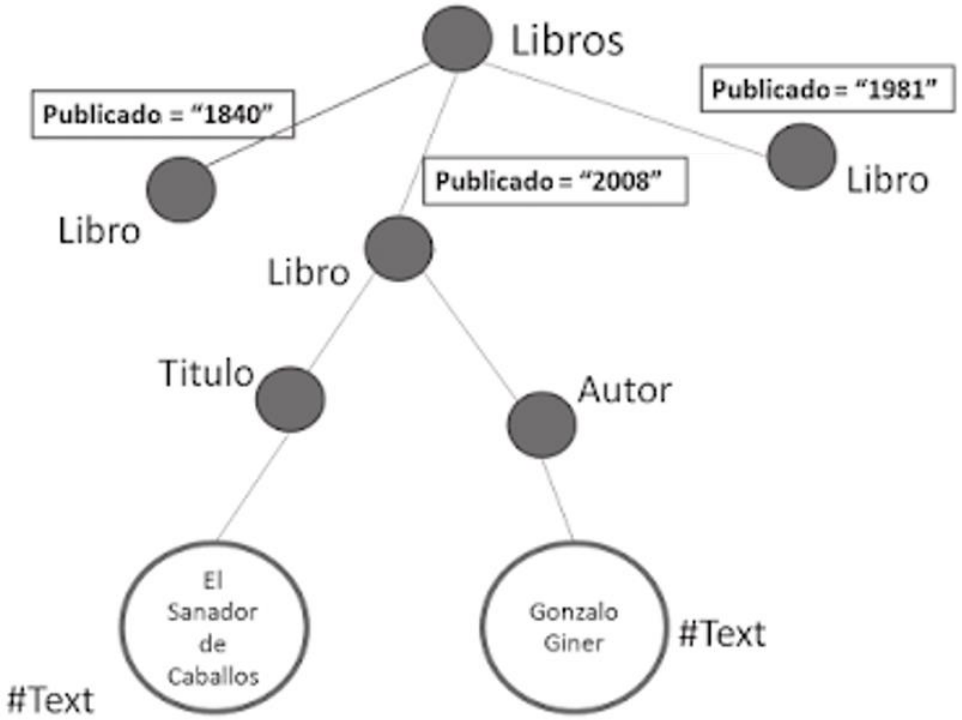
El siguiente documento XML muestra una serie de libros.
Cada uno de ellos está definido por un atributo *Publicado*, un texto que indica el año de publicación del libro y por dos elementos hijo: *Título* y *Autor*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Libros
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:noNamespaceSchemaLocation='LibrosEsquema.xsd'>

    <Libro Publicado="1840">
        <Titulo>El Capote</Titulo>
        <Autor>Nikolai Gogol</Autor>
    </Libro>
    <Libro Publicado="2008">
        <Titulo>El Sanador de Caballos</Titulo>
        <Autor>Gonzalo Giner</Autor>
    </Libro>
    <Libro Publicado="1981">
        <Titulo>El Nombre de la Rosa</Titulo>
        <Autor>Umberto Eco</Autor>
    </Libro>
</Libros>
```

DOM: representación en árbol

- El árbol DOM se ha creado en base al documento XML anterior.
- Se ha simplificando la información para no hacerlo demasiado extenso y solo se desarrolla uno de los libros.



- Para generar el árbol, en principio, solo se comprueba que el XML esté bien formado.
- **Libros** es el elemento raíz que tiene tres hijos.
- Cada elemento **Libro** tiene asociado un atributo **Publicado** con la información del año de publicación.
- Cada libro tiene dos elementos: **Titulo** y **Autor**.
- Cada contenido de un elemento está calificado como **#Text**.

DOM: acceso a ficheros XML desde Java

- Para trabajar con DOM en Java necesitamos los paquetes:
  - **org.w3c.dom**, conocido como *JSDK* ,
  - **javax.xml.parsers** del API estándar de Java para procesamiento XML.
  - **javax.xml.transform** que permite especificar una fuente y un resultado que pueden ser ficheros, flujo de datos o nodos DOM, entre otros.
- Los objetos que se utilizarán en Java para manejar el árbol DOM generado a partir de un fichero XML son:
  - **Document**. Representa el documento XML al que se le tiene que indicar un elemento raíz. Permite crear nuevos nodos en el documento a partir de un elemento raíz
  - **Element**. Representa a cada uno de los elementos del árbol DOM.
  - **Node**: representa a cualquier nodo del documento.
  - **NodeList**: Contiene una lista con todos los nodos hijos de un nodo dado.
  - **Attr**. Permite acceder a los atributos de un nodo.
  - **Text**: Son los datos carácter de un elemento.
  - **CharacterData**: Representa a los datos carácter del documento y facilita herramientas para manipular esta información.
  - **DocumentType**: Proporciona información contenida en la etiqueta <!DOCTYPE>

DOM: ventajas e inconvenientes

- **Ventajas**:
  - Puede ser agregado un nodo (Información) en cualquier punto del árbol.
  - Puede ser eliminada información de un nodo en cualquier punto del árbol.
  - Lo anterior se ejecuta sin incurrir en las penalidades o limitaciones de manipular un archivo de alguna otra manera.
- **Inconveniente**:
  - El proceso de construcción del árbol es costoso y consume bastante recursos de memoria.

DOM-JAVA- 1. Elementos a instanciar

Crear una instancia de una factory de constructores de documentos

