



IES
Puerto
de la
Cruz
Telesforo Bravo

CICLO FORMATIVO DE GRADO SUPERIOR:

“DESARROLLO DE APLICACIONES WEB”



MÓDULO:

**Lenguajes de marcas y
sistemas de gestión de
información**

(4 horas semanales)





Índice

TEMA 5.- XML.....	3
1.- INTRODUCCIÓN AL XML.....	3
1.1.- Características.....	4
1.2.- Documentos xml.....	4
1.3.- Etiquetas xml.....	4
1.4.- Sintaxis de escritura XML.....	5
1.5.- Análisis de documentos XML.....	6
2.- ELEMENTOS DEL LENGUAJE.....	7
3.- EL PRÓLOGO DE LOS DOCUMENTOS XML.....	9
4.- LOS DTDs.....	9
4.1.- Definición de elementos.....	11
4.2.- Elementos hijo.....	12
4.3.- Definición de atributos.....	19
4.4.- Definición de entidades generales.....	22
4.5.- Definición de entidades de parámetro.....	25
5.- HERRAMIENTAS.....	26



TEMA 5.- XML.

1.- Introducción al XML.

XML son las siglas de Extensible Mark-up Language. Se trata de un lenguaje formado a partir de la sintaxis y reglas del SGML (Estandar Generalised Mark-up Language), simplificado y adaptado a la estructura de la información.

La característica fundamental del XML es que más que un lenguaje en sí es un **metalenguaje**, esto es, un lenguaje que nos permite definir nuevos lenguajes adaptados a contextos muy diversos. Por lo tanto difiere del HTML cuyo principal objetivo era la creación de páginas web.

Uno de los lenguajes que se han creado con XML es el XHTML que ya vimos en el tema 3, y que fusiona los principios del XML con las etiquetas del HTML.

Como definición formal: **XML** es una especificación para diseñar lenguajes de marcas, que permite **definir etiquetas personalizadas**, y emplearlas tanto para describir como organizar y tratar información.



La diferencia principal es que el XML nos permite crear nuestras propias etiquetas. Además se va a encargar de la estructura de la información. Con esta relación de estructura los datos podrán ser almacenados, transmitidos, procesados y mostrados por aplicaciones y dispositivos de índole muy diversa, con lo cual se tratará de información **independiente de la plataforma**. Y esta es una de las ideas que impulsó el trabajo en este lenguaje, conseguir un estándar libre que permitiera que plataformas distintas compartieran e intercambiaran información.

A diferencia del HTML el XML es mucho más estricto con los chequeos y las validaciones. Es más abierto y flexible. Será el lenguaje de intercambio de información entre cualquier tipo de dispositivos, ya que se pueden construir analizadores con facilidad.

1.1.- Características.

- * Fácilmente procesable: basta una lectura para generar un árbol de estructura.
- * Separa radicalmente el contenido y el formato de presentación.
- * Diseñado para cualquier lenguaje y alfabeto.
- * Extensibilidad: se han creado muchas aplicaciones que utilizan XML.
- * Validación: permite determinar qué documentos están bien formados y cuales además son documentos válidos.
- * Basado en texto.

1.2.- Documentos xml.

* Son ficheros de texto con extensión .xml. Contienen conjuntos de datos con sus correspondientes etiquetas de marcas. Pueden contener información estructurada de cualquier ámbito.

Los documentos XML:

- Guardan únicamente información.
- No contienen aspectos de presentación.
- Se forman con datos y con etiquetas de marcado.

Ejemplo:

Prologo	<pre><?xml version="1.0" encoding="ISO-8859-1" standalone="no"?> <!DOCTYPE persona SYSTEM "persona.dtd"</pre>
Cuerpo	<pre><persona> <nombre>Luis</nombre> <apellidos>Pérez</apellidos> </persona></pre>

Como ves en **xml** crearemos nuestras propias etiquetas. Éstas siguen una estructura similar a las etiquetas en html:

1.3.- Etiquetas xml.



