Trabajo con ficheros XML: analizadores sintácticos (parser) y vinculación (binding).

U**na aplicación que consume información XML debe:**

- Leer un fichero de texto codificado según dicho estándar.
- Cargar la información en memoria y, desde allí...
- Procesar esos datos para obtener unos resultados (que posiblemente también almacenará de forma persistente en otro fichero XML).

El proceso anterior se enmarca dentro de lo que se conoce en informática como "**parsing" o análisis léxico-sintáctico.**

Trabajo con ficheros XML

Cuando se quieren almacenar datos que deban ser leídos por aplicaciones ejecutadas en múltiples plataformas será necesario recurrir a formatos más estandarizados, como **los lenguajes de marcas.**

Trabajo con ficheros XML

Los documentos **XML** consiguen estructurar la información intercalando una serie de marcas denominadas etiquetas. En **XML** , las marcas o etiquetas tienen cierta similitud con un contenedor de información. Así, una etiqueta puede contener otras etiquetas o información textual. De este modo, conseguiremos subdividir la información estructurando de forma que pueda ser fácilmente interpretada.

Trabajo con ficheros XML

```
▼<pizzas>
  ▼<pizza nombre="Barbacoa" precio="8">
      <ingrediente nombre="Salsa Barbacoa"/>
      <ingrediente nombre="Mozzarella"/>
      <ingrediente nombre="Pollo"/>
      <ingrediente nombre="Bacon"/>
      <ingrediente nombre="Ternera"/>
    </pizza>
  ▼<pizza nombre="Margarita" precio="6">
      <ingrediente nombre="Tomate"/>
      <ingrediente nombre="Jamón"/>
      <ingrediente nombre="Queso"/>
    </pizza>
</pizzas>
```

Trabajo con ficheros XML

- La información es **textual**
- Cualquier dato, ya sea numérica o booleana, habrá que transcribirla en modo texto, de modo que cualquiera que sea el sistema de representación de datos será posible leer e interpretar correctamente la información contenida en un archivo XML .
- XML consigue estructurar cualquier tipo de información jerárquica.

Trabajo con ficheros XML

Hay **cierta similitud** con la forma como la información se almacena en los objetos de una aplicación y la forma como se almacenaría en un documento XML .

- **Aplicaciones orientadas a objeto**
  Estructura, agrupa y jerarquiza en clases.
- **Documentos XML**
  Estructura, organiza y jerarquiza en etiquetas contenidas unas dentro de otras y atributos de las etiquetas.

Trabajo con ficheros XML

Existen herramientas y estándares de programación para leer documentos XML.

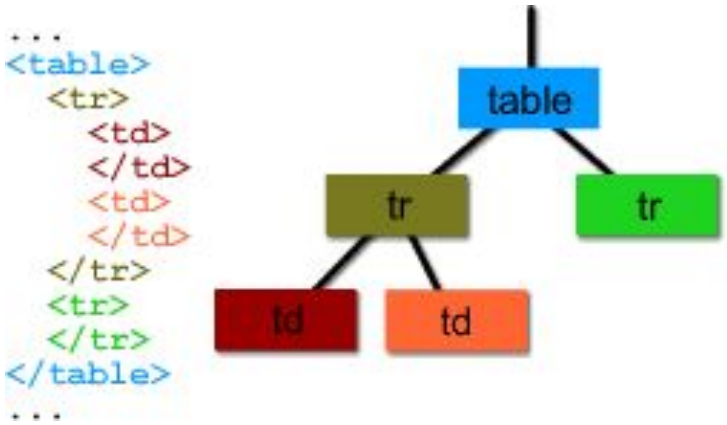
1. Las herramientas o programas que leen el lenguaje XML y **comprueban si el documento es válido sintácticamente**, se denominan **analizadores o "parsers".**
2. Un **parser XML** es un módulo, biblioteca o programa que se ocupa de **analizar, clasificar y transformar un archivo de XML en una representación interna,** extrayendo la información contenida en cada una de las etiquetas y relacionándola de acuerdo con su posición en la jerarquía.

Trabajo con ficheros XML

Existen un gran número de parsers o analizadores sintácticos disponibles:

SAX: analizador secuencial

DOM:analizador jerárquico



Trabajo con ficheros XML

Analizadores secuenciales

Permiten extraer el contenido a medida que se van descubriendo las etiquetas de apertura y cierre

Son analizadores muy rápidos

PROBLEMA

cada vez que se necesita acceder a una parte del contenido es necesario releer todo el documento de arriba a abajo.

