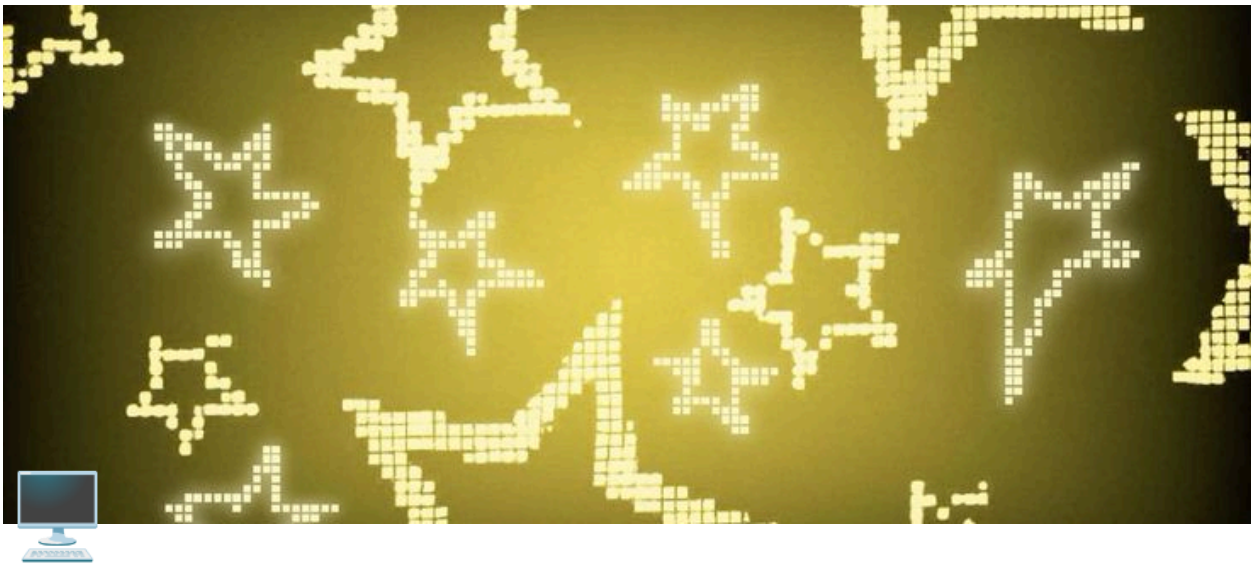


ACTIVIDAD 1.11 - Tratamiento de ficheros XML con DOM en Java

A. Investigación sobre DOM

¿Para qué se utiliza DOM en Java?

DOM (Document Object Model) es una forma de trabajar con documentos XML. Consiste en leer el fichero completo y **cargarlo en memoria como un árbol de nodos.** Una vez cargado, el programa puede recorrer el árbol, buscar etiquetas, leer sus datos, modificarlos, añadir o borrar elementos, y volver a guardar el resultado en un fichero.

Por ejemplo:

```
<instituto tipo="publico">
    <nombre>IES Virgen de la Paloma</nombre>
</instituto>
```

se representa en memoria así:

```
Document
 └── instituto  (atributo tipo="publico")
      └── nombre
           └── "IES Virgen de la Paloma"  (nodo de texto)
```

Cada etiqueta es un nodo, cada atributo también y el texto de dentro es otro nodo. Esa estructura jerárquica es lo que permite moverse por el documento de forma natural.

Ventajas:

- Se puede acceder a cualquier parte del documento en cualquier orden, no solo de principio a fin.
- Permite leer y escribir (modificar) el XML
- Es fácil de entender y de usar.

Inconveniente:

- Al cargar todo el documento en memoria, con ficheros XML muy grandes consume mucha memoria y puede ser lento. Para esos casos se usa **SAX,** que lee el fichero de forma secuencial sin cargarlo entero, aunque solo permite leer.

¿En qué otros ámbitos se usa el enfoque DOM?

DOM es un **estándar del W3C** independiente del lenguaje. Se usa en:

- **Navegadores web y JavaScript:** el ejemplo más conocido. Cuando el navegador carga una página HTML, la convierte en un árbol DOM. JS lo manipula con instrucciones como `document.getElementById("titulo")` para cambiar textos, estilos o añadir elementos sin recargar la pagina.
- **Otros lenguajes:** Python, C#, PHP o C++ tienen sus propias implementaciones de DOM.
- **Otros formatos basados en XML:** SVG (imágenes vectoriales), ficheros de configuración o los diseños de pantalla de Android.

