

## Tema 3: XML

*eXtensible Markup Language*

**3º Parcial - LMSGI**

**1º de Ciclo Superior de DAM**

## Tema 3: XML

**Objetivos:** *conocer características y aplicaciones más interesantes de XML y saber crear documentos XML correctos.*

- ✓ **Repaso** del tema inicial sobre **lenguajes de marcas**: representación de la información, estándares de codificación, ventajas de archivos de texto, lenguajes *orientados a la descripción, clasificación* de lenguajes según **ámbito** de aplicación.
- ✓ **Historia**
- ✓ **Cuestiones básicas** sobre XML
- ✓ **Objetivos**
- ✓ **Características**
- ✓ **Aplicaciones XML**
- Creación de un **documento XML correcto**:
  - 1) Bien **formado**. **Reglas** básicas.
  - 2) **Válido**
- ✓ **¿Atributos o elementos?**
- ✓ **Jerarquía**
- ✓ **Validación por DTD**

## Repaso de XML

- **XML (eXtensible Markup Language):**
  - ✓ Desarrollado por el **World Wide Web Consortium**, es un **subconjunto** de **SGML**, ideado para **mejorarlo**, simplificándolo y adaptándolo a los documentos de Internet.
  - ✓ Es más **entendible**, logrando equilibrio entre simplicidad y flexibilidad, además de ser uno de los lenguajes de etiquetas más importantes.
  - ✓ Define lenguajes con sintaxis **más estricta**, indicando lo que esá permitido y lo que no.
  - ✓ Se trata de un **metalinguaje** que permite crear **etiquetas** adaptadas a las necesidades, **definiendo cómo pueden ser y qué** se puede hacer con ellas.
  - ✓ Todo **documento** debe cumplir dos **condiciones**: ser **válido** y estar **bien formado**.
  - ✓ Permite asignar **atributos** a las etiquetas.
  - ✓ La **estructura** y el **diseño** son **independientes**.
  - ✓ **Soluciona problemas** surgidos de la internacionalización y de la imposibilidad de validar (problemas de HTML).
  - ✓ Es uno de los formatos de documentos más populares para **exportación e importación** de datos.

## Repaso de XML

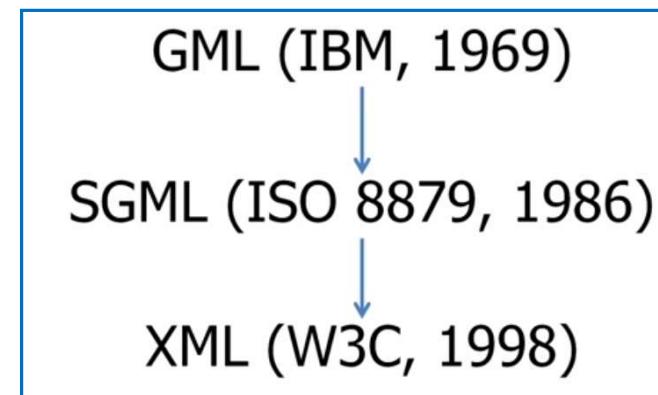
- **Tipos de lenguajes de marcas:**
  - a) **Orientados a la presentación:** sólo se muestra la presentación.
  - b) **Orientados a la descripción -descriptivos o semánticos-:** describen las diferentes partes en las que se *estructura el documento* sin especificar cómo deben representarse.
    - Ej: **XML, JSON** en los que indica una semántica de contenido que lo hace ideal para **almacenar** datos, como el nombre de persona, su Nif, ....
  - c) **Orientados a los procedimientos:** incluyen instrucciones de cómo hay que procesar el texto.
- **Ej de documento XML:**

```
1 <persona>
2   <nombres>Elsa</nombres>
3   <apellidos>Zambrano</apellidos>
4   <fecha-de-nacimiento>
5     <día>18</día>
6     <mes>6</mes>
7     <año>1996</año>
8   </fecha-de-nacimiento>
9   <ciudad>Pamplona</ciudad>
10</persona>
```

## Historia de XML

- **Historia de XML**

- Proviene de **GML** –*General Markup Language*-, lenguaje desarrollado por **IBM** en 1969 para solucionar su problema de *almacenar grandes cantidades de información*
- La **ISO** –*International Organization for Standardization*- en 1986 lo **normalizó** creando **SGML** –*Standard General Markup Language*–
  - **Estándar** mundial en documentos de texto con etiquetas de marcado: sistemas de publicación basada en contenido textual
  - Define las reglas fundamentales de los lenguajes de etiquetas actuales
- La **W3C** en 1998 creó **XML** como un subconjunto más fácil de utilizar



## Historia de XML

- Ej de GML:

- Las etiquetas se definen anteponiendo :
- Ya hay elementos de encabezados, párrafos, listas

```
:h1.Chapter 1: Introduction
:p.GML supported hierarchical containers, such as
:ol
:li.Ordered lists (like this one),
:li.Unordered lists, and
:li.Definition lists
:eol.
as well as simple structures.
:p.Markup minimization (later generalized and formalized
in SGML), allowed the end-tags to be omitted for the "h1"
and "p" elements.
```

## Historia de XML

- Ej de SGML:

- Símbolos < y > para las etiquetas
- Etiquetas utilizadas en otros lenguajes

```
<!doctype Linuxdoc system>
<!-- Here's an SGML example file. Format it and print out the source, and
use it as a model for your own SGML files. As you can see this is a
comment.
--&gt;
&lt;article&gt;
<!-- Title information --&gt;
&lt;title&gt;Quick SGML Example
&lt;author&gt;Matt Welsh, &lt;tt/mdw@cs.cornell.edu/
&lt;date&gt;v1.0, 28 March 1994
&lt;abstract&gt;
This document is a brief example using the Linuxdoc-SGML DTD.
&lt;/abstract&gt;
&lt;!-- Table of contents --&gt;
&lt;toc&gt;
&lt;!-- Begin the document --&gt;
&lt;sect&gt;Introduction
&lt;p&gt;
This is an SGML example file using the Linuxdoc-SGML DTD. You can format it
using the command
&lt;tscreen&gt;&lt;verb&gt;
% sgml2txt example.sgml
&lt;/verb&gt;&lt;/tscreen&gt;
this will produce plain ASCII. You can also produce LaTeX, and HTML
and GNU info.
&lt;sect&gt;The source
&lt;p&gt;</pre>
```

## Historia de XML

- Ej de XML:
  - Formato RSS –difusión noticias a usuarios suscritos a una fuente de contenidos-

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
<channel>
<title><! [CDATA[INFORMACION - Alicante]]></title>
<link><! [CDATA[http://www.diarioinformacion.com]]></link>
<description><! [CDATA[INFORMACION - Alicante]]></description>
<language>es-es</language>
<copyright><! [CDATA[Copyright INFORMACION]]></copyright>
<ttl>60</ttl>
<pubDate>2009-11-09T06:06:57Z</pubDate>
<lastBuildDate>2009-11-09T05:06:57Z</lastBuildDate>
<category>News</category>
<image>
<title><! [CDATA[INFORMACION]]></title>
<url><! [CDATA[http://www.diarioinformacion.com/favicon.ico]]></url>
<link><! [CDATA[http://www.diarioinformacion.com]]></link>
<description><! [CDATA[INFORMACION - Alicante]]></description>
</image>
<item>
<title><! [CDATA[Nick Nolte rodará en Ciudad de la Luz]]></title>
<link><! [CDATA[http://www.diarioinformacion.com/cine/2009/11/10/cine-nick-nolte-rodara-]]></link>
<description><! [CDATA[El actor norteamericano participa en el rodaje de "Mi querido des]></description>
<enclosure url='http://media.epi.es/www.diarioinformacion.com/media/fotos/noticias/150x150/103333_nick_nolte_en_ciudad_de_la_luz.jpg' type='image/jpeg' />
<author><! [CDATA[INFORMACION]]></author>
<guid isPermaLink="true"><! [CDATA[http://www.diarioinformacion.com/cine/2009/11/10/cine-nick-nolte-rodara-]]></guid>
<pubDate><! [CDATA[2009-11-10T15:47:40Z]]></pubDate>
</item>
<item>
<title><! [CDATA[Un hombre armado toma como rehén al director de una escuela en EEUU]]></title>
<link><! [CDATA[http://www.diarioinformacion.com/internacional/2009/11/10/internacional-]]></link>
<description><! [CDATA[Las autoridades están negociando con el secuestrador, que podría ser un]
```

## Historia de XML

- Ej de XML:
  - Vocabulario propio

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="boe.css"?>
<!DOCTYPE BOE SYSTEM "boe.dtd">
<BOE>
  <TITULO>BOLETIN OFICIAL DEL ESTADO</TITULO>
  <FECHA><AAAA>2001</AAAA><MM>03</MM><DD>27</DD></FECHA>
  <SUMARIO>
    <SECCION>
      <TIT-SECCION>I. Disposiciones Generales</TIT-SECCION>
      <APARTADO>
        <ORGANISMO>MINISTERIO DE ASUNTOS EXTERIORES</ORGANISMO>
        <PARRAFO>
          <TEXTO>
            <LEY>Ley 1/2001 de 5 de marzo</LEY>, de la cesión gratuita a las
            Autónomas el Impuesto sobre Animales y Cosas establecido por el
            <RD>REAL DECRETO 13/1991</RD>.
          </TEXTO>
          <PAGINA>11290</PAGINA>
        </PARRAFO>
        <PARRAFO>
          <TEXTO>
            <ORDEN>ORDEN de 28 de febrero de 2001</ORDEN> por la ...
          </TEXTO>
          <PAGINA>11290</PAGINA>
          <PAGINA>11291</PAGINA>
        </PARRAFO>
      </APARTADO>
      <APARTADO>
        <ORGANISMO>MINISTERIO DE HACIENDA</ORGANISMO>
        <PARRAFO>
          <TEXTO>
            <RD>REAL DECRETO 285/2001</RD>, de 19 de marzo, sobre ...
          </TEXTO>
          <PAGINA>11291</PAGINA>
          <PAGINA>11292</PAGINA>
          <PAGINA>11293</PAGINA>
          <PAGINA>11294</PAGINA>
```

## Cuestiones básicas de XML

- **Cuestiones básicas de XML:**

- ✓ ¿Qué es XML?
- ✓ ¿Para qué sirve?
- ✓ ¿Quién lo ha creado?
- ✓ ¿Por qué ha sido creado?
- ✓ ¿A qué se parece?
- ✓ ¿Es difícil?
- ✓ ¿Es útil en Internet?
- ✓ ¿Tiene diferentes versiones?
- ✓ Resumiendo, ¿simplifica las cosas?
- ✓ ¿y por qué no usar HTML como formato de intercambio de datos?



## Cuestiones básicas de XML

- ¿Qué es XML?
  - Traducción: *Lenguaje de Marcado eXtensible*
  - Fichero de texto donde la información viene **etiquetada**
  - Estándar de Internet para *intercambio de datos* desarrollado por **W3C**:
    - *lenguaje sencillo y flexible basado en SGML*
    - **Meta-lenguaje** –crea lenguajes que almacenan la información que necesita una aplicación concreta-
- ¿Para qué sirve?
  - Para **estructurar, almacenar y transportar** información, es decir, **organizar** datos de una forma **sencilla**
  - Para **estandarizar el intercambio de datos** entre aplicaciones, facilitando dicho intercambio de información entre **distintas plataformas y tecnologías**
  - Para crear **ficheros de configuración** para sistemas y móviles.
- Ej: desde *Office 2007*, cada fichero es un **contenedor** que incluye una **estructura** de carpetas y archivos con **extensión xml**

## Cuestiones básicas de XML

- ¿Quién lo ha creado?
    - W3C como encargada de la creación y revisión de estándares del mundo Web
    - Área **XML Activity** y sus diferentes grupos de trabajo:
      - *XML Coordination Group, XML Schema Working Group, .....*
  - ¿Por qué ha sido creado?
    - Para **evitar problemas** en el intercambio de datos de diferentes aplicaciones y sistemas
    - **Ahorra** trabajo de análisis y programación a compañías y desarrolladores
    - Para **simplificar** la creación de las gramáticas de lenguajes de marcas (**SGML**)
    - Ideal para **intercambiar información** de configuración de servidores entre equipos de plataformas distintas
- Ej: desarrollo de un portal financiero
1. Definir qué información se va a obtener y cómo publicarla
  2. Elaborar un estándar para el intercambio de información entre los proveedores y nuestro portal: formato, campos, tipos de datos, ...
  3. Desarrollar programas de comunicación e intercambio fiables
- ¡¡Cualquier mínimo **error**, provocará **fallos** desastrosos en la información!!