```
DocumentBuilderFactory factoryDocument = DocumentBuilderFactory.newInstance();
```

Crear un parser/procesador de documento XML

```
DocumentBuilder builderDocument = factoryDocument.newDocumentBuilder();
```

Crear una instancia de DOMImplementation que permite crear documento DOM

```
DOMImplementation implementacionDOM = builderDocument.getDOMImplementation();
```

DOM-JAVA- 2. Crear nodo raíz de un documento XML

Creamos un documento vacío (*document*) con el nodo raíz de nombre *personas*

```
Document documento = implementacionDOM.createDocument (null, "personas",null);
```

Indicamos la versión de XML

```
documento.setXmlVersion("1.0");
```

Hasta ahora se ha creado tan solo el elemento raíz (*personas*) de un documento DOM que tendrá una estructura arbórea y que se corresponde con el fichero XML que se muestra a continuación


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

</personas>
```

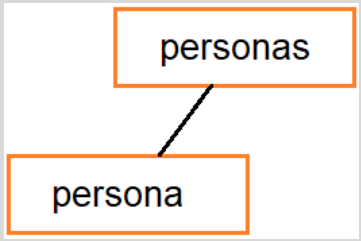
DOM-JAVA- 3. Crear elemento vacío

Crear un elemento llamado *persona* y enlazarlo con el elemento raíz del documento

```
Element elemento = documento.createElement("persona");

documento.getDocumentElement().appendChild(elemento);
```

Ahora ya tenemos en memoria un documento DOM con una estructura de árbol con un elemento raíz (*personas*) del que pende un elemento vacío (*persona*).
El documento XML que se podría generar es el siguiente:




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

        </persona>
</personas>
```

DOM-JAVA- 4. Crear elemento con texto

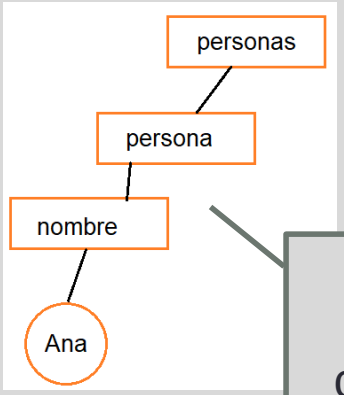
Crear un elemento *nombre* que tenga el valor, *Ana* y enlazarlo a su elemento padre (*elemento*)

```
Element hijo = documento.createElement("nombre"); //Crear un nuevo elemento

Text texto = documento.createTextNode("Ana"); //Crear un nodo con el valor

elemento.appendChild(hijo); //Enlazar en el árbol el elemento hijo a su elemento padre

hijo.appendChild (texto);//El valor se enlaza como un nodo a su elemento
```



árbol DOM creado

```
<?xml version="1.0" encoding="UTF-8"?>
<personas>
        <persona>
                <nombre>Ana</nombre>
        </persona>
</personas>
```

fichero XML que se podrá generar

DOM-JAVA- 5. Crear fichero XML a partir del documento DOM

Crear un fichero XML llamado *personas.xml* a partir de la instancia *documento* de la clase *Document*

```
Source sourceDOM =  new DOMSource(documento);

Result resultado =  new StreamResult(new File("personas.xml"));