Pueden existir etiquetas vacías como en xhtml, y el cierre es similar. Ejemplo: `<fax />`

Veamos un fichero de ejemplo de documento XML para la gestión de un videoclub:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE peliculas SYSTEM "Peliculas.dtd">

<peliculas>
  <pelicula tipo="comedia" sistema="PG-13" ejemplares="5" año="1987">
    <titulo>Raising Arizona</titulo>
    <guionista>Ethan Coen</guionista>
    <guionista>Joel Coen</guionista>
    <productor>Ethan Coen</productor>
    <director>Joel Coen</director>
    <actor>Nicolas Cage</actor>
    <actor>Holly Hunter</actor>
    <actor>John Goodman</actor>
    <comentarios>Una pelicula clasica de Comedia.</comentarios>
  </pelicula>

  <pelicula tipo="ciencia-ficcion" sistema="PG-13" ejemplares="4" año="1989">
    <titulo>The Abyss</titulo>
    <guionista>James Cameron</guionista>
    <productor>Gale Anne Hurd</productor>
    <director>James Cameron</director>
    <actor>Ed Harris</actor>
    <actor>Mary Elizabeth Mastrantonio</actor>
    <comentarios>Una buena pelicula.</comentarios>
  </pelicula>
</peliculas>
```



PRACTICA 01.- Crea un fichero xml para almacenar libros en venta en una librería. Comienza el fichero con la línea `<?xml version=1.0">`.

La librería se compone de libros. Cada libro tiene un autor, un título y una editorial.

Abre el fichero en Internet Explorer. ¿Qué se muestra?

1.4.- Sintaxis de escritura XML.

La sintaxis de xml es dictada por el w3c. Se trata de un estándar abierto, no pertenece a ningún propietario. Actualmente estamos utilizando la versión 1.0 del lenguaje, aprobada en el 2000. Es necesario seguir las normas siguientes:

- El xml es case-sensitive. Por lo tanto hay que tener cuidado con las minúsculas y mayúsculas. Escribiremos todas las etiquetas y sus atributos en minúsculas. Recuerda las reglas o restricciones del xhtml. Todas esas reglas son normas de escritura del xml.
- Toda etiqueta tiene que tener su correspondiente etiqueta de cierre, o bien ser una etiqueta vacía.
- Es muy importante respetar el orden de anidación de las etiquetas.
- En todos los documentos xml existe una etiqueta denominada **raíz del documento**. Todas las demás etiquetas se encerrarán dentro de la raíz. En el caso del fichero mostrado es la etiqueta `<peliculas>`.
- Todos los valores de los atributos van entre comillas.

Aparte de las reglas de sintaxis que hemos visto existen una serie de recomendaciones aceptadas por la mayoría de los programadores. A estas recomendaciones se les llama **normas de buena construcción**, y por supuesto no son obligatorias:

- En los textos la primera letra de los nombres propios y las frases se escribirá en mayúsculas.
- Los nombres compuestos se escribirán juntos o separados por guión bajo: Saca_corchos
- Las etiquetas han de comenzar por un carácter que sea una letra o bien un “_”, pero no por un número. No se permite el espacio en blanco ni los dos puntos.
- Las etiquetas se centrarán en la información, **nunca** en la presentación:
 - Un mal ejemplo sería el siguiente:

```
<Poema> <Negrita>El reino perdido</Negrita>
<Cursiva>Las huestes de don Rodrigo desmayan y <Negrita> huían
</Negrita> ..... <Cursiva> .....
..... </Poema>
```

- Un buen ejemplo sería el siguiente:

```
<Poema> <Título>El reino perdido</Título>
<Cuerpo>Las huestes de don rodrigo desmayan y huían.....
<Cuerpo> ..... <Comentario> buen poema de.. <Comentario>
..... </Poema>
```

1.5.- Análisis de documentos XML.

Se utiliza un analizador xml, que es un programa de software que lee el documento y es capaz de realizar alguna o todas de las siguientes tareas:



Tareas del analizador XML:

- Determinar si es un documento bien formado (well-formed).
- Determinar si es un documento válido (valid).
- Generar el árbol de análisis.



PRACTICA 02.- Instala el editor de EditiX en su versión gratuita y abre el fichero de la práctica 01.

Valida que el código xml esté bien formado.

¿Puedes observar el árbol de análisis?

Un documento está **bien formado** (well-formed) si está escrito siguiendo las reglas del w3c para los documentos xml. Además contiene uno o más elementos, de entre los que sólo uno es el elemento raíz. Si el documento consta de varias partes, todas están bien formadas. Además no se encuentran caracteres prohibidos en el texto.

El documento puede estar bien formado pero no tener un sentido gramatical. Para eso se utilizan tecnologías que veremos como los DTD o los XML-Schema, que determinan qué información puede guardar un documento concreto. Si el documento, aparte de estar bien formado cumple con las especificaciones semánticas descritas en un DTD o XML-Schema se dice que es un documento **válido**.

El **árbol de análisis** es la representación en árbol jerárquico de la información de un documento xml. Podrá ser utilizado por multitud de tecnologías asociadas al XML para realizar tareas diversas: modificar información, buscar y acceder elementos concretos, intercambiar información con otros formatos, traducir a otros documentos, etc. Ya hablaremos de estas tecnologías más adelante, por el momento sólo decir que es generado por un analizador XML, que en este caso se suele llamar **parser**.



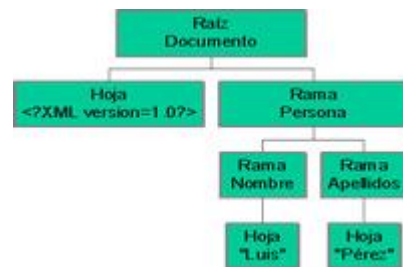
El árbol de análisis será una estructura que muestra los objetos que forman el documento y las relaciones entre ellos. A los componentes de un documento se les llama **objetos** (elementos, comentarios y cadenas de texto). El propio documento es un objeto.

A cada objeto del árbol se le denomina **nodo**. El nodo principal que contiene a los demás se le llama nodo **raíz**.

Cuando un nodo contiene a otro se le denomina **rama**. Los nodos finales, que no contienen otros nodos, se llaman **hojas**.

Veamos un ejemplo:

```
<?xml version="1.0"?>
<empleados><persona>
  <nombre>Luis</nombre>
  <apellidos>Pérez</apellidos>
</persona></empleados>
```



2.- Elementos del lenguaje.

2.1.- Los comentarios:

<!-- Esto es un comentario, y no puedo incluir un doble guión dentro-->

2.2.- Las instrucciones de procesamiento: comienzan por un <?. Y son órdenes del xml. Por ejemplo utilizamos una instrucción para indicar la versión del xml utilizada. Observa en el ejemplo anterior la primera línea. También podremos utilizar estas instrucciones para cargar hojas de estilo CSS:

<? Instrucción ?>

La instrucción no puede incluir los caracteres ?>

2.3.- Las Secciones CDATA: sirven para colocar textos que no queremos que sean analizados como contenido xml. En el ejemplo siguiente hemos colocado en negrita lo que hace falta para definir una sección CDATA:

```
<![CDATA[Este texto no será tratado, puede incluir
  "cualquier" &carácter como los menores y mayores que < >,
  las dobles comillas, el ampersand, etc.
]]>
```

– Estas secciones no serán tratadas por ningún analizador (parser). Además pueden contener cualquier carácter prohibido en el resto del código como comillas (“, ’), ampersand &, relacionales (>, <), etc. Eso sí, no podrán incluir la cadena]]> puesto que se interpretaría como el final de la sección.

2.4.- Las etiquetas y sus textos asociados: son, como sabes, el grueso de los documentos xml. Recuerda que las etiquetas deben comenzar por una letra o por un

carácter subrayado. El segundo y resto de caracteres del nombre de la etiqueta ya podrán incluir números, “-“ o incluso puntos: el resto de caracteres está prohibido. Nunca una etiqueta debe comenzar por la palabra xml.

Al conjunto de una etiqueta incluyendo su texto de marcado y su cierre correspondiente se le llama **ELEMENTO**.



Recuerda que en los elementos hay una serie de caracteres prohibidos como son las comillas simples y dobles, los ampersand y los mayor que – menor que. Si necesitamos incluir estos caracteres tendremos que incluir en el texto de marcado una sección CDATA.

2.5.- Los **atributos** se escriben entre comillas. Cada elemento puede tener cero o más atributos. Los atributos siempre contienen cadenas de texto. Observa los siguientes ejemplos:

EJEMPLO 1:

```
<alumno codigo="1530">
  <dni> 44388767G </dni>
  <nombre> Ana Pérez </nombre>
</alumno>
```

EJEMPLO 2:

```
<alumno>
  <codigo>1530</codigo>
  <dni> 44388767G </dni>
  <nombre> Ana Pérez </nombre>
</alumno>
```

La única diferencia entre ambos ejemplos es que en el primero hemos utilizado código como un atributo de la etiqueta alumno, y en el segundo código es en sí mismo una etiqueta. ¿Cuál de los dos ejemplos es mejor?

Bueno, podemos decidirnos por cualquier ejemplo. Lo importante es saber que un atributo es mucho más limitado que una etiqueta. Un atributo no puede tener información descendiente. Un atributo no será un nodo del árbol sintáctico, etc. En general la propuesta es usar atributos cuando representen sólo modificaciones de una etiqueta concreta.

Con los atributos ocurre que el mismo atributo no puede aparecer repetido en la misma etiqueta, pero sí en etiquetas diferentes. Si trabajamos con un DTD en éste se han de haber definido previamente qué atributos soporta cada etiqueta.




PRACTICA 03.- Modifica el fichero de la práctica 01 para que el elemento libro soporte un atributo llamado “categoría”.

Crea un libro en la categoría de cocina, otro en la categoría de novela y dos en la categoría de programación.

Vuelve a chequear que el documento xml es válido.

3.- El prólogo de los documentos xml.



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

Esta línea es una instrucción de procesamiento, y es **obligatoria**.

- Version: indica la versión de XML que se está utilizando (1.0 en la actualidad). Es obligatorio indicarlo.
- Encoding: indica cómo se codificó el documento, y no es obligatorio indicarlo (por defecto se aplica el formato UTF-8). El xml es válido para otros juegos de caracteres (muchos más que en html, por ejemplo Unicode, Utf-16, etc).
- Standalone: “yes” indica que el documento **no** va acompañado de DTDs externos, mientras que un valor “no” indica que posee DTD externo. No es un atributo obligatorio, lo utilizaremos solamente cuando estemos con documentos que trabajan con DTDs, que veremos en breve.



```
<!DOCTYPE MiDTD SYSTEM "C:\MiDTD.dtd">
```

Esta instrucción sirve para definir qué documento estamos utilizando como DTD, así como el nombre que le daremos al elemento raíz. El nombre del documento raíz y la DTD deben coincidir.

El documento xml que tenga esta instrucción deberá cumplir con las normas establecidas en MiDTD para ser **válido**.

4.- Los DTDs.

DTD son las siglas de Document Type Definition (documento de definición de tipos). Cualquier documento que utilice un DTD concreto tiene que cumplir obligatoriamente con la gramática que éste define. Un DTD es, por tanto, la estructura interna de los datos válidos que puede contener un documento.

La DTD establece la **gramática** de un vocabulario XML, el cual, determina la estructura de los documentos XML que serán válidos. Para que un documento se adhiera a una DTD, deberá incluir en su cabecera una declaración de tipo de documento (DOCTYPE) justo después de declarar la versión de xml:

Una DTD puede ser **interna**, lo que significa que se declara en el mismo documento XML que la contiene. Veamos un ejemplo:



Documento xml con DTD interna.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE planetas SYSTEM [
```

```
<!ELEMENT nombre (#PCDATA) ]
>
<planetas>
  <nombre>Venus</nombre>
  <nombre>Marte</nombre>
</planetas>
```

- Se define la DTD planetas, que coincide con la etiqueta raíz. A su vez esta DTD informa que el fichero se organiza con etiquetas <nombre> que contienen texto.

Por otra parte una DTD **externa** está en un fichero separado de nuestros archivos xml. Muchos archivos xml podrán utilizar esta misma DTD externa. Tendremos, por tanto, que especificar en el DOCTYPE de todos ellos la ruta, llamada URI, hacia la DTD externa. Veamos un ejemplo:



Documento xml con DTD externa.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE planetas SYSTEM "c:\misdatos\planetas.dtd">
<planetas>
  <nombre>Venus</nombre>
  <nombre>Marte</nombre>
</planetas>
```

- Se define la DTD planetas que está declarada en el fichero externo c:\misdatos\planetas.dtd. Es importante colocar rutas relativas o de internet, si está la DTD publicada en un servidor web. Recuerda que las rutas locales dan problemas si movemos nuestros ficheros de carpeta.

También pueden darse combinaciones de ambas, es decir, indicar una ruta hacia un fichero .dtd y además agregar reglas locales al fichero xml que la utiliza.

Ejemplo:

```
<!DOCTYPE planetas SYSTEM "planetas.dtd" [
  <!ELEMENT nombre (#PCDATA) ]>
```

Cuando se utilizan los dos tipos de DTD, la DTD final será la combinación de ambas.

Función principal de un DTD:

Definir los elementos, atributos, entidades y notaciones que pueden utilizarse en un documento xml para que sea **válido**. Además define las reglas que todos sus elementos deben seguir.

Y a esto se le llama: **gramática**.

Las DTD tienen su origen en el lenguaje SGML, y tienen una sintaxis específica que aprenderemos en este tema. Es importante saber que su nombre debe coincidir con la etiqueta raíz del documento xml.

4.1.- Definición de elementos.

Recuerda que llamamos elemento al conjunto de una etiqueta y todo su contenido. Para especificar un elemento en una DTD seguimos la siguiente regla general.

<!ELEMENT NombreElemento Contenido>



ELEMENT es una palabra reservada, escríbela en mayúsculas. NombreElemento será el nombre de la etiqueta, debe ser descriptivo. Contenido es lo que la etiqueta puede contener. En muchos casos una etiqueta contendrá otras etiquetas que podrán ser opcionales u obligatorias. Entramos ahora a detallar todos los casos y como se escriben. Por el momento observa un ejemplo:



```
<!ELEMENT Libros (Libro)+>
<!ELEMENT Libro (Titulo, Autor)>
<!ELEMENT Titulo (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
```

Esta DTD nos indica que comenzamos por la etiqueta Libros.
 La etiqueta Libros puede contener “una o más” etiquetas Libro.
 La etiqueta Libro contiene un Título y un Autor.
 La etiqueta Titulo contiene una cadena de texto.
 La etiqueta Autor contiene una cadena de texto.
 El siguiente código corresponderá a un documento xml válido para la DTD

Libros:



```
<Libros>
  <Libro>
    <Titulo>Don Quijote de la Mancha</Titulo>
    <Autor>Miguel de Cervantes</Autor>
  </Libro>
  <Libro>
    <Titulo>La vida es sueño</Titulo>
    <Autor>Calderon de la Barca</Autor>
  </Libro>
</Libros>
```



PRACTICA 04.- Usando EditiX crea un nuevo archivo de tipo DTD y crea una gramática para nuestra librería. Está formada por libros y cada libro tiene un título, un autor y una editorial. Utiliza el ejemplo anterior para desarrollarla.

Esta DTD trabajará con el fichero xml de la práctica 01 (no de la práctica 03). Agrega la línea de código para incluir en el xml la referencia a su DTD. Valida.

Ahora la validación incluye que el elemento esté bien formado (código xml bien escrito) pero también si es un documento válido (cumple su DTD).



PRACTICA 05.- Agrega al primer libro dos etiquetas autor. Vuelve a validar. ¿Dónde se produce el fallo?

Sigamos aprendiendo cómo escribir DTDs. Hemos visto ELEMENT, que será una etiqueta. Colocamos primero el nombre de la misma y a continuación el contenido. En relación a este contenido hay varias posibilidades:

– **EMPTY**: palabra reservada que indica que el elemento está vacío (aunque puede contener atributos).



Ejemplo en el DTD: `<!ELEMENT LibroAgotado EMPTY>`
Ejemplo en xml: `<LibroAgotado />`

– **ANY**: el elemento puede contener a cualquier otro elemento o incluso contenido de texto. Es decir, cualquier cosa. No se usa mucho, atenta contra una buena estructura de los documentos.



Ejemplo en el DTD: `<!ELEMENT Editorial ANY>`
Ejemplo en xml: `<Editorial> Santillana </Editorial>`
`<Editorial><Jefe>Ana Pérez</Jefe></Editorial>`

– **Otros elementos**: un elemento puede contener uno o más elementos hijos en una cierta secuencia:



Ejemplo en el DTD: `<!ELEMENT Libro (Titulo, Autor)>`
El elemento Libro contiene un elemento Título y un elemento Autor.

– **#PCDATA**: indica que el contenido es simplemente un texto que analizará el parser.



Ejemplo en el DTD: `<!ELEMENT Titulo (#PCDATA)>`
Ejemplo en xml: `<Titulo>Don Quijote de La Mancha</Titulo>`

– **Combinaciones** mixtas: el elemento puede incluir tanto texto como opcionalmente elementos hijos.



Ejemplo en el DTD: `<!ELEMENT Titulo (#PCDATA | Autor)*>`
Ejemplos en xml: `<Titulo> La sombra del viento </Titulo>`
`<Titulo> La sombra del viento`
`<Autor> JL Zafon </Autor>`
`</Titulo>`

4.2.- Elementos hijo.

Para indicar elementos hijo que estarán jerárquicamente dentro de un elemento concreto tenemos varias posibilidades:

– **Secuencia en orden**: indica los elementos hijo en el orden exacto que se debe seguir. Se utiliza el operador **coma**. Podemos leerlo como un “Y”.



Ejemplo en el DTD: `<!ELEMENT Libro (Titulo, Autor)>`
Ejemplo en xml: `<Libro>`
`<Titulo>Don Quijote de La Mancha</Titulo>`
`<Autor>Miguel de Cervantes Saavedra </Autor>`
`</Libro>`



PRACTICA 06.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (DNI, Nombre)>
<!ELEMENT DNI (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <DNI />
  <Nombre>Juan Ruiz </Nombre>
</Alumno>
```

