Ficheros XML

Ya hemos visto cómo podemos trabajar con información y ficheros utilizando las clases base que nos ofrece Java. Habrás notado que no parece ser mejor forma para todos los escenarios… ¿verdad? En ocasiones, para transferir o registrar información no nos sirve un simple fichero de texto plano, o un fichero binario. Es necesario dotar a la información de una estructura, un orden y una lógica. En estos casos, se recurren a otras tecnologías como el XML.

De nuevo, esto es algo que vas a utilizar en otras asignaturas de 2ndo, como PSP o PMDM.

La definición de libro de **XML** es que son las siglas en inglés de **eXtensible Markup Language**. Es un metalenguaje que permite definir tus propios lenguajes de marcas, desarrollado por el *World Wide Web Consortium (W3C)*.

La idea detrás del XML es que puedas almacenar información de forma legible y estructurada. Similar en apariencia al **HTML** (ambos provienen del SGML), XML da soporte a bases de datos y es útil para cuando varias aplicaciones deben comunicarse entre sí.

Puedes encontrar XML en multitud de aplicaciones informáticas, no solamente en aquellas que hagan uso de internet. Por ejemplo:

- **Servidores**: la configuración de servidores tradicionalmente se hace mediante ficheros XML que se leen al arrancar el servidor.

- **App Android**: los layout de una app son en realidad ficheros XML.

- **RPC**: la comunicación entre cliente y servidor mediante tecnología RPC (Remote Procedure Call) usa XML (o JSON) para transferir información entre ambos.

- **Exportar BBDD**: En algunas pueden hacerse backups mediante XML.

Un ejemplo sencillo de un fichero XML:

```
<student>
    <name>Juan</name>
    <surname>Torres</surname>
    <asignature>ADT</asignature>
</student>
```

Una cualidad del XML es su sencillez, lo que facilita el aprendizaje. Hay mucha documentación disponible en internet sobre el tema, por ejemplo, aquí. Dado que ya se ha trabajado con XML en otras asignaturas de 1ero como Lenguaje de Marcas, en este documento sólo vamos a mencionar las características más importantes de XML.

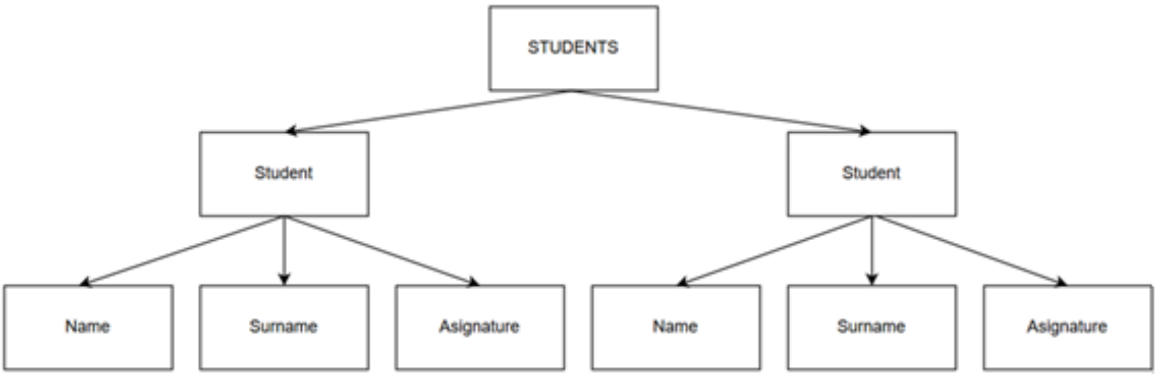
Estructura en árbol

La estructura de un fichero XML es un árbol top-down, esto es, que la raíz se encuentra arriba y sus ramas abajo. A dicho elemento raíz se le llama **root**. Al resto de elementos del XML que dependen de otro elemento se les llama hijos. Por tanto, se usa la nomenclatura típica de padre – hijo para referirse a los diferentes **elementos** que forman el XML.