Transformer transformer = TransformerFactory.newInstance().newTransformer();

transformer.transform(sourceDOM, resultado);
```

```
<?xml version="1.0" encoding="UTF-8"?>
<personas>
        <persona>
                <nombre>Ana</nombre>
        </persona>
</personas>
```

contenido del fichero personas.xml generado

DOM-JAVA- 6. Mostrar el contenido del documento DOM por pantalla

Mostar el documento DOM por la salida estándar System.out

```
Source sourceDOM =  new DOMSource(documento);

Result consola=new StreamResult(System.out);

Transformer transformer = TransformerFactory.newInstance().newTransformer();

transformer.transform(sourceDOM, consola);
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><personas><persona><nombre>Ana</nombre></persona></personas>
```

Documento DOM mostrado por consola

CrearDOM_1

DOM-Ejercicio 1

1. Modificar el programa que se ha desarrollado durante la explicación para incorporar:
   - A cada persona los campos:
     - **id** de tipo **Integer**.
     - **edad** de tipo **Integer**.
     - **apellidos** de tipo **String**
     - **dep** de tipo **Integer**
     - **salario** de tipo **Double**.
   - Agregar 4 elementos de tipo **persona**

2. Leer el fichero de objetos de la clase **Persona** y crear un fichero XML a partir de su contenido. Mostrar el fichero XML por pantalla.

CrearDOM_11

DOM-JAVA- 7. Agregar atributos

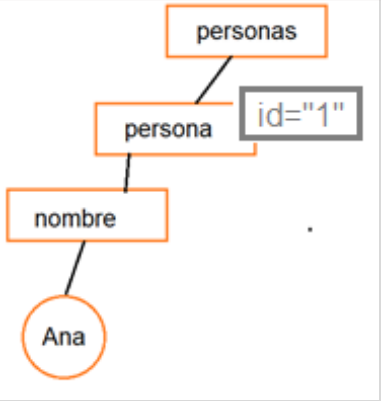
Dado un elemento del DOM, se le agrega el atributo llamado **id** con el valor **“1”**

```
//Crer un objeto atributo llamado id
Attr elAttr=documento.createAttribute(“id”);

//Dar valor al atributo recién creado id=“1”
elAttr.setValue(“1”);

//Enlazar el atributo con su correspondiente elemento
elemento.setAttributeNode(elAttr);
```

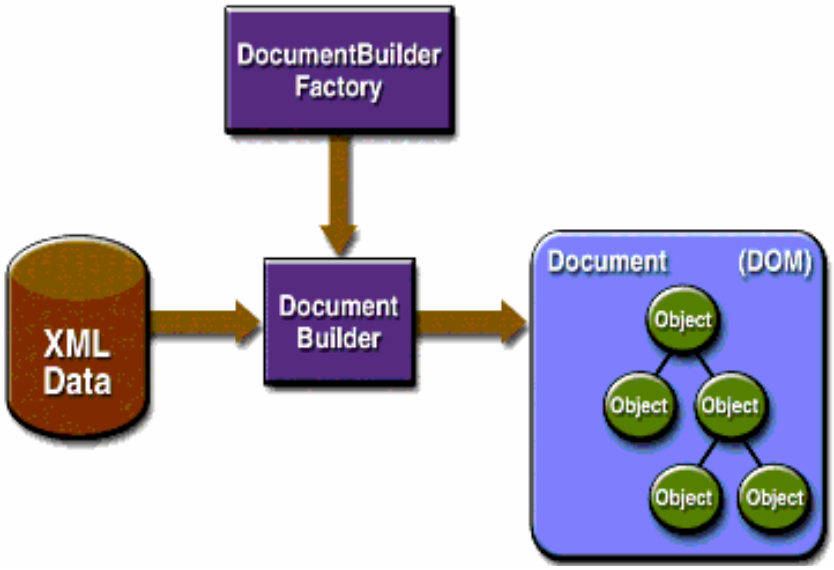
Hemos añadido un atributo al elemento persona.
Podemos ver la representación DOM con el atributo añadido y el XML que se podría generar