Lectura de ficheros XML con DOM

1. Clases para leer y cargar el fichero

Se necesitan tres clases, que siguen un orden:

1. `DocumentBuilderFactory` (`javax.xml.parsers`): es una fábrica que crea el constructor de documentos.
2. `DocumentBuilder` (`javax.xml.parsers`): es el analizador. Su método `parse(File)` lee el fichero XML y lo convierte en árbol.
3. `Document` (`org.w3c.dom`): es el resultado, el árbol completo en memoria. Representa todo el documento y es el punto de partida para acceder al resto.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document documento = builder.parse(new File("informacion-IES.xml"));
documento.getDocumentElement().normalize();
```

La última línea es una buena práctica: unifica nodos de texto contiguos y deja el árbol en un formato coherente. Además, `getDocumentElement()` devuelve el **elemento raíz** del XML (la etiqueta que contiene a todas las demás).

2. Clases para acceder a los nodos y a su información

Una vez cargado el `Document`, se usan estas interfaces del paquete `org.w3c.dom`:

Clase	Qué representa	Métodos más usados
`Node`	Cualquier nodo del árbol (etiqueta, texto, atributo…). Es la interfaz base de todas las demás	`getNodeName()`, `getNodeType()`, `getTextContent()`, `getChildNodes()`, `getParentNode()`
`NodeList`	Una lista de nodos, por ejemplo el resultado de buscar todas las etiquetas con un nombre	`getLength()`, `item(i)`
`Element`	Un nodo que es una **etiqueta**. Es un tipo de `Node` con más funcionalidades	`getAttribute("nombre")`, `getElementsByTagName("etiqueta")`
`Attr`	Un atributo de una etiqueta	`getName()`, `getValue()`
`Text`	El texto que hay dentro de una etiqueta	`getData()`

¿Cómo se usan juntas?: se busca un conjunto de etiquetas, se recorre la lista y de cada nodo se saca la información.

```
NodeList ciclos = documento.getElementsByTagName("ciclo");   // acceso
a las etiquetas

for (int i = 0; i < ciclos.getLength(); i++) {
    Node nodo = ciclos.item(i);

    if (nodo.getNodeType() == Node.ELEMENT_NODE) {            // compr
obar que es una etiqueta
        Element ciclo = (Element) nodo;                       // conve
rtir a Element
        String codigo = ciclo.getAttribute("codigo");         // leer
un atributo
        String nombre = ciclo.getElementsByTagName("nombre_corto")
                             .item(0).getTextContent();       // leer
el texto de una etiqueta hija
        System.out.println(codigo + " - " + nombre);
```

```
    }
}
```

- Para **acceder a los nodos** (las etiquetas) se usan `NodeList`, `Node` y `Element`.
- Para **acceder a la información que contienen**: `getTextContent()` devuelve el texto de una etiqueta y `getAttribute()` devuelve el valor de un atributo.

Lo de comprobar `getNodeType() == Node.ELEMENT_NODE` es importante porque los espacios y saltos de línea entre etiquetas también cuentan como nodos de texto, y esa comprobación evita tratarlos como si fueran etiquetas.

Escritura de ficheros XML con DOM

1. Clases para escribir (guardar) el fichero

El árbol DOM está solo en memoria, así que para volcarlo a un fichero se usa el paquete `javax.xml.transform`:

- **`TransformerFactory`**: fábrica que crea el transformador.
- **`Transformer`**: realiza la conversión del árbol a texto XML mediante su método `transform(origen, destino)`.
- **`DOMSource`**: envuelve el `Document` y actúa como **origen** de los datos.
- **`StreamResult`**: indica el **destino**, normalmente un fichero (o la consola con `System.out`).

```
TransformerFactory tf = TransformerFactory.newInstance();
Transformer transformer = tf.newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");      // para q
ue quede con sangrías

DOMSource origen = new DOMSource(documento);
StreamResult destino = new StreamResult(new File("salida.xml"));
transformer.transform(origen, destino);
```

2. Clases y métodos para crear contenido

Para crear un documento nuevo (o añadir cosas a uno existente), todo se construye a partir del objeto `Document`, que funciona como una factoría de nodos:

Qué se quiere crear	Método	Clase que devuelve
Una etiqueta	`documento.createElement("nombre")`	`Element`

Qué se quiere crear	Método	Clase que devuelve
El contenido (texto) de una etiqueta	`documento.createTextNode("texto")`	`Text`
Un atributo	`elemento.setAttribute("nombre", "valor")` (o `documento.createAttribute("nombre")`)	`Attr`
Unir un nodo a otro	`padre.appendChild(hijo)`	`Node`

Es importante saber que **crear** un nodo no lo coloca en el documento: hay que **colgarlo** de su padre con `appendChild`, que es lo que construye la jerarquía.

```
Document doc = builder.newDocument();                          // documen
to vacío

Element instituto = doc.createElement("instituto");            // crear e
tiqueta
instituto.setAttribute("tipo", "publico");                     // crear a
tributo
doc.appendChild(instituto);                                    // colgarl
a del documento (raíz)

Element nombre = doc.createElement("nombre");
nombre.appendChild(doc.createTextNode("IES Virgen de la Paloma"));  //
crear texto
instituto.appendChild(nombre);                                 // colgarl
a de instituto
```

Que genera:

```
<instituto tipo="publico">
    <nombre>IES Virgen de la Paloma</nombre>
</instituto>
```