La información en un XML siempre está **estructurada**. Tiene que tener un orden y una lógica para ser accesible. Por ejemplo, si estamos almacenando alumnos de una clase, la forma más adecuada sería algo parecido a esto:

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student id="1">
        <name>Juan</name>
        <surname>Torres</surname>
        <asignature>ADT</asignature>
    </student>
    <student id="2">
        <name>Ana</name>
        <surname>Sanz</surname>
        <asignature>PMDM</asignature>
    </student>
</students>
```

El root es students. Tiene dos hijos, que son los elementos student. A su vez, cada student tiene varios elementos más con los datos de cada estudiante. Esto podría representarse en forma de árbol:



Atributos y Elementos

La información en un XML se guarda de dos formas diferentes: como atributo de un elemento como parte del elemento mismo. Si bien tienes libertad para diseñar tu propia solución, en general es mejor evitar el abuso de atributos.

```
<student id="1">
    <name>Juan</name>
    <surname>Torres</surname>
    <asignature>ADT</asignature>
</student>


<student>
    <id>1</id>
    <name>Juan</name>
    <surname>Torres</surname>
    <asignature>ADT</asignature>
</student
```

Ambas soluciones son igual de correctas, mientras conozcas el formato del fichero y sepas acceder a la información pertinente.

APIs para XML

Existen diferentes API que nos permiten procesar información en formato XML de forma automática. Nótese que a estas API no les importa en absoluto que ese XML sea un fichero físico o venga a través de una conexión de internet.

SAX (*Simple API for XML Parsing*) y **DOM** (*Document Object Model*) fueron implementadas por Java mediante la API JAXP (*Java Api for XML Processing*). Por un lado, SAX está pensada únicamente para "parsear" (procesar) los documentos XML, y no permite la creación de los mismos. Por otro **DOM**, permite creación de documentos XML, aunque es un estándar independiente del lenguaje. Existe una versión que implementa este estándar en Java: **JAXP**.

JDOM y **XOM** son APIs basadas en DOM, pero diseñadas específicamente para Java y por tanto son más eficientes y más fáciles de usar en ese ámbito.

APIs para XML – DOM – Escritura

DOM (*Document Object Model*) permite analizar y manipular dinámicamente el contenido, el estilo y la estructura de un documento XML. Para usar DOM empleados las clases del paquete **org.w3c.dom** contenido en el propio JSDK, y del paquete **javax.xml.parsers**.

Las clases más importantes son:

- **DocumentBuilderFactory**
- **DocumentBuilder**

Para generar un fichero XML con DOM, lo que se hace es generar primero un objeto Document en memoria. Dicho objeto contendrá la estructura de nodos o elementos del XML que deseamos generar. Dicha generación se hace a mano, es decir, vamos creando los elementos, les damos un valor, y los añadimos después 'colgando' de otro nodo que ya exista en Document. Puedes añadir cualquier elemento donde quieras, por lo tanto, el proceso se asemeja bastante a ir añadiendo 'hijos' a otro elemento.

Los métodos más relevantes son:

- appendChild: cuelga 'algo' a un nodo; o a Document para indicar que es el elemento raíz.

- createAttribute: genera atributos para un nodo.

- setAttributeNode: añade el atributo a un nodo.

La mayor fuente de **errores** a crear un XML es, simple y llanamente, que nos liamos o nos olvidamos de 'colgar' un elemento en su sito correspondiente. Hacer appendChild en el sitio equivocado arruina el trabajo completo.

Un ejemplo de DOM que genera un XML de estudiantes podría ser:

```
// Get the Factory
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

// Get the DocumentBuilder
DocumentBuilder builder = factory.newDocumentBuilder();

// Create the new document (in memory)
Document document = builder.newDocument();

// Let's add the different elements to the document
// First, the root element
Element root = document.createElement("students");
document.appendChild(root); // Add root to the document

// Now, the child (student)
Element student = document.createElement("student");
Attr attrID = document.createAttribute("id");
attrID.setValue("1");
student.setAttributeNode(attrID);

// Now then, the children of student
// Note we add the
Element name = document.createElement("name");
name.appendChild (document.createTextNode("Juan"));
student.appendChild(name); // Add nodes to the student

Element surname = document.createElement("surname");
surname.appendChild (document.createTextNode("Torres"));
student.appendChild(surname);

Element asignature = document.createElement("asignature");
asignature.appendChild (document.createTextNode("ADT"));
student.appendChild(asignature);

root.appendChild(student); // Add student to the root

// Transformer to write
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();