Trabajo con ficheros XML

En Java hay dos analizadores secuenciales:

- **SAX,** que es el acrónimo de Simple API for XML . Es una analizador muy usado en varias bibliotecas de tratamiento de datos XML
- **STAX**, Streaming API for XML, posterior a SAX y que lo ha superado.

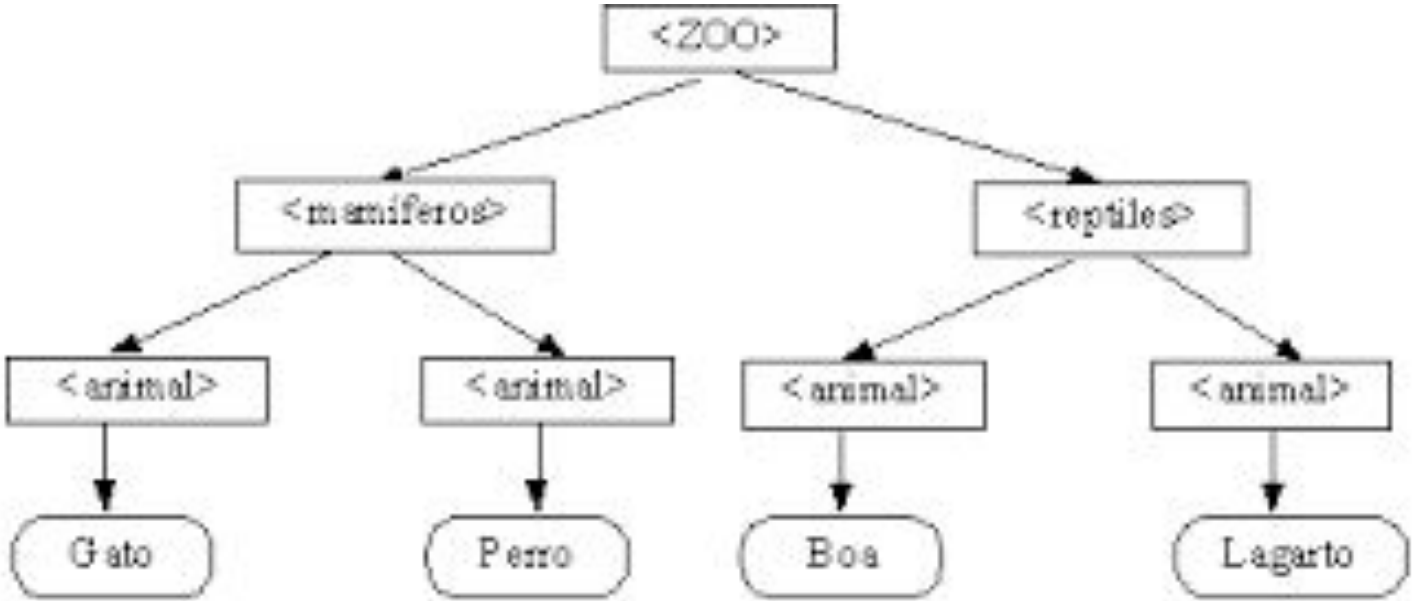
Trabajo con ficheros XML

Analizadores jerárquicos

Los analizadores jerárquicos guardan todos los datos del XML en memoria dentro de una estructura jerárquica.