```
<?xml version="1.0" encoding="UTF-8"?>
<personas>
        <persona id="1">
                <nombre>Ana</nombre>
        </persona>
</personas>
```

DOM. Resumen

- En el gráfico podemos ver un esquema de las clases principales implicadas en la generación de un documento DOM en memoria.



- La clase DocumentBuilderFactory se utiliza para obtener una instancia de la clase DocumentBuilder.
- Se utiliza una instancia de DocumentBuilder para crear un objeto Document que cumpla con la especificación **DOM**. Para ello necesita que se le indique el elemento raíz en su constructor.
- La instancia de Document crea un árbol DOM, que está compuesto de elementos de tipo Node**.**
- Los elementos de tipo Node pueden ser, elementos (Elements), atributos (Attrib) o contenido de los elementos (Text), entre otros.
- A partir de la instancia Document podemos.
  - Crear un fichero XML en el disco.
  - Mostrar por pantalla el contenido del árbol DOM en formato XML
  - Recorrer el árbol DOM elemento a elemento para su procesamiento (mostrar en pantalla, realizar búsquedas, modificar el árbol, etc.

DOM-Ejercicio 2

1. Modificar el ejercicio nº2 para que agregar un atributo a cada elemento **persona** que se llamará **“id”** y contendrá como valor su posición relativa en el fichero, así , el primer elemento tendrá ***id=“1”***, el segundo ***id=“2”,*** etc.

CrearDOM_111

- La **reflexión** en Java es el conjunto de operaciones que permiten analizar y modificar el contenido y características de una clase en tiempo de ejecución. Se estas operaciones se ocupa la **API Java Relection**.
- Las clases que permiten la reflexión están en el paquete **java.lang.reflect**
  - **Field** → trabajan con las propiedades de la clase
  - **Method** → trabajan con los métodos de la clase
  - **Constructor** → trabajan con los constructores de la clase
  - **Modifier** → trabajan con los modificadores de acceso de la clase, los métodos y las propiedades.
- La clase **java.lang.Class** permite conocer datos relacionados con las clases:
  - String **getName**() → Devuelve el nombre de la clase
  - Class **getSuperclass**() → Devuelve la referencia de superclase
  - int **getModifiers**() → Devuelve un valor entero que representa los modificadores de la clase especificada.
- Para saber más
  - https://www.arquitecturajava.com/el-concepto-java-reflection/
  - https://jarroba.com/reflection-en-java/
  - https://www.arumeinformatica.es/blog/java-reflection-parte-1/
    https://docs.oracle.com/javase/tutorial/reflect/

Uso en : **CrearDOM_111**
Implementación en: **Lib_FicheroSerializablePersonas.mostrarDatosObjetoFicheroSerializable**

DOM. Recorrer el árbol

- Java dispone de clases e interfaces para recorrer un fichero XML y crear una estructura DOM en memoria que permita su consulta y modificación.
- La operación de lectura de un fichero XML para convertirlo en un árbol DOM en memoria, se denomina ***parser.***
- Para recorrer el árbol DOM se utilizan las siguientes interfaces/clases/métodos, pertenecientes al paquete **org.w3c.dom**:
  - **Node**
    - **getNodeName()** → Devuelve el nombre (String) de un nodo (*Node*).
    - **getTextContent()** → Devuelve el texto (String) asociado a la etiqueta de un nodo (*Node*).
    - **getNodeType()** → Devuelve el tipo de nodo (es una constante).
      - Node.ELEMENT_NODE → tipo *Element*.
      - Node. ATTRIBUTE_NODE → tipo *Attr*.
      - Node. TEXT_NODE → tipo *Text*.
    - **getChildNodes**: devuelve un *NodeList* que contiene el conjunto de nodos hijos.
    - **getAttributes()** → Devuelve un objeto *NamedNodeMap*() (lista de atributos) de un objeto *Element*.
    - **getFirstChild**: devuelve el primero Node hijo.
    - **getLastChild**: devuelve el último Node hijo.
    - **getNextSibling**: devuelve la siguiente ocurrencia del Node.
    - **getPreviousSibling**: devuelve la anterior ocurrencia del Node.
    - **getParentNode**: devuelve el padre del Node.
  - **NodeList**: Lista de nodos.
  - **Element**
    - **getElementsByTagName(String name):** devuelve un *NodeList* que contiene los elementos hijos del nodo cuyo nombre se pasa por parámetro (* para todos los elementos).
    - **getAttribute(String name)** → Devuelve el valor (String) correspondiente al valor del atributo *name*.

DOM. Recorrer el árbol paso a paso

- 0. Dado un fichero XML almacenado en el disco, se va a construir su DOM y recorrer con clases específicas.

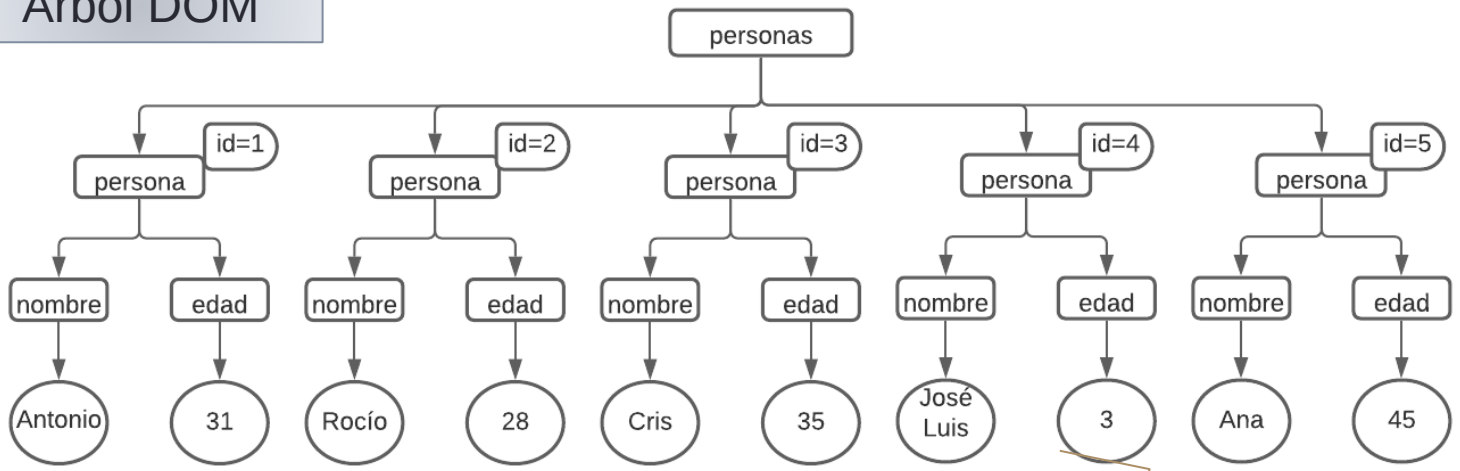
Realizamos las operación necesarias para *parsear* un fichero XML en un árbol DOM