// DOM
DOMSource source = new DOMSource (document);
StreamResult result = new StreamResult (new File ("example.xml"));
transformer.transform(source, result);
```

El producto de este código es el XML mencionado, aunque con un único estudiante. Para añadir más elementos, tendríamos que ir añadiéndolos de forma manual. Obviamente, para ficheros más complejos, se hace necesario reorganizar el código en diferentes métodos etc.

APIs para XML – DOM – Lectura

De forma similar, la lectura de un XML se realiza 'a mano' recorriendo el árbol de elementos. En este caso, se disponen de métodos para recuperar cada elemento y su información correspondiente. Por ejemplo:

- getFirstChild: obtiene el primer hijo de un elemento.
- getLastChild: obtiene el ultimo hijo de un elemento.
- getNextSibling: obtiene el siguiente hermano de un elemento.
- getPreviousSibling: obtiene el anterior hermano de un elemento.
- getAttribute: retorna un atributo de un elemento.
- getElementsByTagName: devuelve elementos por su nombre.

Por tanto, el proceso de lectura de un XML consiste en recorrer poco a poco el árbol de elementos.

Un ejemplo de DOM que accede a un XML de estudiantes podría ser:

```
// Get the Factory
factory = DocumentBuilderFactory.newInstance();

// Get the DocumentBuilder
builder = factory.newDocumentBuilder();

// Parse the XML file into a Document
Document document = builder.parse(new File("example.xml"));
document.getDocumentElement().normalize();

// Root
Element root = document.getDocumentElement();
System.out.println("Root: " + root.getNodeName());

// The root's children (student)
NodeList rootNodeList = document.getElementsByTagName("student");

// Let's iterate and get all students
for (int i = 0; i < rootNodeList.getLength(); i++) {
    Node node = rootNodeList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;
        System.out.println("student: " + i);
        System.out.println("id: " + element.getAttribute("id"));

        NodeList childNodes = node.getChildNodes();
        for (int j = 0; j < childNodes.getLength(); j++) {
            Node childNode = childNodes.item(j);
            if (childNode.getNodeType() == Node.ELEMENT_NODE) {
                Element childElement = (Element) childNode;
                System.out.println("childElement: " + childElement.getTextContent());
            }
        }
    }
}
```

Es muy probable que a estas alturas ya te hayas dado cuenta del principal problema de trabajar así con los ficheros XML: tienes que **conocer** cómo es ese XML para poder trabajar con él de forma correcta. Esto complica mucho la programación cuando hay atributos que no son obligatorios (por ejemplo), no se señaliza adecuadamente un conjunto de elementos, o incluso el debugeo de errores en el código.

APIs para XML – DOM – Modificación

Esta es una pregunta con trampa. Sí es posible modificar un fichero XML, pero para ello generamos uno nuevo con el contenido del original en memoria, cambiamos lo que nos parezca, y luego sobrescribimos el fichero original. Así de sencillo.

```
// Get the Factory
factory = DocumentBuilderFactory.newInstance();

// Get the DocumentBuilder
builder = factory.newDocumentBuilder();

// Parse the XML file into a Document
Document document = builder.parse(new File("example.xml"));
document.getDocumentElement().normalize();

// Root
Element root = document.getDocumentElement();
System.out.println("Root: " + root.getNodeName());

// The root's children (student)
NodeList rootNodeList = document.getElementsByTagName("student");

// Let's iterate and get the student with id = 1
for (int i = 0; i < rootNodeList.getLength(); i++) {
    Node node = rootNodeList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;
        String id = element.getAttribute("id");

        // Set id to 999
        if (id.equalsIgnoreCase("1")) {
            Attr attrID = document.createAttribute("id");
            attrID.setValue("999");
            element.setAttributeNode(attrID);
            break;
        }
    }
}

// Transformer to write
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();

// DOM
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(new File("example.xml"));
transformer.transform(source, result);
```

Práctica - 4

Crea un programa que permita gestionar tu colección de videojuegos. Se va a almacenar la información de la siguiente manera:

```
Videojuego: DOOM3
    Distribuidor: Activision
    Fecha: 2007
    Tipo: FPS
    Jugadores: 1
```

El fichero XML va a funcionar como una base de datos, de forma que tiene que soportar las operaciones habituales de alta, borrado y modificación, a demás de permitirnos buscar un videojuego.

- Mostrar todos: Se muestra el contenido del fichero.
- Buscar: Busca un videojuego y lo muestra.
- Alta: Se añade un nuevo videojuego con todos sus campos al fichero.
- Baja: Se elimina un videojuego por su nombre.
- Modificación: Se cambia el nombre de un videojuego por otro.