## Cuestiones básicas de XML

- ¿ A qué se parece?
  - a **HTML**:
    - Lenguaje de marcas
    - Basados en documentos de texto plano
    - Editados con cualquier editor de textos
    - Fácilmente entendibles a primera vista
  - Se **diferencia** en:
    - Más fiable y definido
    - Diseñado para transportar datos y no para visualizarlos
    - No tiene etiquetas predefinidas
- ¿ Es difícil?
  - **Fácil** aprendizaje
  - Necesario experiencia para su dominio
  - Aumenta la **complicación** con las **tecnologías** desarrolladas a su alrededor: **XLST, Xpath, DTD, XLM Schema, XML Query, XHTML, ...**

## Cuestiones básicas de XML

- ¿ Es útil en Internet?
  - Aporta **beneficios** en el desarrollo de aplicaciones
  - Está **integrado** en el conjunto de lenguajes y tecnologías adaptadas dentro del mundo Web
- ¿ Tiene diferentes versiones?
  - **1.0: primera versión**
    - Compatibilidad completa con UNICODE
    - Evita problemas de compatibilidad con diferentes herramientas
    - Recomendada por W3C desde **Noviembre 2008**
  - **1.1: segunda versión**
    - **Septiembre 2006**

## Cuestiones básicas de XML

- Resumiendo, ¿simplifica las cosas?
  - Simplifica la **compartición** de datos
    - Intercambiar datos con **formatos incompatibles** entre diferentes sistemas provoca **pérdida de datos** frecuentemente, además de consumir tiempo y recursos de los desarrolladores
  - Simplifica el **transporte** de datos
    - Al almacenar los datos en texto plano, proporciona una manera de **almacenar, transportar y compartir** datos totalmente **independiente** del software/hardware
  - Simplifica los **cambios de plataforma**
    - Es más fácil actualizar o expandir nuevos s.o., nuevas aplicaciones o nuevos navegadores sin pérdida de datos
  - Simplifica la **disponibilidad** de datos
    - para cualquier clase de *máquina lectora* –gente, servidores, ...-
  - Es **versátil y fácil**

## Cuestiones básicas de XML

- ¿y por qué no usar **HTML** como formato de intercambio de datos?
  - **Problemas:**
    - La mayoría de las etiquetas *no son semánticas* (**hasta HTML5**) => no sirven para decir el **tipo de contenido** que tenemos. Servían para indicar el formato –font-
    - Es un lenguaje rígido, **no es extensible** => no podemos añadir etiquetas ya que ningún navegador las reconocerá –sería necesario cambiar el estándar y los navegadores tienen que adaptar dichos cambios-
    - Requiere de arreglos “extraños” para añadir potencia y funcionalidad, por lo que los diseñadores tienden a incrustar dentro del código HTML código de lenguajes como **PHP** o **Javascript** que *dificultan su legibilidad* y comprensión

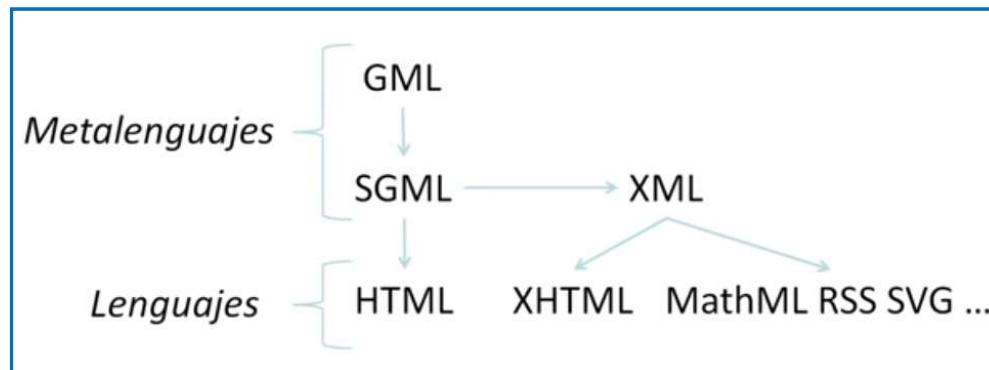
## Objetivos de XML

- **Objetivos de XML**
  - Debía de ser *similar* a HTML (ambos se basan en SGML)
  - Debía de ser *extensible*, es decir que sea posible añadir nuevas etiquetas sin problemas. Esa es la base del lenguaje XML
  - Debía de tener unas *reglas* concisas y *fáciles*, además de *estrictas*
  - Debía de ser fácil de implantar en *todo* tipo de *sistemas*
    - XML nace con una vocación *multiplataforma*, como base de **intercambio de información** entre sistemas de toda índole
  - Debía ser *fácil de leer* por los humanos y fácil crear software de procesadores XML (llamados *parsers*)
  - De **comprensión sencilla** de la estructura y el significado de los datos que utiliza
  - **Facilitar el intercambio de información** entre distintas plataformas y tecnologías
  - Diseñado para **evolucionar** de forma sencilla y manteniendo la **compatibilidad** con las versiones ya existentes

## Características de XML

### ■ Características de XML

- ✓ NO es un **lenguaje**, ya que no define un conjunto de etiquetas
- ✓ Es un **meta-lenguaje**:
  - ✓ Permite **crear lenguajes** ya que proporciona un *conjunto de reglas para poder definir las etiquetas y atributos que se van a usar, y pueda definir las relaciones q existen entre esas etiquetas.*
  - Permite **definir un lenguaje de marcas para cada necesidad específica**
    - Ej: almacenar información para matemáticas
    - Proporciona **reglas** para que cualquiera pueda **definir** su propio lenguaje:
      - con **etiquetas y atributos**
      - con **relaciones** entre las **etiquetas**
  - ✓ El conjunto de reglas en sí es la **gramática** del lenguaje



## Características de XML

### ■ Características de XML

- ✓ Es una *herramienta independiente* de cualquier software/hardware que permite **almacenar y transportar** datos
- ✓ Es un **estándar de intercambio de datos**
  - Alrededor de este estándar, se han construido multitud de **herramientas** que permiten trabajar con nuestros propios lenguajes basados en Xml => podemos crear nuestro propio lenguaje y aprovechar el resto de herramientas
- ✓ Formato muy **flexible** => permite representar diferentes formatos de información, tanto información estructurada como semiestructurada usando las mismas reglas
- ✓ Es **extensible** porque se pueden incorporar nuevos contenidos a los documentos –etiquetas/atributos- sin necesidad de realizar ningún cambio en las aplicaciones que lo utilizan.
- ✓ Antes, sólo algunos navegadores reconocían y visualizaban los .xml
- Los *datos se describen a sí mismos* => se **conserva la información**
  - **Ej1:** un documento Wordperfect de formato binario difícil de interpretar sin conocer el algoritmo de creación
  - **Ej2:** un documento XML lleva información para interpretar correctamente los datos que almacena

## Aplicaciones XML

### ■ Aplicaciones XML

- Referido a todo **lenguaje de marcas basado en XML** y diseñado para crear un **vocabulario** específico que se **utilice** en un determinado ámbito o **dominio**.
- **Algunos vocabularios:**
- ✓ ***Chemical Markup Language (CML)***: lenguaje para expresar información molecular y datos de química, cristalografía, materiales, etc.
  - **Ej:** expresar la fórmula del agua

```
<?xml version="1.0" ?>
<cml xmlns="http://www.xml-cml.org/schema/cml2/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance"
      xsi:schemaLocation="http://www.xml-
      cml.org/schema/cml2/core/cmlCore.xsd">
<molecule title="Water">
    <atomArray>
        <atom id="a1" elementType="H" hydrogenCount="0" />
        <atom id="a2" elementType="O" hydrogenCount="2" />
        <atom id="a3" elementType="H" hydrogenCount="0" />
    </atomArray>
    <bondArray>
        <bond atomRefs2="a1 a2" order="1" />
        <bond atomRefs2="a2 a3" order="1" />
    </bondArray>
```

- **Aplicaciones XML**
  - **Algunos vocabularios:**
  - ✓ **DocBook:** lenguaje utilizado en el área de **documentación técnica**, para documentar materiales, programas, etc.
    - **Ej:** indicar la estructura de un libro

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book lang="es" id="simple libro">
  <title>Un libro muy simple</title>

  <chapter id="capitulo_1">
    <title>Capítulo 1</title>
    <para>Hola mundo!</para>
    <para>¡Yo espero que tu día sea bueno!</para>
  </chapter>

  <chapter id="capitulo_2">
    <title>Capítulo 2</title>
    <para>Hola otra vez, mundo!</para>
  </chapter>
</book>
```

## Aplicaciones XML

- **Aplicaciones XML**

- **Algunos vocabularios:**

- ✓ **Keyhole Markup Language (KML)**: lenguaje utilizado para mostrar información geográfica en aplicaciones como **Google Earth** y **Google Maps**, desarrollado por Google. Estandarizado posteriormente por la *OGC*.

- **Ej:** indicar una determinada posición de Nueva York

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <name>New York City</name>
      <description>New York City</description>
      <Point>
        <coordinates>-74.006393,40.714172,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

## Aplicaciones XML

- Aplicaciones XML

- Algunos vocabularios:

- ✓ **Scalable Vector Graphics (SVG)**: lenguaje utilizado para mostrar el formato de gráficos vectoriales. Desarrollado por W3C.

- Ej: mostrar un triángulo de color rosa

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg"
      width="12cm" height="8cm">
    <title>The pink triangle!!!</title>
    <text x="10" y="15">This is SVG!</text>
    <polygon style="fill: pink"
              points="0,311 180,0 360,311" />
</svg>
```

## Aplicaciones XML

- **Aplicaciones XML**

- **Algunos vocabularios:**

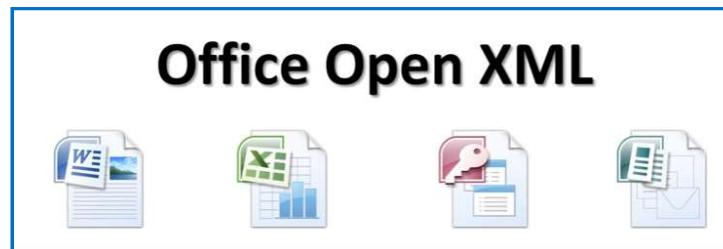
- ✓ ***Mathematical Markup Language (MathML)***: lenguaje utilizado para expresar fórmulas y ecuaciones **matemáticas** entendibles por humanos y ordenadores. Utilizado junto con **XHTML** para mostrar expresiones matemáticas en páginas web y para intercambiar información en programas matemáticos.
    - ✓ ***Systems Biology Markup Language (SBML)***: lenguaje que permite intercambiar información de **biología** (redes metabólicas, señalización celular, regulaciones genéticas, etc.).
    - ✓ ***Dublin Core***: lenguaje utilizado para realizar **búsquedas**, tanto en bases de datos, como en programas y en sitios web.
    - ✓ ***Open eBook* y *ePub***: formato de **libros digitales**. Son ficheros comprimidos en ese formato, que contiene ficheros **.xml** y metadatos de la publicación electrónica.

## Aplicaciones XML

### ■ Aplicaciones XML

#### ■ Algunos vocabularios:

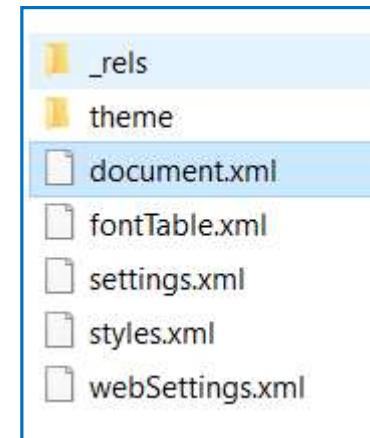
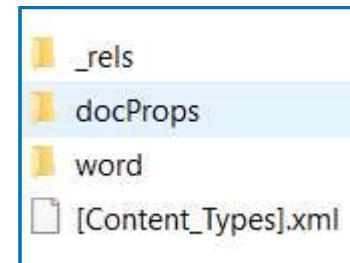
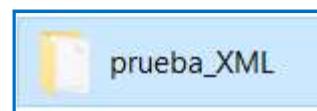
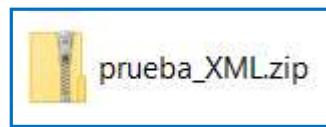
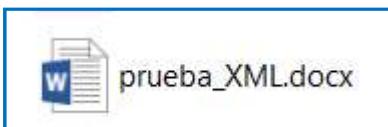
- ✓ **Open Document**: formato para crear ficheros de paquetes ofimáticos como **OpenDocument** y **LibreOffice** donde cada fichero se guarda como una estructura de subcarpetas y ficheros xml.
- ✓ **Office Open XML**: compite con el formato anterior. Desarrollado inicialmente por **Microsoft**. Querían evitar trabajar con ficheros binarios. Se estandarizó a partir de 2006.