```
//1º Crear una nueva instancia de una fábrica de constructores de documentos a partir de la cual fabricamos un constructor de documentos, que procesará el XML.
DocumentBuilder builderDocument =
                        DocumentBuilderFactory.newInstance().newDocumentBuilder()
;
//2º Parsear/procesar el fichero *.xml y, como consecuencia, se crea un árbol DOM
Document documento = builderDocument.parse(new File("personas_1111.xml"));

//3º Elimina nodos vacíos
documento.getDocumentElement().normalize();
```

Ver a continuación el fichero personas_111.xml y el árbol DOM que resulta al ejecutar las instrucciones anteriores

Árbol DOM




```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<personas>
    <persona id="1">
        <nombre>Antonio</nombre>
        <edad>31</edad>
    </persona>
    <persona id="2">
        <nombre>Rocío</nombre>
        <edad>28</edad>
    </persona>
    <persona id="3">
        <nombre>Cris</nombre>
        <edad>35</edad>
    </persona>
    <persona id="4">
        <nombre>José Luis</nombre>
        <edad>33</edad>
    </persona>
    <persona id="5">
        <nombre>Ana</nombre>
        <edad>45</edad>
    </persona>
</personas>
```

Al *parsear* el fichero XML, se crea la estructura DOM correspondiente

Fichero XML

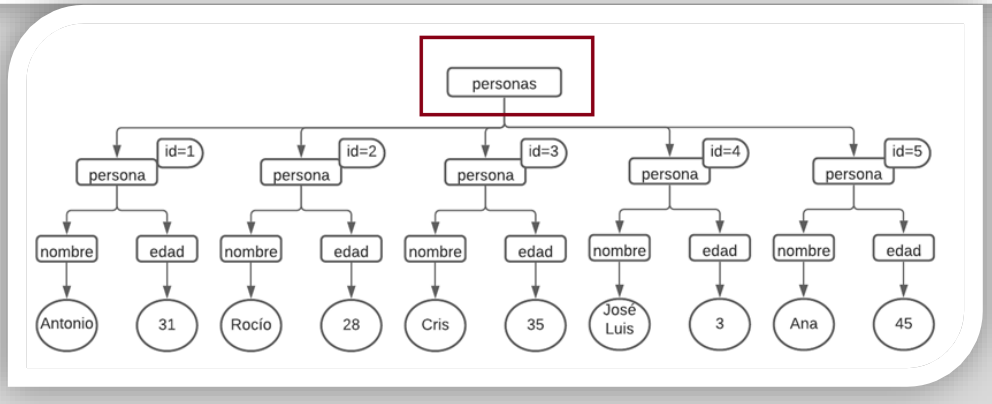
DOM. Recorrer el árbol paso a paso

- 1. Obtener el elemento raíz del DOM.

Obtiene el elemento raíz del objeto Document creado a partir del fichero *.xml

