XML (eXtensible Markup Language) y DOM (Document Object Model)

XML es un lenguaje de marcas muy estandarizado utilizado para almacenar e intercambiar datos de forma legible, caracterizado por estructurarse a través de etiquetas.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<versiones>
   <version numero="1.5">
      <nombre>Cupcake</nombre>
      <api>3</api>
   </version>
   <version numero="1.6">
      <nombre>Donut</nombre>
      <api>4</api>
   </version>
   <version numero="2.0">
      <nombre>Éclair</nombre>
      <api>5</api>
   </version>
   <version numero="2.2">
      <nombre>Froyo</nombre>
      <api>8</api>
   </version>
   <version numero="2.3">
      <nombre>Gingerbread</nombre>
      <api>9</api>
   </version>
   <version numero="3">
      <nombre>Honeycomb</nombre>
      <api>11</api>
   </version>
   <version numero="4.0">
      <nombre>Ice Cream Sandwich</nombre>
      <api>14</api>
   </version>
   <version numero="4.1">
      <nombre>Jelly Bean</nombre>
      <api>16</api>
   </version>
   <version numero="4.4">
      <nombre>KitKat</nombre>
      <api>19</api>
   </version>
   <version numero="5.0">
      <nombre>Lollipop</nombre>
      <api>21</api>
   </version>
   <version numero="6.0">
      <nombre>Marshmallow</nombre>
      <api>23</api>
   </version>
</versiones>
```

Lectura de un archivo XML

A la hora de leer un archivo XML a través de **DOM** debemos instanciar una serie de clases antes de poder tratar el fichero. Primero utilizaremos la conocida clase File para cargar nuestro fichero.

```
File file = new File("versiones.xml");
```

Posteriormente y ya dentro de un try/catch (para tratar los excepciones) parsearemos el fichero con estas clases: **DocumentBuilderFactory**, **DocumentBuilder** y **Document**.

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(file);
```

Una vez hecho todo esto ya podremos leer el archivo versiones.xml además de usar otros métodos como los ejemplos de aquí abajo.

Método	Descripción
getDocumentElement()	Accede al nodo raíz del documento
normalize()	Elimina nodos vacíos y combina adyacentes en caso de que los hubiera

```
// Estos métodos podemos usarlos combinados para normalizar el archivo XML
doc.getDocumentElement().normalize();
```

Siguiendo dentro del try/catch podemos utilizar la clase **NodeList** para almacenar el elemento que le indicaremos como parámetro.

```
// Almacenamos los nodos para luego mostrar la
// Cantidad de ellos con el método getLength()
NodeList nList = doc.getElementsByTagName("version");
System.out.println("Número de versiones: " + nList.getLength());
```

Una vez tenemos almacenados los datos del nodo "**version**" podemos leer su contenido teniendo en cuenta que este código depende de que conozcamos la estructura y etiquetas utilizadas.

```
for(int temp = 0; temp < nList.getLength(); temp++) {
Node nNode = nList.item(temp);

if(nNode.getNodeType() == Node.ELEMENT_NODE) {
        Element eElement = (Element) nNode;

        System.out.println("\nVersion id: " + eElement.getAttribute("numero"));
        System.out.println("Nombre: "
                +
eElement.getElementsByTagName("nombre").item(0).getTextContent());
        System.out.println("api: "
                + eElement.getElementsByTagName("api").item(0).getTextContent());
          }
         }
```

Nos mostraría por consola:

```
Número de versiones: 11

Version id: 1.5
Nombre: Cupcake
api: 3

Version id: 1.6
Nombre: Donut
api: 4

Version id: 2.0
Nombre: Éclair
api: 5

Version id: 2.2
Nombre: Froyo
api: 8

Version id: 2.3
Nombre: Gingerbread
api: 9

Version id: 3
Nombre: Honeycomb
api: 11

Version id: 4.0
Nombre: Ice Cream Sandwich
api: 14

Version id: 4.1
Nombre: Jelly Bean
api: 16

Version id: 4.4
Nombre: KitKat
api: 19

Version id: 5.0
Nombre: Lollipop
api: 21

Version id: 6.0
Nombre: Marshmallow
api: 23
```

Escritura archivo XML

Si quisiéramos escribir un archivo XML siguiendo la misma estructura de las versiones, deberíamos instanciar las clases **DocumentBuilderFactory**, **DocumentBuilder** y **Document**, definir toda la estructura del archivo (siempre dentro de un bloque try/catch) y por último instanciar las clases **TransformerFactory**, **Transformer**, **DOMSource** y **StreamResult** para crear el archivo.

```
try {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.newDocument();

        // Definimos el elemento raíz del documento
        Element eRaiz = doc.createElement("Versiones");
        doc.appendChild(eRaiz);

        // Definimos el nodo que contendrá los elementos
        Element eVersion = doc.createElement("version");
        eRaiz.appendChild(eVersion);

        // Atributo para el nodo version
        Attr attr = doc.createAttribute("numero");
        attr.setValue("10");
        eVersion.setAttributeNode(attr);

        // Definimos cada uno de los elementos y le asignamos un valor
        Element eNombre = doc.createElement("nombre");
        eNombre.appendChild(doc.createTextNode("Andruito"));
        eVersion.appendChild(eNombre);

        Element eApi = doc.createElement("api");
        eApi.appendChild(doc.createTextNode("25"));
        eVersion.appendChild(eApi);

        // Clases necesarias finalizar la creación del archivo XML
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        //Lo pongo bonito insertando saltos de línea al final de cada línea
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");

        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File("ejercicio3.xml"));

        transformer.transform(source, result);

      } catch(Exception e) {
        e.printStackTrace();
      }
```