➤ **Comprueba**: crea un documento con una versión de Word a partir de 2007 y guárdalo con el formato predeterminado **.docx**. Para comprobar si realmente dicho documento es un **contenedor** que contiene una estructura de carpetas y ficheros **xml**, cámbiale la extensión a **.zip**. Descomprime dicho fichero y observa la estructura que se ha creado.

## Estructura interna de un .docx

### ➤ Solución:



```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<w:document mc:Ignorable="w14 w15 w16se wp14" xmlns:wps="http://schemas.microsoft.com/office/word/2006/wordml" xmlns:wpi="http://schemas.microsoft.com/office/word/2006/relationships/picture" xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml" xmlns:wp="http://schemas.microsoft.com/office/word/2010/wordprocessingML" xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup" xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml" xmlns:w16="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing" xmlns:v="urn:schemas-microsoft-com:vml" xmlns:m="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" compatibility="2006" xmlns:cx="http://schemas.microsoft.com/office/drawing/2006/chart" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
    <w:body>
        - <w:p w:rsidRDefault="0097653D" w:rsidR="000431EA">
            - <w:r>
                <w:t>Creando un documento en formato docx y abrirlo después para ver el código XML</w:t>
            </w:r>
        </w:p>
        - <w:p w:rsidRDefault="0097653D" w:rsidR="0097653D">
            - <w:r>
                <w:t xml:space="preserve">Aquí características d formato: </w:t>
            </w:r>
            - <w:r w:rsidRPr="0097653D">
                - <w:rPr>
                    <w:b/>
                </w:rPr>
                <w:t>negrita</w:t>
            </w:r>
        </w:p>
    </w:body>
</w:document>
```

## Software para edición XML

- **Software para edición XML**
  - a) **Editores XML:** se puede utilizar un editor de texto o uno específico de XML
    - **Notepad++**
      - Juego caracteres predeterminado: Ansi (NO sirve para español)
      - para Windows
      - <http://notepad-plus-plus.org/>
    - **XML Copy Editor**
      - para Windows y Linux
      - <http://xml-copy-editor.sourceforge>
    - **Amaya**
      - Creado por W3C para Windows, Linux y Mac OS
      - <https://www.w3.org/Amaya/>
    - **Code Browser**
      - Es un editor diseñado para Windows y Linux muy fácil de usar
      - <https://tibleiz.net/code-browser/download.html>
  - b) **Procesadores XML:** software que posee un **analizador de XML** que comprueba que el documento cumple las normas establecidas

## Estructura de un documento XML

- **Estructura de un documento XML**
  - Cada documento .xml está formado por una mezcla de **datos** e información de **etiquetado** sobre los mismos
  - Un documento XML está formado por una **jerarquía de elementos** a partir de un **elemento raíz único**
  - **Elemento**: componente semántico **básico** de un documento
    - consta de una **etiqueta de inicio**, un **contenido** y una **etiqueta de fin**
    - **Elemento raíz**: elemento obligatorio que se abre tras el *prólogo* y se debe cerrar justo al final
  - **Partes**:
    - 1) **Prólogo**: primera zona del documento y sirve para **describir qué tipo** de documento es
    - 2) **Cuerpo**: contiene los **elementos**
      - **Elemento raíz**: elemento obligatorio que se abre tras el prólogo y se debe cerrar justo al final del documento. Cualquier elemento está **dentro** del elemento raíz

## Estructura de un documento XML

- **Partes de un documento XML**
  - 1) **Prólogo**: primera zona del documento y sirve para **describir qué tipo** de documento es
    - Similar al apartado **head** de **HTML**
    - Puede contener:
      - ✓ **Declaración del documento**: indica el tipo de documento XML que es. Es **opcional** pero se **recomienda** incluirla
      - ✓ **Instrucciones** para el procesado del documento
      - ✓ **Comentarios**
      - ✓ **Indicaciones del documento DTD, XSD o RelaxNG**: para comprobar si el mismo es **válido** según las **reglas** impuestas por dicho documento – **reglas de validación**–
      - ✓ **Indicaciones de otros documentos que afectan al actual**: documentos que permiten especificar la forma en la que el documento se debe mostrar en pantalla, por ejemplo **XSLT**

## Estructura de un documento XML

- **Partes de un documento XML**

- 1) **Prólogo:**

- **Ej de una declaración de un documento XML:**

```
<?xml version="1.0" encoding="UTF-16" standalone="yes" ?>
```

- **encoding:** indicar tipo de codificación
- **standalone:** relación entre el documento y su definición
  - **yes:** la especificación *DTD* se encuentra **dentro** del documento
  - **no:** la especificación *DTD* es **externa**
- Si no se indica los atributos, los **valores por defecto** son:
  - *version*: “1.0”, codificación de caracteres “**UTF-8**” y *DTD* interno (standalone con el valor “yes”)
- Si se indican los atributos:
  - en ese orden, los valores entre comillas, y sólo el tipo de codificación en mayúsculas

## Estructura de un documento XML

- **Partes de un documento XML**

- 2) **Cuerpo: Elemento raíz único**

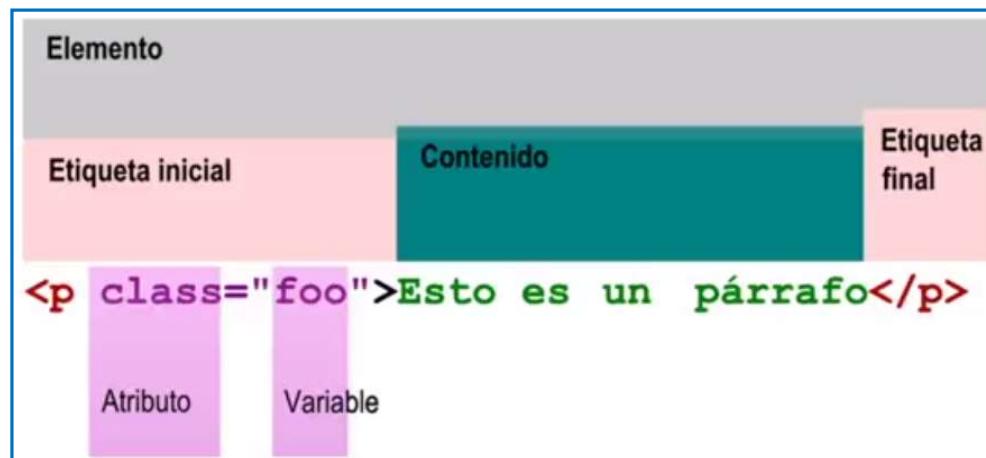
- Se corresponde con el **cuerpo** del documento
- Todo el contenido del documento debe de estar incluido en él
- Elemento **obligatorio** que se abre tras el prólogo y se debe cerrar justo al final
- Cualquier elemento está dentro del **elemento raíz**
- **Contiene:**
  - ✓ Otros **elementos**
  - ✓ **Atributos**
  - ✓ **Texto normal**
  - ✓ **Entidades**
  - ✓ **Comentarios**

## Estructura de un documento XML

- Partes de un documento XML

- 2) Cuerpo: contiene los **elementos**

- Estructura de un elemento:
    - *Etiqueta inicial y final*, con el mismo nombre y entre < >
    - Sólo dentro de la etiqueta **inicial** puede haber **atributos**
    - Los **atributos** deben llevar un **valor**
    - Las etiquetas pueden llevar **contenido –texto** y otros **elementos**
    - *Cada dato de un documento XML tiene que formar parte del contenido de algún elemento*
    - Ej:



## Estructura de un documento XML

- Ejemplos de elementos según contenido:

- a) Texto

- Siempre **entre** una etiqueta de apertura y una de cierre => es **parte** de un **elemento XML**
    - Válido cualquier carácter Unicode, excepto < y >
    - Ej: <**descripción**>Producto rebajado</**descripción**>

- b) Elementos:

- Hay que cerrar primero la última etiqueta abierta para **evitar solapamientos**  
–**Regla de documentos bien formados**–
    - Ej: <**persona**>

<**nombre**>Pepe</**nombre**>

<**apellido**>Pérez</**apellido**>

</**persona**>

- a) **Vacío**: para aquellos elementos que no tienen contenido. Dos formas:

- Ej1: <**casado**></**casado**>
    - Ej2: <**casado** />

## Estructura de un documento XML

- **Reglas básicas sobre los *nombres* de los elementos:**
  - ✓ Debe **coincidir** en la etiqueta de apertura y cierre (tiene que ser el mismo)
  - ✓ **Sensibles** a mayúsculas/minúsculas
  - ✓ Deben **comenzar** por un carácter alfabético (acentuado o no) o un guión bajo “\_”
  - ✓ **No puede** comenzar por un número, ni por un punto, ni por un guión medio
  - ✓ **No pueden** comenzar por la combinación de letras **XML**, independientemente de cómo esté escrita
  - ✓ **NO se recomienda** empezar con : (problemas con reglas de *espacio de nombres*)
  - ✓ **No pueden** tener **espacios en blanco**
  - ✓ Pueden **contener** cualquier **carácter alfabético** (acentuado o no) y los caracteres “.”, “\_” y “-”
  - ✓ También pueden contener el carácter “:” pero se desaconseja su uso
- **Ej:** ¿qué nombres son válidos y cuáles no?

<módulo>	<módulo Daw>
<_sesión>	<+sesión>
<módulo_daw>	<xmlCurso>
<móduloDaw>	<xMlCurso>

## Estructura de un documento XML

- **Atributos:**

- ✓ Representa una **propiedad** de un elemento: modo de incorporar **características o información complementaria** a los elementos
- ✓ Compuesto por una pareja de **nombre y valor**
- ✓ Un elemento puede **contener o no** atributos, pero sólo en la **etiqueta de inicio**.
- ✓ Su **valor** debe ir entre **comillas dobles o simples** y hay que utilizar el mismo tipo para apertura y cierre
- ✓ Siempre tienen que tener valor, o asignarles "" (o '')
- ✓ **Mismas restricciones** para su nombre que los nombres de las etiquetas
- **Ej:** ¿qué atributos son válidos y cuáles no?

```
<elem at1="valor1" at2="valor2" at1="valor3">...</elem>
<teléfono tipo="fijo">999999999</teléfono>
<elemento atrib1=""></elemento>
<elemento atrib1></elemento>
```

## Estructura de un documento XML

- **¿Atributos o Elementos?**
  - Hay veces que para representar una información en un documento XML podemos dudar entre utilizar un elemento o un atributo.
- **Consideraciones:**
  - **Elementos:**
    - ✓ Pueden contener más elementos en su interior
    - ✓ El orden en el que aparecen es significativo
    - ✓ Pueden tener atributos
    - ✓ Puede haber múltiples ***ocurrencias*** de un elemento
  - **Atributos:**
    - ✓ Están vinculados/asociados con los elementos
    - ✓ Son *modificadores* de la información de los elementos
    - ✓ El **orden** en el que aparecen **no** es significativo
    - ✓ NO se pueden extender con elementos en su interior
    - ✓ NO pueden aparecer varias veces en el mismo elemento

## Estructura de un documento XML

- **¿Atributos o Elementos?**
- **Símil:** un elemento es un **sustantivo** y un atributo un **adjetivo**
- **Recomendaciones:**
  - ✓ Si la información tiene una **estructura interna** debe ser un **elemento**
  - ✓ Si contiene una **gran cantidad de información**, es más adecuado un **elemento**
  - ✓ Los mecanismos de procesamiento y presentación de documentos, permiten tener mejor control sobre los elementos
  - ✓ Información con procesamiento o presentación **complejos** debe ser un **elemento**
  - ✓ Los **atributos** **no** pueden contener **varios valores**
  - ✓ Los atributos **no** pueden contener **estructuras en árbol**
  - ✓ Los atributos **no** se pueden **extender** fácilmente para **futuros cambios**
  - ✓ En caso de duda, **elemento**

## Estructura de un documento XML

- **¿Atributos o Elementos?**
  - **Deduce:** observa los siguientes ejemplos e indica si facilitan la misma información o no.