```
System.out.println("Elemento raíz del documento XML: "
        + documento.getDocumentElement().getNodeName());
```

```
Elemento raíz del documento XML: personas
```



DOM. Recorrer el árbol paso a paso

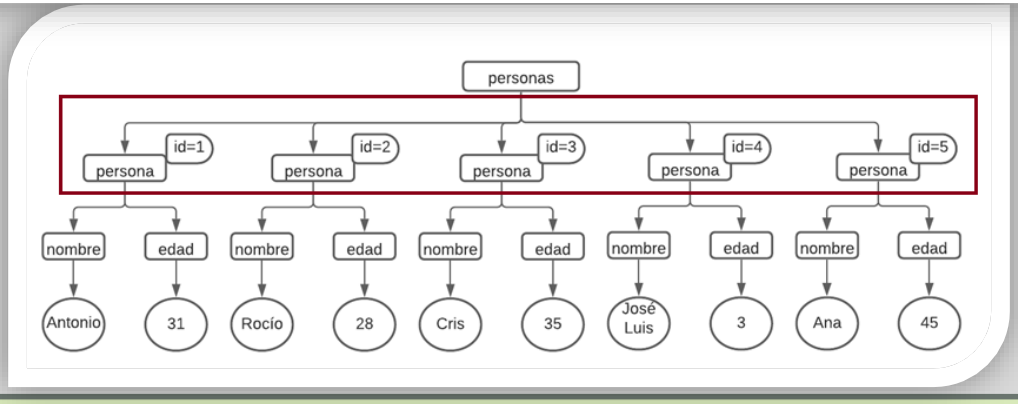
- 2. Mostrar el número de nodos con una etiqueta/*tag* dada.

Realizamos las operación hasta *parsear* un fichero XML en un árbol DOM

```
//Obtenemos unha lista de nodos con nombre "persona" de todo el documento
NodeList listaPersonas = documento.getElementsByTagName("persona");

//Mostramos el nº de nodos llamados "persona" que existen en el DOM (documento)
System.out.println("El nº de elementos de tipo persona en el DOM es " +
                                                    listaPersonas.getLength());
```