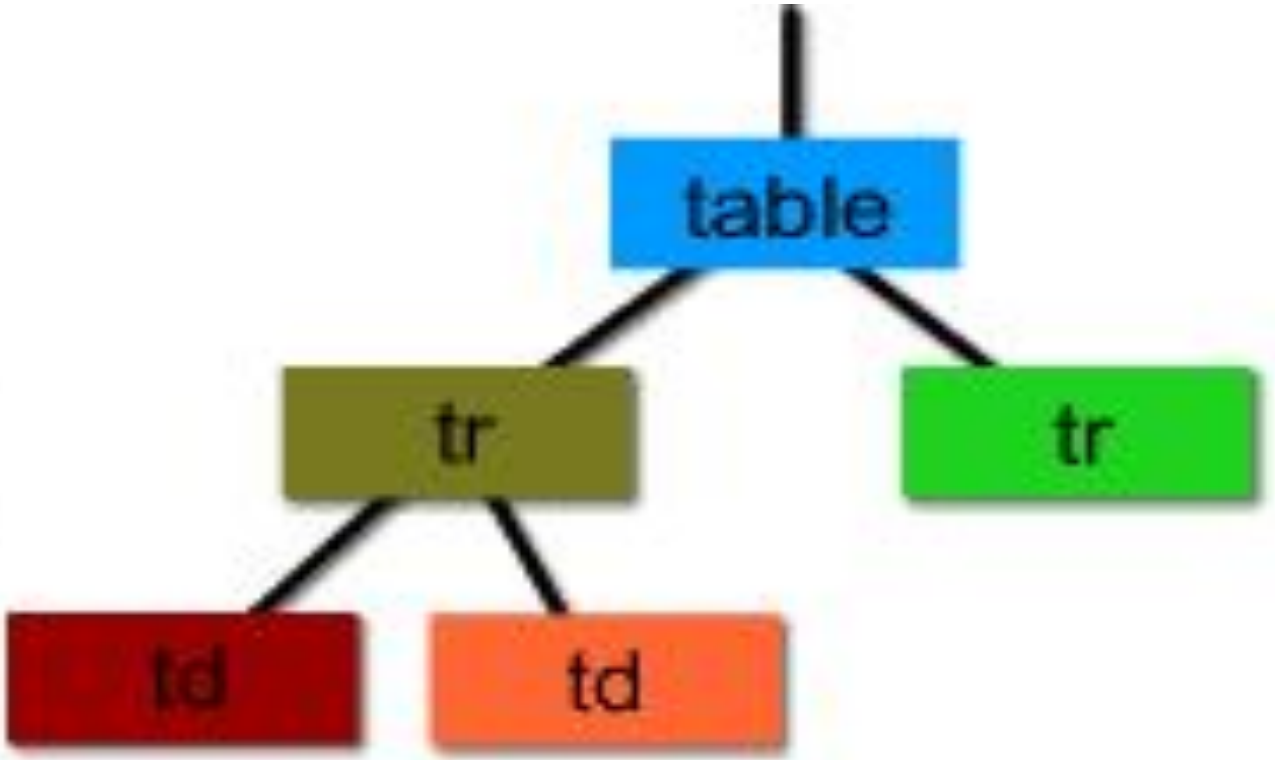
Son ideales para aplicaciones que requieran una consulta continua de los datos

DOM


<ZOO>
<mamiferos>
<reptiles>
<animal>
<animal>
<animal>
<animal>
Gato
Perro
Boa
Lagarto

DOM

```
...
<table>
    <tr>
        <td>
        </td>
        <td>
        </td>
    </tr>
    <tr>
    </tr>
</table>
...
```



Trabajo con ficheros XML

DOM

El formato de la estructura donde se almacena la información en la memoria RAM ha sido especificado por el organismo internacional W3C (World Wide Web Consortium) y se suele conocer como **(Document Object Model)**

Trabajo con ficheros XML

DOM

Define una interfaz que permite a los programas acceder y actualizar el estilo, la estructura y el contenido de los documentos XML.

Los analizadores XML compatibles con DOM implementan esta interfaz.

Trabajo con ficheros XML

DOM

El estándar W3C define la especificación de la clase **DocumentBuilder** con el propósito de poder instanciar estructuras **DOM a partir de un XML**