### Ej1:

- ```
<persona genero="femenino">
    <nombre>Ana</nombre>
    <apellido>Pérez</apellido>
</persona>
```

### Ej2:

- ```
<persona>
    <genero>femenino</genero>
    <nombre>Ana</nombre>
    <apellido>Pérez</apellido>
</persona>
```

## Estructura de un documento XML

- ¿Atributos o Elementos?
  - Cada solución es más o menos adecuada **dependiendo del uso** que se vaya a hacer del documento
  - **Deduce:** ¿creees que tiene sentido facilitar la información relacionada con el elemento **nota** toda con atributos?

```
<nota>
  <fecha>26/11/2013</fecha>
  <para>Pepito</para>
  <de>Marta</de>
  <encabezado>Recordatorio</encabezado>
  <cuerpo>No te olvides de llamarme</cuerpo>
</nota>
```

```
<nota fecha="26/11/2013">
  <para>Pepito</para>
  <de>Marta</de>
  <encabezado>Recordatorio</encabezado>
  <cuerpo>No te olvides de llamarme</cuerpo>
</nota>
```

```
<nota>
  <fecha>
    <dia>26</dia>
    <mes>11</mes>
    <año>2013</año>
  </fecha>
  <para>Pepito</para>
  <de>Marta</de>
  <encabezado>Recordatorio</encabezado>
  <cuerpo>No te olvides de llamarme</cuerpo>
</nota>
```

```
<nota dia="26" mes="11" año="2013" para="Pepito"
      de="Marta" encabezado="Recordatorio"
      cuerpo="No te olvides de llamarme" />
```

## Reglas básicas de documento XML bien formado

- **¿Cómo se crea un documento XML correcto?**
  - 1) **Bien formado:** si cumple todas las reglas de creación de documento
    - Ej: se utilizan caracteres válidos para el nombre de una etiqueta
  - 2) **Válido:** si está bien formado y además cumple la **definición del lenguaje** que se está empleando (**DTD** o **XSD**)
    - Ej: los nombres de las etiquetas y de los atributos son los correctos
- **Reglas básicas de documento XML bien formado**
  1. Incluir la **declaración** de procesamiento XML: `<?xml version="1.0" encoding="UTF-1`
  2. Añadir *elementos* en el **cuerpo**
  3. Cada documento debe tener un elemento **raíz único**
  4. **Estructura jerárquica** (organización de los elementos entre sí)
  5. **No solapamiento** (elementos anidados correctamente)
  6. **Cumplir las reglas básicas** para los nombres de **elementos** y **atributos**.
    - Ej: ¿el elemento libro está bien formado? <LIBRO>Hambre</libro>
  7. **Elementos vacíos**
    - Ej: <LIBRO></LIBRO> o <LIBRO />

## Reglas básicas de documento XML bien formado

**8. Atributos entre comillas ‘simples’ o “dobles”**

**9. Utilizar *entidades* para usar comillas dentro del valor del atributo**

- Ej: &quot; - para comillas dobles – y &apos; - para comillas simples-

**10. Atributos reservados para las especificaciones XML**

- Comienzan con el prefijo **xml**:
  - **xml:lang** => para especificar el idioma del elemento
  - **xml:space:default | preserve** => para definir el procesamiento normal o no de espacios en blanco (conservándolos)
  - **xml:id** => para definir un identificador único del elemento en el documento

**11. Comentarios**

- <! -- no pueden meterse dentro de la etiqueta de un elemento. Añadirlos después de la declaración de XML -- >

**12. Caracteres equivalentes para los espacios en blanco**

- Tabulador, avance línea, retorno carro y espacio blanco

## Reglas básicas de documento XML bien formado

**13. Caracteres especiales:** representarlos con *entidades* de carácter cuando se necesite utilizarlos como literales.

- Si no se hace así, escribir directamente cualquiera de esos símbolos especiales provocará un **error** en el documento XML
- **Entidades en XML:**

Carácteres no permitidos	Sustitución de la entidad
<	&lt;
>	&gt;
&	&amp;
'	&apos;
"	&quot;

## Reglas básicas de documento XML bien formado

### 14. Secciones o elemento CDATA (Character Data)

- El texto contenido en las secciones **CDATA** no se interpretan como XML por el **analizador XML**, ya que las **ignorado** y las interpreta como texto.
- En dichas secciones se puede añadir además de texto, código de algún lenguaje de programación.
- Al utilizar la sección `<![CDATA...]>` se mostrará el texto tal y como está escrito, independientemente de los símbolos utilizados.
- **Sintaxis:** `<! [CDATA [ Texto no procesable... ] ] >`
- **Ej1:** añadir código Html dentro de un documento Xml

**Ej1:** `<cod>&lt;h1&gt;El operador de concatenación: &amp;&lt;/h1&gt;</cod>`

**Utilizando CDATA:** `<cod><![CDATA[<h1>El operador de concatenación:&</h1>]]></cod>`

- **Ej2:**

```
<?xml version="1.0" encoding="UTF-8"?>
<prueba>
    <texto><![CDATA["Los caracteres < y & no pueden escribirse si
no es como comienzo de marcas"]]></texto>
</prueba>
```

## Ejemplo de documento XML mal formado

- **Ejemplo1:** dado el siguiente código, corrige los errores de los nombres y valores de los elementos y atributos. Intenta visualizar dicho documento desde el navegador.

```
tarea1.xml *  
1 <xml version="1.0" encoding="utf-8">  
2 <!-- Corrige los errores de los nombres y valores  
3 <colección musical>  
4   <artista nome=Queen>  
5     <album nome=A Night at the Opera>  
6       <Bohemian Rhapsody>  
7         <'39>  
8         <album>  
9       </artista>  
10      <artista nome=R.E.M.>  
11        <album nome='Life's Rich Pageant'>  
12          <Superman>  
13          <These days>  
14        </album>  
15      </artista>  
16    </colección musical>
```

## Ejemplo de documento XML bien formado

- Sol. Ejemplo1: corregir errores sintácticos y crear elementos para almacenar el título de las canciones

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Errores corregidos-->
3 <colección_musical>
4   <artista nome="Queen">
5     <album nome="A Night at the Opera">
6       <canción>Bohemian Rhapsody</canción>
7       <canción>'39</canción>
8     </album>
9   </artista>
10  <artista nome="R.E.M.">
11    <album nome="Life's Rich Pageant">
12      <canción>Superman</canción>
13      <canción>These days</canción>
14    </album>
15  </artista>
16 </colección_musical>
17
```

## Visualización de un documento XML desde un navegador

### ➤ Sol. Ejemplo1: Visualización desde el navegador *Firefox* y desde *Edge*

Este fichero XML no parece tener ninguna información de estilo asociada. Se muestra debajo el árbol del documento.

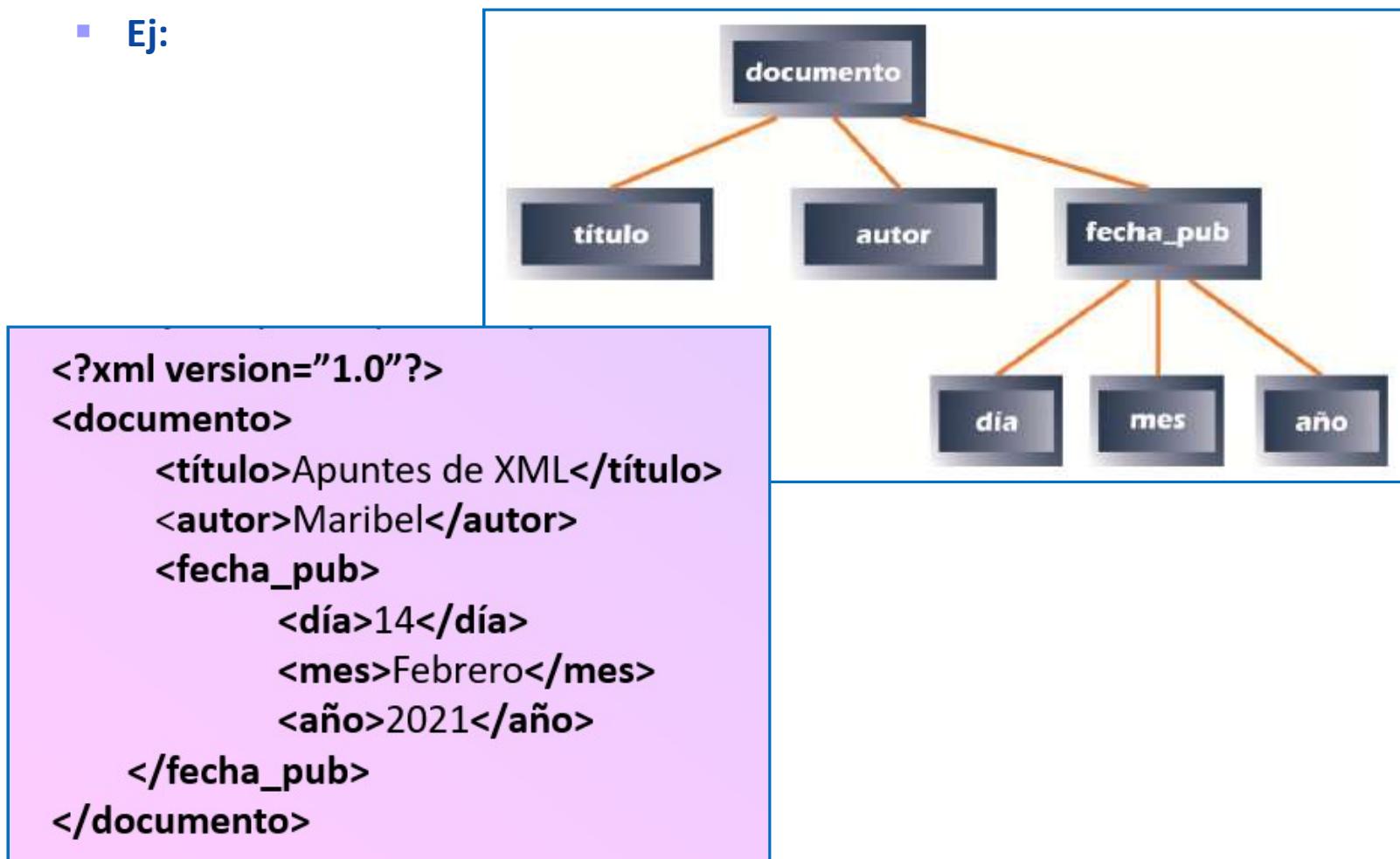
```
<!--documento xml bien formado -->
-<colección_musical>
--<artista nome="Queen">
--<album nome="A Night at the Opera">
  <canción>Bohemian Rhapsody</canción>
  <canción>'39</canción>
</album>
</artista>
--<artista nome="R.E.M.">
--<album nome="Life's Rich Pageant">
  <canción>Superman</canción>
  <canción>These days</canción>
</album>
</artista>
</colección_musical>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--documento xml bien formado -->
- <colección_musical>
  + <artista nome="Queen">
  - <artista nome="R.E.M.">
    + <album nome="Life's Rich Pageant">
    </artista>
  </colección_musical>
```

## Jerarquía XML

- **Jerarquía XML**

- Los elementos de un documento XML establecen una **jerarquía** que **estructura** el contenido del mismo
- Esa jerarquía se puede representar en forma de **árbol**
- Ej:



## Primera aproximación a un documento XML

- **Pasos para ir creando nuestro primer documento XML**
  1. Utilizar el **bloc de notas** y crear un fichero con extensión **.xml**
  2. Saber qué información se quiere almacenar y cómo se va a ir asignando: como **elemento** o como **atributo** -asignándole el valor correspondiente-.
  3. Diseñar una **estructura jerárquica** para ir almacenando la información que se quiere guardar.
  4. Recomendable incluir la **declaración** de procesamiento al principio de todo.
  5. Continuar definiendo el **elemento raíz**, con su etiqueta de apertura y de cierre.
  6. A medida que se van creando el resto de elementos, ir comprobando desde el navegador si es capaz de visualizar el contenido del documento. Ante cualquier fallo, no mostrará nada.
    - **Fallo1:** si el editor es el bloc de notas y usa como sistema de codificación **ANSI**, dará error las vocales acentuadas y las “ñ”.
    - **Fallo2:** si nos olvidamos de cerrar alguno de los elementos
    - **¡No hay una única solución!! =>** hay que encontrar la más adecuada según el **uso** que se haga del documento.

## Primera aproximación a un documento XML

- **Creación de un primer documento XML:** sigue las recomendaciones anteriores y crea un documento **XML bien formado** que permita almacenar información sobre las **personas** que conozco, diferenciándolas de si son del *trabajo* o no (cualquier amigo de la infancia, del vecindario, etc.).
- Tras diferenciar dicho aspecto, hay que almacenar información sobre su *dirección postal*, su *teléfono*, *dirección email*, qué coche tiene, cuándo es su cumpleaños y cuál es su comida favorita.
  - Además, también interesa almacenar una serie de características personales, tales como la altura, el peso, color del pelo y cualidad que lo describe mejor.
  - Tras definir la estructura, añade información de varias personas y comprueba si se puede ver su estructura desde el navegador.
  - Observa las etiquetas y orden de los elementos que han creado tus compañeros ¿hay muchas diferencias? ¿podrían ser compatibles dichos documentos?

## Primera aproximación a un documento XML

### ➤ Possible sol.:

```
<?xml version="1.0" encoding="UTF-8" ?>
<people>
    <work>
        <george height="1,75" weight="80" personality="nice">
            <address>Torre del Oro, s/n</address>
            <phone>630120012</phone>
            <email>george@gmail.com</email>
            <birthday>03/03/93</birthday>
            <cartype>No car</cartype>
            <favFood>Anything</favFood>
        </george>
    </work>
    <home>
        <paul height="1,80" weight="85" personality="hardworking">
            <address>Torre de Plata, 24</address>
            <phone>670120012</phone>
            <email>paul@gmail.com</email>
            <birthday>03/06/90</birthday>
            <cartype>Toyota Corolla</cartype>
            <favFood>Anything</favFood>
        </paul>
    </home>
</people>
```

## Ejercicio de documento XML bien formado

- **Ejercicio1:** Crea un documento XML bien formado desde **XML Copy Editor** que permita almacenar información sobre canciones. Cumple con los siguientes **requisitos**:
- Define la primera línea de **declaración** del documento XML incluyendo únicamente información sobre la versión (1.0)
  - Define como elemento raíz **<canciones>**
  - Define el elemento **<cancion>** que contendrá información sobre una única canción con los siguientes datos:
    - ✓ Artista
    - ✓ Título
    - ✓ Duración
    - ✓ Url de la página oficial donde comprar dicha canción
  - Comprueba si está bien formado
  - ¿Se visualiza jerárquicamente desde el navegador?

## Ejercicio de documento XML bien formado

### ➤ Sol. Ej1:

Este fichero XML no parece tener ninguna información de estilo asociada. Se muestra debajo el árbol del documento.

```
--<canciones>
  --<cancion>
    <artista>Queen</artista>
    <titulo>A Night at the Opera</titulo>
    <duracion>5 minutos</duracion>
    <url>www.queen.com</url>
  </cancion>
  --<cancion>
    <artista>R.E.M.</artista>
    <titulo>Superman</titulo>
    <duracion>4 minutos</duracion>
    <url>www.rem.com</url>
  </cancion>
</canciones>
```

## Ejercicio de documento XML bien formado

- **Ejercicio2:** Crea un documento XML bien formado desde **XML Copy Editor** que permita almacenar información sobre **cursos**. Cumple con los siguientes **requisitos**:
- Estructura qué elementos/atributos se deben de crear y en qué orden
  - Define la primera línea de **declaración** del documento XML incluyendo únicamente información sobre la versión (1.0)
  - Recopila información sobre:
    - ✓ **Título** de curso
    - ✓ **Contenido**
    - ✓ **Título** del contenido
    - ✓ **Temas** a desarrollar (según contenido)
    - ✓ Cada tema tiene también su **título**
  - ¿Se visualiza jerárquicamente desde el navegador? Colapsa y expande los diferentes elementos
  - Provoca algún error eliminando alguna etiqueta. ¿Qué indica el navegador?
  - Observa las diferencias con respecto a la estructura de tus compañeros

## Ejercicio de documento XML bien formado

### ➤ Sol. Ej2:

```
+ <cursos></cursos>
```

```
- <cursos>
  + <curso></curso>
</cursos>
```

```
- <cursos>
  - <curso>
    <titulo>Introducción a XML</titulo>
    + <contenido titulo="XML"></contenido>
    </curso>
</cursos>
```

```
- <cursos>
  - <curso>
    <titulo>Introducción a XML</titulo>
    - <contenido titulo="XML">
      - <temas>
        <tema titulo="Introducción">Introducción a los lenguajes de marcas </tema>
        <tema titulo="Desarrollo">Evolución y desarrollo de los lenguajes de marcas </tema>
        <tema titulo="Ejemplos">Diferentes ejemplos de los lenguajes de marcas </tema>
      </temas>
    </contenido>
  </curso>
</cursos>
```

## Ejercicio de documento XML bien formado

- **Ejercicio3:** A partir del siguiente documento XML, corrige los diferentes errores y crea el documento XML bien formado correspondiente. Considera la posibilidad de convertir algún elemento a atributo y viceversa.
- La información que se debe almacenar es:
    - ✓ **Programas** para un determinado período de tiempo
    - ✓ Para cada programa indicar **nombre, licencia y sistemas operativos**.
  - Una vez que hayas terminado comprueba si se visualiza jerárquicamente desde el navegador, lo cual indica que está bien formado.

```
<?xml version="1.0" encoding="utf-8">
<programas>
  <2008-2013>
    Programas con fecha > 2008 y < 2013
    <programa nombre="Notepad+
    +" licencia=GPL licencia=LGPL Windows />
    </programa>
    <programa nombre="XML Copy Editor" licencia=GPL Windows Linux />
    </programa>
  </2008-2013>
</programas>
```

## Ejercicio de documento XML bien formado

### ➤ Sol. Ej3:

```
<?xml version="1.0" encoding="UTF-8"?>
<programas>
    <periodo rango="2008-2013">
        Programas con fecha &gt; 2008 y &lt; 2013
        <programa>
            <nombre>Notepad++</nombre>
            <licencia>GPL</licencia>
            <licencia>LGPL</licencia>
            <sist_op>Windows</sist_op>
        </programa>
        <programa>
            <nombre> XML Copy Editor </nombre>
            <licencia> GPL</licencia>
            <sist_op>Windows</sist_op>
            <sist_op>Linux</sist_op>
        </programa>
    </periodo >
</programas>
```

## Ejercicio de documento XML bien formado

- **Ejercicio4:** crea un documento XML bien formado a partir de la siguiente información:

Ciudades		
Nombre	País	Continente
Nueva Delhi	India	Asia
Lisboa	Portugal	Europa
Rabat	Marruecos	África

- Una vez que hayas terminado comprueba si se visualiza jerárquicamente desde el navegador, lo cual indica que está bien formado.

## Ejercicio de documento XML bien formado

### ➤ Sol. Ej4:

```
<ciudades>
    <!-- Ciudades de Asia -->
    <ciudad>
        <nombre>Nueva Delhi</nombre>
        <pais>India</pais>
        <continente>Asia</continente>
    </ciudad>
    <!-- Ciudades de Europa -->
    <ciudad>
        <nombre>Lisboa</nombre>
        <pais>Portugal</pais>
        <continente>Europa</continente>
    </ciudad>
    <!-- Ciudades de África -->
    <ciudad>
        <nombre>Rabat</nombre>
        <pais>Marruecos</pais>
        <continente>África</continente>
    </ciudad>
</ciudades>
```

## Ejercicio de documento XML bien formado

- **Ejercicio5:** Crea un documento XML bien formado para almacenar información sobre los libros de una biblioteca.
- Se sigue la especificación 1.0 de XML
  - Para cada **libro** deberá aparecer:
    - ✓ **ISBN**
    - ✓ **Título**
    - ✓ **Año** de publicación
    - ✓ **Editorial**
    - ✓ Nombre del **autor/autores**
    - ✓ **Descripción:** cada libro puede tener una o varias descripciones, o ninguna.
    - ✓ En cada descripción se indica el **idioma** en el que está escrita.
  - Una vez que hayas terminado comprueba si se visualiza jerárquicamente desde el navegador.
  - Compara los nombres de elementos/atributos y el orden de aparición de tu documento XML con el de tus compañeros ¿es exactamente el mismo?

## Ejercicio de documento XML bien formado

➤ Sol. Ej5:

```
<?xml version="1.1" encoding="UTF-8"?>
<biblioteca>
    <libro ISBN="9788467028317">
        <título>Pequeña Historia de España</título>
        <año_public>2008</año_public>
        <editorial>Espasa</editorial>
        <autor>Manuel Fernández Álvarez</autor>
        <sinopsis xml:lang="es">Ofrece un recorrido por la historia de
            España, desde la Prehistoria a nuestros días, presentada y narrada
            como si fuera un cuento o un relato de aventuras.</sinopsis>
    </libro>
    <libro ISBN="978-8491220275">
        <título>El loco cuento de todos los cuentos</título>
        <año_public>2013</año_public>
        <editorial>Santillana</editorial>
        <autor>Jose María Plaza</autor>
        <sinopsis xml:lang="es">Divertida y alocada historia donde nada es
            lo que parece.</sinopsis>
        <sinopsis xml:lang="en"> Fun and crazy story where nothing is what
            it seems</sinopsis>
    </libro>
</biblioteca>
```

## ¿Cómo saber si la estructura de un documentos XML es válida?

- En los ejercicios vistos hasta ahora, la estructura de cada documento se crea según se **interpreta** la información que se debe de almacenar.
  - Se pueden adoptar **múltiples soluciones**, tanto en el nombre de los elementos, en decidir si son elementos o atributos, en el orden, nivel de anidamiento, etc.
  - Al diseñar documentos de forma **arbitraria**, dichos documentos **NO** serán **compatibles** entre sí.
- ¿Cómo se puede saber si la estructura de un documento XML es válida?
- Para **describir cuál es la estructura de un documento XML**, se debería poder especificar los siguientes **requisitos**:
    - 1) Qué **nombres** se deben emplear para los **elementos** de un documento XML, cuántas veces se pueden usar –**cardinalidad**- y en qué **orden**
    - 2) Qué **nombres** se deben emplear para los **atributos**, qué elementos los usan y si son **obligatorios** u opcionales
    - 3) Los posibles **valores**, predeterminados o permitidos que admiten los **atributos**
  - Una vez definida dicha **estructura** los usuarios podrán crear **documentos XML válidos => todos** los documentos cumplen las **mismas reglas**

## Documentos XML válidos

- **¿Cómo se crea un documento XML correcto?**
  - 1) **Bien formado**: si cumple todas las reglas de creación de documento
    - Ej: se utilizan caracteres válidos para el nombre de una etiqueta
  - 2) **Válido**: si está bien formado y además cumple la **definición del lenguaje** que se está empleando (**DTD** o **XSD**)
    - Ej: nombres de atributos correctamente estructurados: qué elementos los usan, si son obligatorios u optionales, qué valores admiten, ...
- **¿Cómo se especifica la gramática de un lenguaje?**
  - Para especificar la **definición de un documento XML** podemos emplear:
    - a) **DTD (Definición de Tipo de Documento)**: archivo de definición
    - b) **XML Schema (esquema XML)**: lenguaje de esquema basado en la gramática más complejo y más potente que **DTD**. Utiliza sintaxis XML.
    - c) **RelaxNG**: lenguaje de esquema basado en la gramática, más intuitivo y fácil que **XML Schema**.
    - d) **Schematron**: para definir lo que se permite en un documento XML utiliza *expresiones de acceso o afirmaciones* en lugar de gramaticales

## Definición de DTD

- ¿En qué consiste la *validación* de un documento XML?  
*comparar un documento XML con un conjunto de reglas definidas en el mismo u otro documento*

- ¿Qué es DTD?

- ✓ *Definición de Tipo de Documento*
  - ✓ Es una **descripción de la estructura** y de la **sintaxis** de un documento XML => **gramática**
  - ✓ No se desarrolló a partir de XML => **hereda de SGML** con lo que su sintaxis no es XML, lo cual representa un **problema**.



- ¿Cuál es la función de DTD?

- ✓ Describir la **estructura** de los documentos XML
  - ✓ Mantener la **consistencia** de los documentos XML que utilicen el **mismo DTD**
  - ✓ Permite la **validación** de dichos documentos
  - ✓ Facilita la **compartición** de documentos entre varios usuarios

## Vincular un DTD a un documento XML

- **Formas de declarar un DTD**
- **<!DOCTYPE>**: contiene la definición de la gramática
  - a) **DTD interna**
    - Definir la gramática en el **interior** del documento XML –embebida–
    - Añadir la declaración **<!DOCTYPE>** dentro del **protocolo**, justo **después** de la **declaración**
    - Documento autosuficiente (**standalone=“yes”** )
    - Útil cuando queremos transmitir el documento a un destino en el que no sabemos si se tiene acceso al DTD
    - No se podrá utilizar en diferentes documentos
    - **Sintaxis:**