```
El n° de elementos de tipo persona en el DOM es 5
```

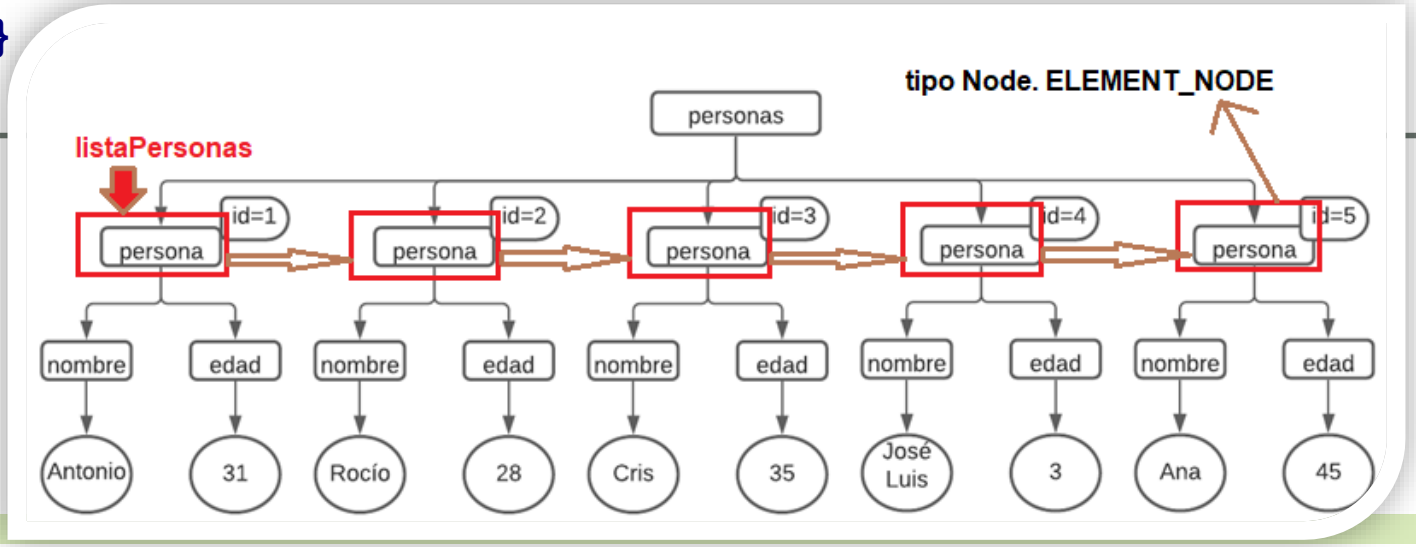


DOM. Recorrer el árbol paso a paso

- 3. Recorremos todos los nodos almacenados en la instancia de **NodeList**, listaPersonas, y mostramos sus nodos hijos. De cada hijo mostramos su texto, y de cada atributo, su contenido.

Recorremos todos los **nodos** de la instancia **Nodelist**, *listaPersonas.* Para elemento de tipo **Node.ELEMENT_NODE** obtenemos los datos

```
for (int i = 0; i < listaPersonas.getLength(); i++) {//Recorrer la lista de elementos
    //Extraer el elemento i de la lista de nodos creada
    Node node_persona = listaPersonas.item(i);
    if (node_persona.getNodeType() == Node.ELEMENT_NODE) {//tipo de nodo
        Element elemento = (Element) node_persona;//Castear a tipo Element
        /*Código para obtener los datos de elemento"*/
        ……..se muestra a continuación……..
    }
}
```

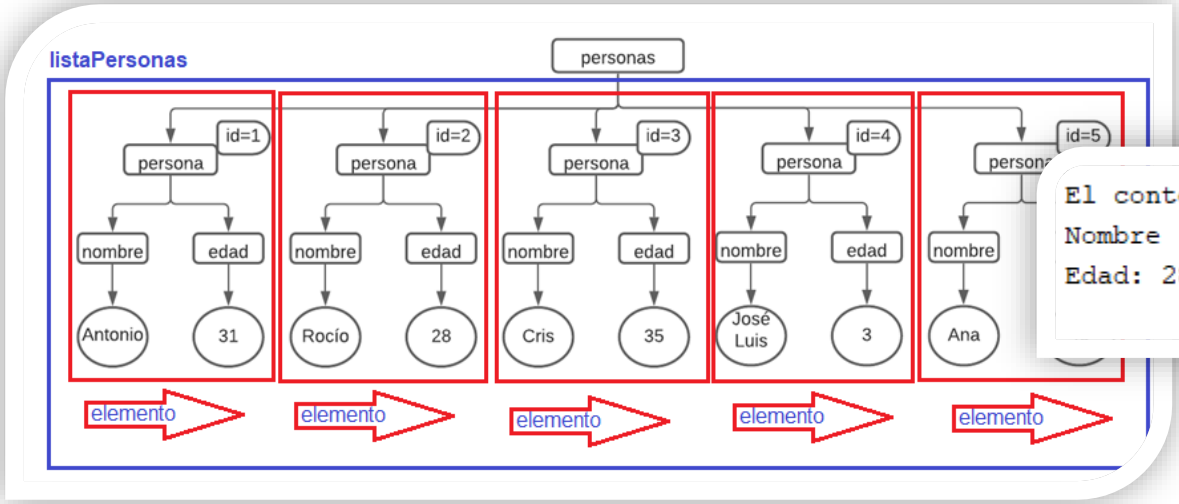


DOM. Recorrer el árbol paso a paso

- 4.Para cada elemento, se muestra por pantalla, su atributo, sus elementos hijos y los valores de cada uno de ellos.
  - 4.1. Conociendo los tags de la información que se muestra por pantalla.

De un nodo de tipo ELEMENT se accede a distinta información de la cual conocemos su nombre/*tag*