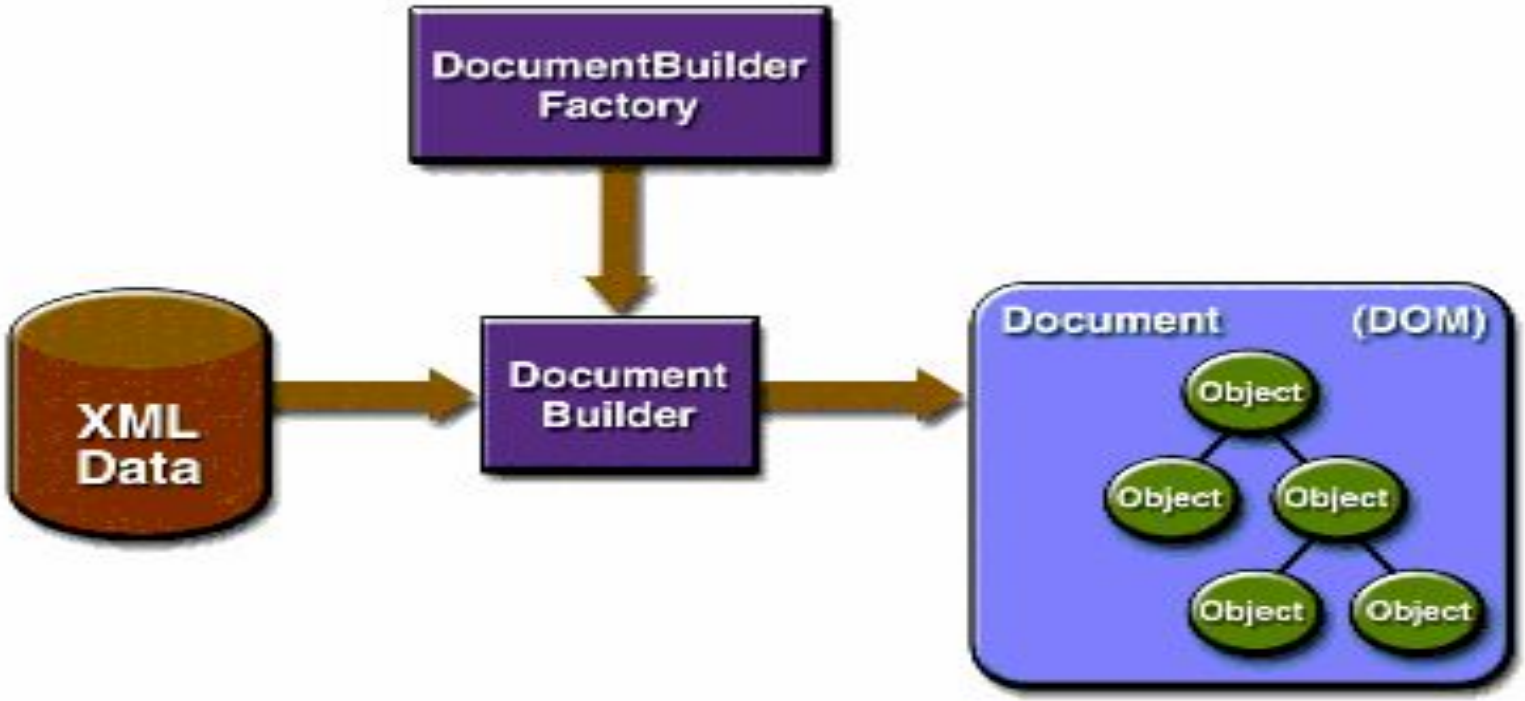
Trabajo con ficheros XML

DOM

DocumentBuilder es una clase abstracta, y para que se pueda adaptar a las diferentes plataformas, puede necesitar fuentes de datos o requerimientos diversos. Las clases abstractas no se pueden instanciar de forma directa.

Trabajo con ficheros XML

DOM



Trabajo con ficheros XML

DOM

Las instrucciones necesarias para **leer un archivo XML** y crear un objeto Documento serían las siguientes:

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
newInstance() ;
DocumentBuilder dBuilder = dbFactory. newDocumentBuilder ;
Document doc = dBuilder. parse ( new File ( "fitxer.xml" ) ) ;
```

Trabajo con ficheros XML

DOM

La escritura de la información contenida en el DOM se puede secuenciar en forma de texto utilizando otra utilidad de Java llamada **Transformer**

- **Es también una clase abstracta**
- Permite pasar la información contenida en un objeto *Documento* a un archivo de texto en formato XML.
- También realiza lla operación inversa, pero el mismo **DocumentBuilder** ya se encarga de ello.

Trabajo con ficheros XML

DOM

La clase Transformer puede trabajar con multitud de contenedores de información porque en realidad trabaja con un par de tipos adaptadores (clases que hacen compatibles jerarquías diferentes) que se llaman **Source y Result**. Las clases que implementen estas interfaces s**e encargarán de hacer compatible un tipo de contenedor específico al requerimiento de la clase Transformer.**

Trabajo con ficheros XML

DOM

Existen las clases **DOMSource, SAXSource o StreamSource** como adaptadores del contenedor de la fuente de información.

Trabajo con ficheros XML

DOM

El código básico para realizar una transformación de DOM archivo de texto XML sería el siguiente:

```
// Creación de una instancia Transformer
Transformer trans = TransformerFactory. newInstance . newTransformer ;
// Creación de los adaptadores Source y Results a partir de un Documento
// y un File.
StreamResult result =  new StreamResult ( file ) ;
DOMSource source =  new DOMSource ( doc ) ;
trans.transform ( source, result ) ;
```

Trabajo con ficheros XML

DOM

```
// Asocio el source con el Document
   Source source = new DOMSource(documento);

