ACTIVIDAD 1.11: TRATAMIENTO FICHEROS XM

1. ¿Para qué se utiliza DOM en Java?

DOM significa Document Object Model, es decir, **Modelo de objetos del documento**. Permite representar un documento XML como un árbol de nodos y trabajar con él mediante objetos: **consultar datos, modificar contenido, añadir elementos o eliminarlos.**

Ejemplo:

```
<instituto tipo="publico">
  <nombre>IES Ejemplo</nombre>
  <modulo id="0485">
    <titulo>Programación</titulo>
    <horas>256</horas>
  </modulo>
</instituto>
```

Los elementos y sus textos son nodos diferentes: <titulo> es un elemento y Programación es el texto que contiene. Esta distinción ayuda a entender por que DOM ofrece tipos como Element y Text.

En Java, el proceso de lectura se puede resumir así:

Fichero XML -> analizador XML -> árbol DOM -> consultas desde Java

2. ¿En qué otros ámbitos se utiliza DOM?

DOM no es exclusivo de Java: **es un modelo independiente del lenguaje para representar y manipular documentos HTML y XML.** Uno de sus usos más habituales está en el desarrollo web, donde JavaScript accede al documento HTML del navegador para cambiar su contenido o responder a eventos.

Ejemplo:

```
<h1 id="titulo">Hola</h1>
document.getElementById("titulo").textContent = "Hola, DAM";
```

En este ejemplo se localiza un elemento del documento y se cambia su texto. La idea es la misma que utilizarás con XML en Java: localizar un nodo y trabajar con él.

3. **¿Qué clases se utilizan para leer y cargar un XML?**

Document, Node, Element y NodelList son interfaces de la API DOM.

Clase o interfaz	Paquete	Función
DocumentBuilderFactory	javax.xml.parsers	Crea y permite configurar el analizador.
DocumentBuilder	javax.xml.parsers	Lee el XML y construye el documento DOM.
Document	org.w3c.dom	Representa el documento completo.
Node	org.w3c.dom	Representa un nodo del árbol.
Element	org.w3c.dom	Representa un elemento XML, como <modulo>.
NodeList	org.w3c.dom	Agrupa nodos para poder recorrerlos.

Estas piezas permiten pasar de un fichero a un documento y después recorrer su contenido.

4. **¿Cómo se accede a las etiquetas y a su información?**

Una vez cargado el XML, ya se puede consultar el árbol.

Obtener el elemento raíz:

```
Element instituto = documento.getDocumentElement();
```

En el XML de ejemplo, el elemento raíz es <instituto>. No hay que confundir el Document, que representa todo el documento, con su elemento raíz.

Buscar elementos por su etiqueta:

```
NodeList modulos = documento.getElementsByTagName("modulo");
```

Esto obtiene los elementos llamados modulo del documento. NodelList permite consultar cuántos hay con getLength() y acceder a cada uno mediante item(indice).

```
for (int i = 0; i <  modulos.getLength(); i++) {
        Element modulo = (Element) modulos.item(i);
        System.out.println(modulo.getTextContent());
}
```

La conversión a **Element** permite tratar cada resultado como un elemento XML. En este caso sabemos que lo es porque hemos buscado por nombre de etiqueta. Al recorrer nodos genéricos con **getChildNodes(),** en cambio, también pueden aparecer nodos de texto y otros tipos.

Leer texto y atributos

Para el XML de ejemplo:

```
Element modulo = (Element) modulos.item(0);

String id = modulo.getAttribute("id");

String titulo = modulo
        .getElementsByTagName("titulo")
        .item(0)
        .getTextContent();

String horas = modulo
        .getElementsByTagName("horas")
```

```
            .item(0)
            .getTextContent();
System.out.println(id + " - " + titulo + " - " + horas);
```

La distinción fundamenta es:

Qué quieres obtener	Método
Los elementos con determinada etiqueta	getElementsByTagName("etiqueta")
El texto contenido en un nodo	getTextContent()
El valor de un atributo	getAttribute("atributo")

Estos métodos permiten acceder a los elementos del árbol y a sus datos sin leer manualmente las etiquetas como cadenas de texto.

- getElementsByTagName() busca entre los descendientes, no solamente entre los hijos directos.
- El ejemplo presupone que existen <titulo> y <horas>; en un programa más robusto habría que comprobar los resultados antes de utilizarlos.

5. ¿Qué clases se utilizan para escribir un XML?

Escribir un XML tiene dos pasos distintos:

- Crear o modificar el árbol DOM en memoria.
- Guardar ese árbol en un fichero.

Clase	Función
TransformerFactory	Crea el transformador.
Transformer	Convierte el árbol DOM en la salida XML.
DOMSource	Representa el documento DOM que se va a guardar.
StreamResult	Indica el destino de la escritura.

6. **¿Cómo se crean etiquetas, texto y atributos?**

El objeto **Document** ofrece métodos para crear los elementos y otros nodos que formarán su árbol.

Crear una etiqueta

```
Element modulo = documento.createElement("modulo");
```

Esto crea el elemento, pero todavía hay que incorporarlo al árbol.

Añadir contenido de texto

Puedes establecerlo directamente:

```
Element titulo = documento.createElement(“titulo”);
titulo.setTextContent(“Programacion”);
```

O crear un nodo de texto explícitamente:

```
Text texto = documento.createTextNode(“Programación”);
Titulo.appendChild(texto);
```

Son dos formas alternativas de construir el contenido de ese elemento; no necesitas ejecutar ambas.

Crear un atributo

La forma más directa es:

```
modulo.setAttribute(“id”,”0485”);
```

También puedes utilizar la interfaz Attr:

```
Attr atributo = documento.createAttribute(“id”);
atributo.setValue(“0485”);
modulo.setAttributeNode(atributo);
```

Finalmente, se enlazan los elementos con appendChild();

```
modulo.appendChild(titulo);
documento.getDocumentElement().appendChild(modulo);
```

El resultado de esa construcción será:

```
<modulo id="0485">
  <titulo>Programación</titulo>
</modulo>
```

Crear un elemento no lo añade por sí solo al documento; tienes que conectarlo con su padre.