```
System.out.println("El contenido del atributo id del elemento es “+
                                                    elemento.getAttribute("id"));
System.out.println("Nombre : " +
                elemento.getElementsByTagName("nombre").item(0).getTextContent());
System.out.println("Edad: "
                    elemento.getElementsByTagName("edad").item(0).getTextContent());
```




```
El contenido del atributo id del elemento es 2
Nombre : Rocio
Edad: 28
```

Información del elemento 2º de la lista listaPersonas

DOM. Recorrer el árbol paso a paso

- 4.Para cada elemento, se muestra por pantalla, su atributo, sus elementos hijos y los valores de cada uno de ellos.
  - 4.2. Sin conocer los tags de la información que se muestra por pantalla. Suponiendo que solo hay 1 nivel de hijos

De un nodo de tipo ELEMENT se accede a su información sin conocer sus *tag*

```
if (elElement.hasAttributes()){
   NamedNodeMap miListaAtributos= elNodo.getAttributes();
   for (int j=0; j<miListaAtributos.getLength(); j++) {
            System.out.println("El atributo es "+
                                                    miListaAtributos.item(j).getNodeName()+
                                                    " y su contenido: "+
                                                    miListaAtributos.item(j).getNodeValue());
   }
}
//Analizamos si tiene hijos y mostramos su contenido
if (elElement.hasChildNodes()) {
   NodeList elNodeList = elNodo.getChildNodes();
   for (int j = 0; j < elNodeList.getLength(); j++) {
            Node elNode = elNodeList.item(j);
            //Muestra el tag del nodo y su valor
            System.out.println(elNode.getNodeName()+": "+elNode.getTextContent() );
/* Hay otra forma de acceder al texto de una etiqueta teniendo en cuenta de que el texto es también un
hijo del elemento: System.out.println(nd.getFirstChild().getNodeValue());*/
}
```

Para el item (0) muestra:

```
El atributo es id y su contenido: 1
nombre: Antonio
edad: 31
```

DOM. Ejercicio 3

- Implementar los métodos para recorrer el árbol DOM del ejercicio.
  - Recorrer el árbol conociendo sus etiquetas/*tags*.
  - Recorrer el árbol sin conocer sus etiquetas pero conociendo los niveles a analizar.

CrearDOM_1111

- Voluntario:
  - Recorrer el árbol con un método recursivo. No se conocen ni las etiquetas/*tags* ni los niveles.

DOM. Práctica

- Crear un proyecto que
  - Cree un fichero serializado de una clase llamada Alumnos que tenga como propiedades curso, nombre y apellidos. El fichero se llamará **alumnos.dat**.
  - A partir del fichero **alumnos.dat** crear un fichero XML que tenga la estructura que se indica a continuación. El fichero se llamará **alumnos.xml**.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<alumnos>
    <alumno curso="C90">
      <nombre>Pilar</nombre>
      <apellidos>Pérez Sousa</apellidos>
    </alumno curso="C80">
    <alumno>
      <nombre>Carmen</nombre>
      <apellidos>Novoa Real</apellidos>
    </alumno>
</alumnos>
```

  - Crear un método que recorra el fichero **alumnos.xml** y lo muestre por pantalla.

DOM-Solución

- Se puede descargar el ejercicio del siguiente enlace:
  - https://github.com/estherff/AD-UD1-ManejoFicheros
  - https://github.com/estherff/AD-UD1-ManejoFicheros/tree/master/src/main/java/gal/teis/ud1/dom
- Para descargar una parte de un proyecto de GitHub en lugar de la totalidad se puede utilizar el siguiente enlace
  - https://minhaskamal.github.io/DownGit/#/home?url=
  -
- Para saber más:
  - https://wiki.cifprodolfoucha.es/index.php?title=Prog_Tratamientos_de_documentos_XML
  - https://www.dsi.uclm.es/personal/miguelfgraciani/mikicurri/Docencia/LenguajesInternet0910/web_LI/Teoria/XML/Programaci%C3%B3n%20en%20castellano_%20Apuntes%20de%20XML.%20Y%20JAVA.htm
  - http://www.jtech.ua.es/j2ee/2002-2003/modulos/xml/apuntes/apuntes3.htm