```
<!DOCTYPE nombreElementoRaiz [ declaraciones ]>
```

- nombreElementoRaiz: nombre del elemento raíz
- declaraciones: lista del conjunto de declaraciones de los elementos y atributos del documento

## Vincular un DTD a un documento XML

- **Formas de declarar un DTD**

- b) **DTD externa**

- Añadir la declaración `<!DOCTYPE>` justo **después** de la *declaración* indicando la ruta y el fichero que contiene la definición de la gramática DTD (**ruta/fichero.dtd**)
    - Documento no autosuficiente (**standalone="no"** )
    - Útil cuando tenemos varios documentos XML almacenados y todos deben cumplir las mismas restricciones definidas en un fichero DTD
    - **Sintaxis DTD externa privada:**

```
<!DOCTYPE nombreElementoRaiz SYSTEM "fichero.dtd">
```

- fichero.dtd: nombre del archivo de la DTD

- **Sintaxis DTD externa con referencia pública:**

```
<!DOCTYPE nombreElementoRaiz PUBLIC "identificación dtd" "url dtd">
```

- **Ej:** <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

## Vincular un DTD a un documento XML

- Ejemplos:

- a) DTD interna

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
    <!ELEMENT address (name,company,phone)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
]>
<address>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</address>
```

- a) DTD externa privada

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</address>
```

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

## Estructura de *DTD*

- **Estructura de un DTD**

- ✓ Define los **elementos** que hay que utilizar y su contenido
- ✓ Define los **atributos** que se pueden emplear en cada elemento con valores válidos
- ✓ Define la **estructura: orden** en el que se van añadiendo los elementos en un documento.
- Ej: observa la siguiente definición de un DTD e intenta entender su significado. ¿Cuántos **elementos** define? ¿En qué **orden**? ¿Cuáles son **opcionales**? ¿Qué elementos contienen una cadena de caracteres?

```
<!ELEMENT lista_de_personas (persona*)>
<!ELEMENT persona (nombre, fechanacimiento?,
                   sexo?, numeroseguridadsocial?)>
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT fechanacimiento (#PCDATA) >
<!ELEMENT sexo (#PCDATA) >
<!ELEMENT numeroseguridadsocial (#PCDATA)>
```

## Estructura de DTD

- Ej: observa los siguientes documentos XML e indica si son válidos con respecto a la definición DTD anterior

```
<lista_de_personas>
  <persona>
    <nombre>Tim Berners-Lee</nombre>
    <fechanacimiento>8 de junio, 1955</fechanacimiento>
  </persona>
</lista_de_personas>
```

```
<lista_de_personas>
  <nombre>Larry Roberts</nombre>
  <nombre>Tim Berners-Lee</nombre>
</lista_de_personas>
```

```
<lista_de_personas>
</lista_de_personas>
```

```
<lista_de_personas>
  <persona>
    <numeroseguridadsocial>3452345435</numeroseguridadsocial>
    <nombre>Tim Berners-Lee</nombre>
    <fechanacimiento>8 de Junio, 1955</fechanacimiento>
    <sexo>masculino</sexo>
  </persona>
</lista_de_personas>
```

```
<lista_de_personas>
  <persona>
    <fechanacimiento>8 de Junio, 1955</fechanacimiento>
    <numeroseguridadsocial>3452345435</numeroseguridadsocial>
  </persona>
</lista_de_personas>
```

## Estructura de *DTD*

- **Estructura de un DTD**

- ✓ Se compone de una **lista de declaraciones** que va entre los símbolos `<!` y `>`

- **Tipos de declaraciones:**

- a) **Elementos (*ELEMENT*)**: define un **elemento válido** y su posible **contenido**
    - b) **Atributos (*ATTLIST*)**: define un **atributo válido** y su posible **contenido**
    - c) **Entidad (*ENTITY*)**: define un nombre abreviado en sustitución de contenidos extensos que se repiten mucho

- Ej: `<!ENTITY nom "Juan Pérez López">`

- d) **Notación (*NOTATION*)**: identifican tipos específicos de datos binarios externos

- Ej: `<!NOTATION GIF87A SYSTEM "GIF">`

## Estructura de DTD: Elementos

- **Tipos de declaraciones:**
  - a) **Elementos (*ELEMENT*)**: define un **elemento válido** y su posible **contenido**
    - **Sintaxis:** `<!ELEMENT nombre contenido>`
      - ✓ **nombre**: hay que cumplir las reglas básicas para crear un nombre de etiqueta *bien formado*.
      - ✓ **contenido**: tipo de contenido del elemento que se está creando
    - **Tipo de contenido:**
      - ✓ **EMPTY**: permite definir elementos sin contenido. Puede tener atributos
      - ✓ **ANY**: permite **cualquier tipo** de contenido –*texto* y *elementos hijos*–. **NO** se recomienda –documentos sin estructuración–
      - ✓ **#PCDATA**: contenido textual
      - ✓ **ELEMENT**: contiene **subelementos**. Especificarlos empleando *modelos de contenidos* o de *grupos* para indicar *qué elementos*, el *orden* y *cuántos*
      - ✓ **MIXED**: Caracteres textuales o mezcla de éstos y subelementos
    - **Ej:** `<!ELEMENT br EMPTY>`    `<!ELEMENT titulo (#PCDATA)>`  
`<!ELEMENT cuerpo (#PCDATA|imagen)*>`

## Estructura de DTD: Elementos

- **Modelos de contenido o de grupo** para indicar qué *elementos*, en qué *orden* y cuántos

- Un **único elemento hijo**: indica que el elemento anidado debe aparecer una única vez. Se escribe directamente entre paréntesis ()

- Ej:

```
<!ELEMENT tutor (nombre)>
<tutor>
  <nombre>Pepe</nombre>
</tutor>
```

- **Varios elementos hijos**:

- ✓ **Lista de tipo secuencia**: indica qué **elementos** deben aparecer y en qué **orden**. Se escriben entre paréntesis y separados por **comas** (elem1, elem2, ...)
- ✓ **Lista de tipo elección**: indica entre qué elementos se puede elegir **uno** de ellos. Dichos elementos se escriben entre **paréntesis** y se separan con la barra | => (elem1 | elem2 | elem3...)

- Ej:

```
<!ELEMENT tutor (nombre, apellidos)>
<tutor>
  <nombre>Pepe</nombre>
  <apellidos>Pérez</apellidos>
</tutor>
```

```
<!ELEMENT tutor ((nombre|apodo), apellidos)>
<tutor>
  <nombre>Pepe</nombre>
  <apellidos>Pérez</apellidos>
</tutor>
```

## Estructura de DTD: Elementos

### ■ Modelos de contenido o de grupo

- **Cuantificador o Indicador de Frecuencia:** indica el **nº de veces** que de forma sucesiva puede aparecer un elemento.
- Situarlo a continuación del identificador del elemento, sin añadir espacios en blanco.
- Cualquier elemento que **no lleve** cuantificador, debe aparecer **1 vez** en la posición indicada.
- ✓ ?: para componentes **opcionales**, puede incluirse o no (**0 o 1 vez**)
- ✓ +: para componentes **obligatorios**, deben incluirse al menos 1 vez (**1 o más**)
- ✓ \*: para componentes que se pueden repetir o no, sin importar el **nº de veces** que se repiten (**0 o más**)

#### ■ Ej:

```
<!ELEMENT telefono (movil, fijo?)>  
<telefono><movil>699999999</movil></telefono>
```

```
<!ELEMENT provincia (nombre, (cp,poblacion)+)>  
<provincia><nombre>Lugo</nombre>  
<cp>27003</cp> <poblacion>Lugo</poblacion>  
<cp>27850</cp><poblacion>Viveiro</poblacion></provincia>
```

```
<!ELEMENT maquinas (ip, nombremaquina)*>  
<maquinas></maquinas>
```

## Estructura de DTD : Elementos

- **Modelos de contenido o de grupo**

- Ej: observa las siguientes declaraciones e indica si el ejemplo de documento XML que la acompaña se corresponde o no con dichas declaraciones

```
<!ELEMENT algo (a, b?, c+, d*)>

<algo>
<a /><c /><c></c><d></d><d />
</algo>
```

```
<!ELEMENT algo ((a | b), c?)>

<algo>
<a /><b />
<c></c>
</algo>
```

- ¿y estas que son un poco más complejas? Se entenderán mejor más adelante

```
<!ELEMENT algo ((a | b)+, c?)>

<algo>
<a /><b /><a /><a /><b /><c />
</algo>
```

```
<!ELEMENT algo ((a, b)+, (c | d)+)>

<algo>
<a /><b /><a /><c /><d />
</algo>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio6:** Define la declaración DTD **interna** para documentos XML que almacenen la información relacionada sobre mensajes de correo electrónico
- Para cada **correo electrónico** deberá aparecer:
    - ✓ **remitente**
    - ✓ **destinatario** (uno o más)
    - ✓ **asunto**
    - ✓ **cuerpo** (uno o ninguno)
    - ✓ **adjunto** (uno, varios o ninguno)
  - Una vez que hayas terminado de crear el DTD, añade información y comprueba si se visualiza jerárquicamente desde el navegador.

- **Ej:**

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<!DOCTYPE emails>
- <emails>
  - <email>
    <de>maria@gmail.com</de>
    <para>ignacio@gmail.com</para>
    <para>pedro@edu.xunta.es</para>
    <asunto>Ahí tenéis los archivos</asunto>
    <adjunto>D:\fotos\dePesca.jpg</adjunto>
    <adjunto>D:\fotos\enLaCena.jpg</adjunto>
  </email>
</emails>
```

## Ejercicio de documento XML con declaración DTD

### ➤ Sol. Ej6:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE emails [
    <!ELEMENT emails (email*)>
    <!ELEMENT email (de, para+, asunto, cuerpo?, adjunto*)>
    <!ELEMENT de (#PCDATA)>
    <!ELEMENT para (#PCDATA)>
    <!ELEMENT asunto (#PCDATA)>
    <!ELEMENT cuerpo (#PCDATA)>
    <!ELEMENT adjunto (#PCDATA)>
]>
<emails>
    <email>
        <de>maria@gmail.com</de>
        <para>ignacio@gmail.com</para>
        <para>pedro@edu.xunta.es</para>
        <asunto>Ahi tenéis los archivos</asunto>
        <adjunto>D:\fotos\dePesca.jpg</adjunto>
        <adjunto>D:\fotos\enLaCena.jpg</adjunto>
    </email>
</emails>
```

## DTD combinando opciones y cuantificadores

- **DTD combinando opciones y cuantificadores**
- Vamos a comprobar cómo funcionan las reglas que involucran **estructuras complejas**, por ejemplo, si queremos **combinar** los diferentes **cuantificadores** con listas de **opciones**.
  - **Caso1 -secuencia seguida de otra secuencia-**: crear una **lista de ventas** en el que se vean de forma consecutiva todos los posibles elementos *ventapc* seguidos de todos los posibles elementos *ventamonitor*:
  - **Ej:**

```
<listaventas>
    <ventapc>100</ventapc>
    <ventapc>300</ventapc>
    <ventapc>400</ventapc>
    <ventamonitor>200</ventamonitor>
    <ventamonitor>400</ventamonitor>
    <ventamonitor>500</ventamonitor>
</listaventas>
```
  - **Sol:** DTD que se adapta a la descripción anterior

```
<!ELEMENT listaventas (ventapc+,ventamonitor+)>
<!ELEMENT ventapc      (#PCDATA)>
<!ELEMENT ventamonitor (#PCDATA)>
```

## Ejercicio DTD con estructura compleja

- **Ejercicio7:** Vamos a modificar ligeramente el planteamiento anterior para almacenar los elementos *ventapc* y *ventamonitor*. **Crea el DTD** de cada caso.