    // Creo el Result, indicado que fichero se va a crear
    Result result = new StreamResult(new File("libros.xml"));
  // Creo un transformer, se crea el fichero XML
Transformer transformer TransformerFactory.newInstance().newTransformer();
 transformer.transform(source, result);
```

Trabajo con ficheros XML

- La estructura DOM toma la forma de un árbol, donde cada parte del XML se encontrará representada en forma de nodo.
- En función de la posición en el documento XML, hablaremos de diferentes tipos de nodos.
- El nodo principal que representa todo el XML entero denomina **documento**.
- Las diversas etiquetas, incluida la etiqueta raíz, se conocen como nodos elementos.
- El contenido textual de una etiqueta se instancia como nodo de tipo **TextElement** y los atributos como nodos de tipo **Atribute.**
- Cada nodo específico dispone de métodos para acceder a sus datos concretos (nombre, valor, nodos hijos, nodo padre, etc.).

Trabajo con ficheros XML

La interfaz Document Contempla un conjunto de métodos para seleccionar diferentes partes del árbol a partir del nombre de la etiqueta o de un atributo identificador. Las partes del árbol se devuelven como objetos **Element,** los cuales representan un nodo y todos sus hijos. De este modo, podremos ir explorando partes del árbol sin necesidad de tener que pasar por todos los nodos.

Trabajo con ficheros XML

DOM define varias interfaces, las más comunes son las siguientes:

- **Node** − Representa a cualquier nodo del documento.
- **Element** − expone propiedades y métodos para manipular los elementos del documento y sus atributos.
- **Attr** − Representa un atributo de un elemento.
- **Text** − Contenido de un elemento o atributo.
- **Document** − Representa al documento XML completo. Generalmente nos referiremos a él como **árbol DOM**. Proporciona información del documento. Permite crear nuevos nodos en el documento.
- **NodeList**. Colección de nodos a los que se puede acceder por medio de un índice.

Métodos DOM más usuales

Document.getDocumentElement() − Retorna el elemento raíz del documento.
Node.getFirstChild() − Retorna el primer nodo hijo del nodo.
Node.getLastChild() − Retorna el último nodo hijo del nodo.
Node.getNextSibling() − Retorna el siguiente hermano de un nodo.
Node.getPreviousSibling() − Retorna el hermano anterior de un nodo.
Node.getAttribute(attrName) − Retorna el atributo del nodo con el nombre que se pasa como atributo.

Trabajo con ficheros XML

```
public static String getValorEtiqueta ( String etiqueta, Element
elemento )
 {
    Nodo nValue = elemento. getElementsByTagName (etiqueta ) . item ( 0 ) ;
     return nValue. getChildNodes . item ( 0 ) . getNodeValue ;
 }
```

Trabajo con ficheros XML

```
public static Elemento getElementEtiqueta ( String etiqueta,
Element elemento )
{
    return ( Element ) elemento. getElementsByTagName ( etiqueta ).item ( 0 ) ;
}
```

Trabajo con ficheros XML

- Los objetos **Elemento** disponen de métodos para añadir nuevos hijos (**appendChild**) o asignar el valor a un atributo (**setAtributte**).
- También permiten la consulta del valor de los atributos (**getAttributte**) o la navegación por los nodos del árbol (**getParentNode**, para obtener el padre; **getFirstChild/getLastChild,** para obtener el primer / último hijo, o **getNextSibling** para navegar de hermano en hermano).

EJEMPLO

Fichero **clase.xml**

```
<? xml version = "1.0"?>
<clase>
   <alumno numero = "393">
      <nombre> Luis </ nombre>
      <apellido> Luna </ apellido>
      <apodo> Na </ apodo>
      <marcas> 85 </ marcas>
   </ alumno>
.....
<alumno numero = "394">
     ……
     ……
</ alumno>

<alumno numero = "395">
     ……
     ……
</ alumno>

<alumno numero = "395">
     ……
     ……
</ alumno>
```

EJEMPLO

```
public static void main(String[] args) {
    try {
        File inputFile = new File("clase.xml");
        DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(inputFile);
        doc.getDocumentElement().normalize();
        System.out.println("Root element :" +
        doc.getDocumentElement().getNodeName());
        NodeList nList = doc.getElementsByTagName("alumno");
        System.out.println("----------------------------");
```

EJEMPLO

```
for (int temp = 0; temp < nList.getLength(); temp++) {
        Node nNode = nList.item(temp);
        System.out.println("\nCurrent Element :" + nNode.getNodeName());
        if (nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            System.out.println("numero de alumno : "+ eElement.getAttribute("numero"));
            System.out.println("nombre : "+
eElement.getElementsByTagName("nombre").item(0).getTextContent());
            System.out.println("apellido :"+
eElement.getElementsByTagName("apellido").item(0).getTextContent());
System.out.println("apodo : "+
eElement.getElementsByTagName("apodo").item(0).getTextContent());
System.out.println("marcas :
"+eElement.getElementsByTagName("marcas").item(0).getTextContent());
 }
  }
```

EJEMPLO

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