- **Caso2 –se admite cualquier orden y repetición en los elementos-:** en la **lista de ventas** se permite cualquier orden y combinación en los posibles elementos *ventapc* y *ventamonitor*:
  - **Ej:** existen varias posibilidades de combinar los elementos

```
<listaventas>
    <ventapc>100</ventapc>
    <ventapc>300</ventapc>
    <ventapc>400</ventapc>
    <ventamonitor>200</ventamonitor>
    <ventamonitor>400</ventamonitor>
    <ventamonitor>500</ventamonitor>
</listaventas>
```

```
<listaventas>
    <ventapc>100</ventapc>
    <ventamonitor>200</ventamonitor>
    <ventapc>300</ventapc>
    <ventapc>400</ventapc>
    <ventamonitor>400</ventamonitor>
</listaventas>
```

- **Caso3 –se admite como secuencia válida la pareja de elementos-:** en la lista de ventas se admite el par *ventapc-ventamonitor*

- **Ej:**

```
<listaventas>
    <ventapc>100</ventapc>
    <ventamonitor>400</ventamonitor>
    <ventapc>100</ventapc>
    <ventamonitor>400</ventamonitor>
    <ventapc>100</ventapc>
    <ventamonitor>400</ventamonitor>
</listaventas>
```

## Ejercicio DTD con estructura compleja

- Vamos a modificar ligeramente el planteamiento anterior para almacenar los elementos *ventapc*, *ventamonitor* y *comprapc*, *compramonitor*
  - **Caso4 –se admite cualquier orden de los elementos, pero sólo un tipo de elementos: o compras o ventas, no ambos-**: en el elemento raíz **movimientos** se admiten uno o varios elementos de ventas *ventapc-ventamonitor* o uno o varios elementos de compras *comprapc-compramonitor* , pero no la mezcla.
  - **Ej:** existen varias posibilidades

```
<movimientos>
    <ventamonitor>200</ventamonitor>
    <ventapc>800</ventapc>
    <ventamonitor>800</ventamonitor>
    <ventamonitor>300</ventamonitor>
</movimientos>
```

```
<movimientos>
    <comprapc>400</comprapc>
    <comprapc>400</comprapc>
    <compramonitor>400</compramonitor>
    <compramonitor>400</compramonitor>
</movimientos>
```

- Tras haber creado los DTD correctos, añade datos en tu documento XML en diferente orden y repetición y comprueba su **validez** desde **XML Copy Editor**

## Ejercicio DTD con estructura compleja

### ➤ Soluciones:

- **Sol. Caso2** –se admite cualquier orden en los elementos-:

```
<!ELEMENT listaventas (ventapc|ventamonitor)+>
<!ELEMENT ventapc      (#PCDATA)>
<!ELEMENT ventamonitor (#PCDATA)>
```

- **Sol. Caso3** –se admiten en pares-:

```
<!ELEMENT listaventas (ventapc,ventamonitor)+>
<!ELEMENT ventapc      (#PCDATA)>
<!ELEMENT ventamonitor (#PCDATA)>
```

- **Sol. Caso4** –se admiten o compras ventas-:

```
<!DOCTYPE movimientos [
  <!ELEMENT movimientos ((ventapc|ventamonitor)+ | (comprapc|compramonitor)+ )>
  <!ELEMENT ventapc      (#PCDATA)>
  <!ELEMENT ventamonitor (#PCDATA)>
  <!ELEMENT comprapc     (#PCDATA)>
  <!ELEMENT compramonitor (#PCDATA)>
]>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio8:** Define la declaración DTD **interna** para documentos XML que almacenen la información relacionada con la fabricación de ratones y teclados.

- Requisitos:
  - ✓ La raíz es **productos**
  - ✓ Dentro de **productos** puede haber **<raton>** o **<teclado>** que pueden **repetirse** e ir en cualquier **orden**
  - ✓ Todo **<raton>** tiene siempre un **<codigo>** y puede o no tener una **<descripción>**.
  - ✓ Todo **<teclado>** tiene siempre un **<codigo>**, debe llevar siempre una **<descripcion>** y puede o no tener un **<peso>**
- Una vez que hayas terminado de crear el DTD, añade información y comprueba su validación.
- **Ej:**

```
<productos>
    <teclado>
        <codigo>T1</codigo>
        <descripcion>Teclado inalamb.</descripcion>
    </teclado>
</productos>
```

## Ejercicio de documento XML con declaración DTD

### ➤ Sol. Ej8:

```
<!ELEMENT productos (raton|teclado)* >
<!ELEMENT raton (codigo, descripcion?) >
<!ELEMENT codigo (#PCDATA) >
<!ELEMENT descripcion (#PCDATA) >
<!ELEMENT teclado (codigo,descripcion,peso?)>
<!ELEMENT peso (#PCDATA) >
```

```
<productos>
  <raton>
    <codigo>R1</codigo>
  </raton>
  <teclado>
    <codigo>T1</codigo>
    <descripcion>Teclado inalamb.</descripcion>
    <peso>0.5</peso>
  </teclado>
</productos>
```

- Comprueba: ¿cómo harías la declaración DTD del elemento **<p>** si dentro puede contener texto y elementos hijos como **<strong>** y **<em>**? ¿Qué tipo de contenido define dicho elemento?

## Estructura de DTD: Atributos

- **Tipos de declaraciones:**

- b) **Atributos (ATTLIST)**: define un **atributo válido** y su posible **contenido**

- La etiqueta **ATTLIST** se utiliza para declarar atributos en el **DTD**

- **Características de atributos:**

- ✓ Proporcionan **información adicional** sobre los elementos
    - ✓ Permiten hacer cosas que los elementos por sí solos no pueden
    - ✓ **No** pueden contener **subatributos**
    - ✓ Los elementos pueden tener cero, uno o varios atributos
  - Se pueden emplear para:
    - Definir un **valor por defecto**
    - Definir un **conjunto de valores válidos**
    - Definir valores **fijos** (constantes)
    - Crear **referencias** entre distintos elementos

## Estructura de DTD: Atributos

- Tipos de declaraciones:
  - b) **Atributos (ATTLIST)**: define un **atributo válido** y su posible **contenido**
  - Sintaxis: `<!ATTLIST ElementName AttrName Type Default>`
    - ✓ **elementName**: elemento que tiene dicho atributo
    - ✓ **attrName**: nombre del atributo que se está definiendo
    - ✓ **type**: tipo de atributo o contenido a asignar
    - ✓ **default**: valor por defecto del atributo. También puede indicar si es obligatorio o no
  - Sintaxis si declaramos varios atributos sobre el mismo elemento:

```
<!ATTLIST ElementName  
          AttrName Type Default  
          AttrName Type Default  
          ...  
          >
```

- Ej: `<!ATTLIST artista nombre CDATA #REQUIRED>`

## Estructura de DTD: Atributos

### b) Atributos (**ATTLIST**):

- **Tipos de atributo:** referido al contenido a asignar a los atributos; incluso puede ser un valor entre los que se ofrecen en una lista de valores

#### 1) CDATA, NMTOKEN y NMTOKENS:

- **CDATA** (*characters data*)
  - Tipo más habitual para asignar texto
  - Contiene una cadena de texto con cualquier símbolo/carácter
- **NMTOKEN** (*name token*)
  - Parecidos a CDATA, sólo aceptan los **caracteres permitidos por XML** (letras, números, puntos, guiones, subrayados y dos puntos), **sin espacios en blanco ni símbolos de <, >**, etc
  - Ej: **<!ATTLIST nota fecha NMTOKEN #REQUIRED>**  
**<nota fecha="22-02-16">**
- **NMTOKENS** (*lista de NMTOKEN*)
  - Pueden contener una o varias palabras con los caracteres permitidos por NMTOKEN separadas por **espacios en blanco**

## Estructura de DTD: Atributos

### b) Atributos (**ATTLIST**):

- **Tipos de atributo:**

#### 2) Enumeraciones o lista de valores enumerados:

- Para aquellos atributos que sólo pueden tomar un valor de entre los valores indicados en una lista
- La lista se especifica entre paréntesis y los valores se separan con |
- Ej: <!ATTLIST curso nivel (bajo | medio | alto) “bajo”> <curso nivel=“alto”>

#### 3) ID, IDREF o IDREFS:

- **ID:** para **identificar** al elemento al que está asociado.
  - Debe ser un valor único en el documento y comenzar por una letra o guión bajo
- **IDREF:** referencia a un id de otro elemento
  - Para que sea válido debe existir ese ID en otro elemento del mismo documento
- **IDREFS:** lista de referencias a id's
  - Representa múltiples ID de otros elementos, separados por espacios

## Estructura de DTD: Atributos

### b) Atributos (**ATTLIST**):

- **Tipos de atributo:**

### 3) ID, IDREF o IDREFS:

- **Ej1:** definir los atributos adecuados para el elemento *empleado* de forma que cada empleado quede identificado por su propio código y se le asigne el código que identifica a cada jefe

```
<!ATTLIST empleado
          idEmple ID #REQUIRED
          idJefe IDREF #IMPLIED>

<empleado idEmple="e3" idJefe="j10">
```

- **Ej2:** **<!ATTLIST persona**  
                 **nif ID #REQUIRED**  
                 **contacto IDREF #IMPLIED>**

```
<persona nif="W34343460" contacto="R65434567">María Silva</persona>
<persona nif="R65434567">Juan Pérez</persona>
```

## Estructura de DTD: Atributos

### b) Atributos (**ATTLIST**):

- **Valores Default:** referido al valor predeterminado del atributo o a la obligatoriedad o no de que se defina

#### 1) “valor”: valor predeterminado del atributo

- En caso de que no se indique el atributo en el elemento en el que está definido, el valor que se le asigna es su valor predeterminado
- Ej1: <!ATTLIST curso nivel CDATA “bajo”> <curso nivel=“alto”>...
- Ej2: <!ATTLIST persona ciudad CDATA “Vigo”> <persona>...</persona>

#### 2) #IMPLIED, #REQUIRED y #FIXED:

- **#IMPLIED:** para atributos **opcionales**. **NO** se le puede asignar un valor predeterminado
  - Ej: <!ATTLIST curso nivel NMTOKEN #IMPLIED> <curso>...
- **#REQUIRED:** para atributos **obligatorios**. **Sin** valor predeterminado.
  - Ej: <!ATTLIST curso nivel CDATA #REQUIRED> <curso nivel=“alto”>...
- **#FIXED “valor”:** para asignar un **valor fijo** a un atributo
  - Ej: <!ATTLIST ciudad codigo CDATA #FIXED “36200”> <ciudad>...</ciudad>

## Ejercicio de documento XML con declaración DTD

- **Ejercicio9:** Define la declaración DTD **externa** para almacenar la información sobre facturas. Los requisitos son los siguientes:
- ✓ Los ficheros deben tener un elemento **<listafacturas>**
  - ✓ Dentro de la lista debe haber una o más **facturas**.
  - ✓ Las facturas tienen un atributo **fecha** que es optativo.
  - ✓ Toda factura tiene un **emisor**, que es un elemento obligatorio y que debe tener un atributo **cif** que es obligatorio. Dentro de emisor debe haber un elemento **nombre**, que es obligatorio y puede o no haber un elemento **volumenventas**.
  - ✓ Toda factura debe tener un elemento **pagador**, el cual tiene exactamente la misma estructura que **emisor**.
  - ✓ Toda factura tiene un elemento **importe**.
- Crea el **DTD** desde un validador XML, eligiendo la extensión adecuada para el fichero. ¿Se añade alguna información automáticamente?
  - Tras haber creado el DTD, crea un documento XML con un par de facturas y comprueba si **es válido**.

## Ejercicio de documento XML con declaración DTD

➤ Sol. Ej9:

```
<!ELEMENT ·listafacturas·(factura+)>
<!ELEMENT ·factura·(emisor,·pagador,·importe)>
<!ATTLIST ·factura·fecha CDATA ·#IMPLIED>
<!ELEMENT ·emisor·(nombre,·volumenventas?)>
<!ELEMENT ·nombre·(#PCDATA)>
<!ATTLIST ·emisor·cif CDATA ·#REQUIRED>
<!ELEMENT ·volumenventas·(#PCDATA)>
<!ELEMENT ·pagador·(nombre,·volumenventas?)>
<!ATTLIST ·pagador·cif CDATA ·#REQUIRED>
<!ELEMENT ·importe·(#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE ·listafacturas· SYSTEM "listafacturas.dtd">
```

```
<listafacturas>
  <factura fecha="11-2-2020">
    <emisor cif="34523553">
      <nombre>Cnes. Conde, S.L.</nombre>
    </emisor>
    <pagador cif="36987987y">
      <nombre>Talleres Almart, S.L.</nombre>
      <volumenventas>20</volumenventas>
    </pagador>
    <importe>2500</importe>
  </factura>
</listafacturas>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio10\_a:** Corrige el siguiente **DTD** para que el XML sea correcto.
- **Pista:** la información a almacenar se refiere a la sintaxis de un lenguaje de programación donde no es obligatorio indicar el tipo de dato de las **variables**.
- Tras haber corregido el DTD, añade más datos al documento XML y comprueba si es válido.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE declaraciones [
    <!ELEMENT declaraciones (constante | tipo_de_dato | variable)>
    <!ELEMENT constante (#PCDATA)>
        <!ATTLIST constante valor CDATA #REQUIRED>
    <!ELEMENT tipo_de_dato (#PCDATA)>
    <!ELEMENT variable (#PCDATA)>
        <!ATTLIST variable tipo CDATA #REQUIRED>
]>
<declaraciones>
    <constante valor="3.141592">PI</constante>
    <variable tipo="real">radio</variable>
    <variable>area</variable>
    <variable>longitud</variable>
</declaraciones>
```

➤ Sol. Ej10\_a:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<!DOCTYPE declaraciones [
  <!ELEMENT declaraciones (constante|tipo_de_dato|variable)+>
  <!ELEMENT constante (#PCDATA)>
    <!ATTLIST constante valor CDATA #REQUIRED>
  <!ELEMENT tipo_de_dato (#PCDATA)>
  <!ELEMENT variable (#PCDATA)>
    <!ATTLIST variable tipo CDATA #IMPLIED>
]>

<declaraciones>
  <constante valor="3.141592">Pi</constante>
  <variable tipo="real">radio</variable>
  <variable>area</variable>
  <variable>longitud</variable>
</declaraciones>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio10\_b:** Corrige el siguiente documento XML para que sea válido con respecto a la definición del DTD **externo**.
- ¿Qué implicaciones tiene un atributo de tipo **ID** en vez de **CDATA**?

```
!ELEMENT agenda (persona)+  
<!ELEMENT persona (nombre, telefono)>  
  <!ATTLIST persona  
    | | | | | codigo ID #REQUIRED  
    | | | | | tipo (trabajo | gimnasio | vecino) "trabajo">  
<!ELEMENT nombre (#PCDATA)>  
<!ELEMENT telefono (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!DOCTYPE agenda SYSTEM "agenda.dtd">  
  
<agenda>  
  <persona codigo="1" tipo="trabajo">  
    <telefono>986234523</telefono>  
    <telefono>669834520</telefono>  
  </persona>  
  <persona codigo="1" tipo="gimnasio">  
    <nombre>María</nombre>  
    <telefono>916565433</telefono>  
  </persona>  
  <persona codigo="2" tipo="vecino">  
    <nombre>Juan García</nombre>  
    <telefono>971234525</telefono>  
    <telefono>663234525</telefono>  
  </persona>  
</agenda>
```

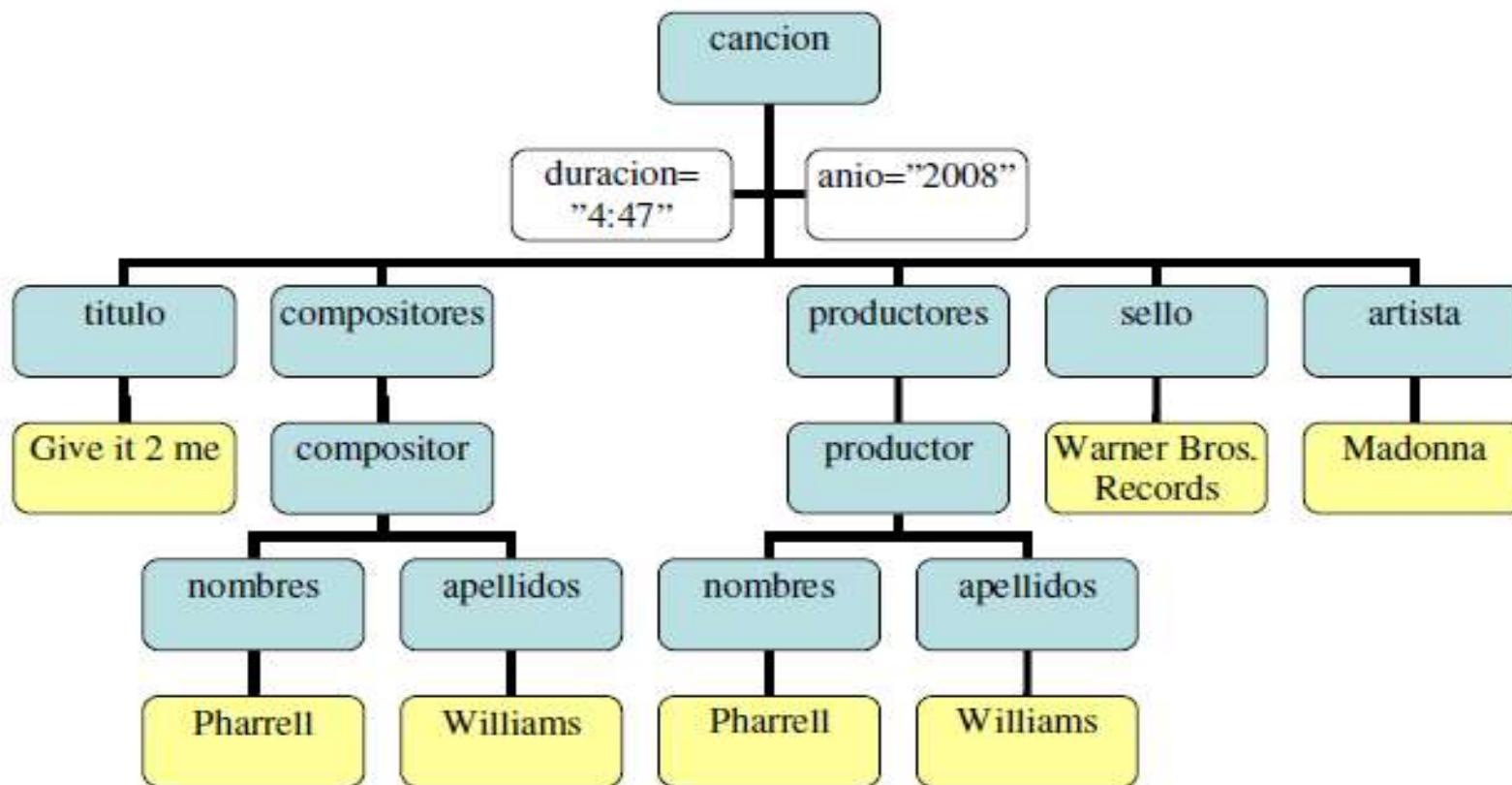
## Ejercicio de documento XML con declaración DTD

- **Ejercicio10\_c:** Crea el DTD adecuado para que el XML que se proporciona sea válido.

```
<lista_de_personas>
    <persona nacionalidad="español">
        <nombre>José García</nombre>
        <fechanacimiento>25/04/1984</fechanacimiento>
        <telefono>670201256</telefono>
        <telefono>968472898</telefono>
        <sexo>Varón</sexo>
    </persona>
    <persona nacionalidad="español">
        <nombre>Juan Antonio</nombre>
        <fechanacimiento>25/04/1984</fechanacimiento>
        <telefono>670201256</telefono>
        <numeroseguridadsocial>2569587412569</numeroseguridadsocial>
    </persona>
</lista_de_personas>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio11:** Define la declaración DTD **externa** para almacenar información sobre **canciones**. Los requisitos que debe cumplir vienen reflejados en el siguiente organigrama:
  - Tras haber creado el DTD, crea un documento XML con los datos que se indican y comprueba si **es válido**.



➤ Sol. Ej11:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE canciones SYSTEM "canciones.dtd">
<canciones>
  <cancion duracion="4:47" año="2008">
    <titulo>Give it 2 me</titulo>
    <compositores>
      <compositor>
        <nombre>Pharrel</nombre>
        <apellidos>Williams</apellidos>
      </compositor>
    </compositores>
    <productores>
      <productor>
        <nombre>Pharrel</nombre>
        <apellidos>Williams</apellidos>
      </productor>
    </productores>
    <sello>Warner Bros. Record</sello>
    <artista>Madonna</artista>
  </cancion>
</canciones>
```

```
<!ELEMENT canciones (cancion+)>
<!ELEMENT cancion (titulo, compositores, productores, sello, artista+)>
<!ATTLIST cancion duracion CDATA #REQUIRED>
<!ATTLIST cancion año CDATA #REQUIRED>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT compositores (compositor+)>
<!ELEMENT compositor (nombre, apellidos)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT productores (productor+)>
<!ELEMENT productor (nombre, apellidos)>
<!ELEMENT sello (#PCDATA)>
<!ELEMENT artista (#PCDATA)>
```

## Ejercicio de documento XML con declaración DTD

- **Ejercicio12:** Crea el DTD con las definiciones adecuadas para que se ajusten lo más posible al siguiente documento XML y que sea válido.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE cursos SYSTEM "cursos.dtd">
<cursos>
    <empresa>Professional Training, S.A.</empresa>
    <curso codigo="C89" nombre="Java" fechaInicio="7/10-2019" nivel="avanzado" />
    <curso codigo="C90" nombre ="PHP" fechaInicio ="6/11-2019" />
    <curso codigo="C100" nombre ="XML" fechaInicio ="30/03/2020" nivel="medio" />
    <alumnos>
        <alumno cursos="C90">
            <nombre>Isabel</nombre>
            <apellidos>Díaz Ayuso</apellidos>
        </alumno>
        <alumno cursos="C89 C90 C100">
            <nombre>Rocío</nombre>
            <apellidos>Monasterio San Martín</apellidos>
        </alumno>
        <alumno>
            <nombre>Pablo</nombre>
            <apellidos>Iglesias Turrión</apellidos>
            <telefono>698811111</telefono>
        </alumno>
        <alumno cursos="C100 C89">
            <nombre>Mónica</nombre>
            <apellidos>García Gómez</apellidos>
            <telefono>698811111</telefono>
        </alumno>
    </alumnos>
</cursos>
```

➤ Sol. Ej12:

```
<!ELEMENT cursos (empresa, curso*, alumnos)>
<!ELEMENT empresa (#PCDATA)>
<!ELEMENT curso EMPTY>
<!ATTLIST curso
  codigo ID #REQUIRED
  nombre CDATA #REQUIRED
  fechainicio CDATA #REQUIRED
  nivel (bajo|medio|avanzado) #IMPLIED
  >
<!ELEMENT alumnos (alumno*)>
<!ELEMENT alumno (nombre, apellidos, telefono?)>
  <!ATTLIST alumno cursos IDREFS #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellidos (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

## Limitaciones de los DTD's

- **Limitaciones de los DTD's:**
  - ✓ **Esquema cerrado** (closed): **no** se permite una reutilización ni una ampliación del documento mediante elementos no definidos => **no es extensible**
  - ✓ **No se basa en XML**: los **parsers** no pueden comprobar si están bien formados ya que no utiliza la sintaxis XML
  - ✓ Al **no soportar tipado** de elementos, **No** se pueden definir declaraciones globales ni fijar restricciones sobre los **tipos de datos** de elementos y atributos: tipo de datos, tamaño, etc
  - ✓ **No soporta los espacios de nombres (XMLNS)**
  - ✓ **No se pueden enumerar** los posibles valores de los **elementos**
  - ✓ Los **elementos no** pueden tener **valores por defecto**
  - ✓ **Control limitado** sobre las **cardinalidades** de los elementos (0, 1,  $\infty$ )
    - **Indica:** ¿cómo se definen los elementos que deben almacenar un nº de expediente, una fecha, una altura, un peso, o incluso el indicar si se cumple o no un determinado requisito? ¿qué implica dicha definición de **tipo**? ¿los **atributos** tienen las mismas restricciones?