Documento 2:

```
<Alumno>
  <DNI>44304238J</DNI>
  <Nombre>Pedro Vargas</Nombre>
</Alumno>
```

Documento 3:

```
<Alumno>
  <Nombre>Eva Murillo</Nombre>
  <Nombre>Juan Ruiz </Nombre>
</Alumno>
```

Documento 4:

```
<Alumno>
  <DNI>44304238J</DNI>
  <Nombre> </Nombre>
</Alumno>
```

Solución: todos son válidos salvo el documento 3. El hecho de que un elemento incluya un texto con #PCDATA no quiere decir que ese texto no pueda ser vacío.

– **Secuencia opcional:** indica que puede ser uno cualquiera de los elementos. Se utiliza el operador **barra vertical**. Podemos leerlo como un “**Ó**”



Ejemplo en el DTD: `<!ELEMENT Libro (Titulo | Autor)>`

Ejemplo en xml:

```
<Libro>
  <Titulo>Don Quijote de La Mancha</Titulo>
</Libro>
```

```
<Libro>
  <Autor>Miguel de Cervantes Saavedra </Autor>
</Libro>
```



PRACTICA 07.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (DNI | Nombre)>
<!ELEMENT DNI (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <DNI />
  <Nombre>Juan Ruiz </Nombre>
</Alumno>
```

Documento 2:

```
<Alumno>
  <DNI>44304238J</DNI>
  <Nombre>Pedro Vargas</Nombre>
</Alumno>
```

Documento 3:

```
<Alumno>
```

Documento 4:

```
<Alumno>
```

```
<Nombre>Eva Murillo</Nombre>      <DNI>44304238J</DNI>
</Alumno>                          </Alumno>
```

Solución: Sólo son válidos el documento 3 y el 4. Podemos tener DNI o nombre, pero no los dos a la vez.

– **Cero o más repeticiones:** indica que el elemento puede no aparecer, aparecer una vez o aparecer muchas veces. Se utiliza el operador **asterisco**.



Ejemplo en el DTD: `<!ELEMENT Libro (Titulo*)>`
 Ejemplo en xml:

```
<Libro>
  <Titulo>Don Quijote de La Mancha</Titulo>
  <Titulo> El asombroso Hidalgo </Titulo>
</Libro>

<Libro>
  </Libro>
```



PRACTICA 08.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (Nombre, Hermano*)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Hermano (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <Nombre>Juan Ruiz </Nombre>
  <Hermano>Eva Ruiz </Hermano>
</Alumno>
```

Documento 2:

```
<Alumno>
  <Nombre>Luis Vargas</Nombre>
  <Nombre>Pedro Vargas</Nombre>
</Alumno>
```

Documento 3:

```
<Alumno>
  <Nombre>Eva Murillo</Nombre>
</Alumno>
```

Documento 4:

```
<Alumno>
  <Nombre>Iker Lopez</Nombre>
  <Hermano>Ana Lopez </Hermano>
  <Hermano>Eva Lopez</Hermano>
  <Hermano>Alba Lopez</Hermano>
</Alumno>
```

Solución: Sólo son válidos el 1, 3 y 4.

– **Una o más repeticiones:** indica que el elemento puede aparecer una vez o aparecer muchas veces. Una aparición es obligatoria. Se utiliza el operador **suma**.



Ejemplo en el DTD: `<!ELEMENT Libro (Titulo+)>`
 Ejemplo en xml:

```
<Libro>
  <Titulo>Don Quijote de La Mancha</Titulo>
  <Titulo> El asombroso Hidalgo </Titulo>
</Libro>

<Libro>
  <Titulo>La sombra del viento </Titulo>
</Libro>
```



PRACTICA 09.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (DNI*, Nombre+)>
<!ELEMENT DNI (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <Nombre>Juan Ruiz </Nombre>

</Alumno>
```

Documento 2:

```
<Alumno>
  <DNI>78999999L </DNI>
  <Nombre>Pedro Vargas</Nombre>

</Alumno>
```

Documento 3:

```
<Alumno>
  <DNI>78645534H</DNI>
  <DNI>44312211F</DNI>

</Alumno>
```

Documento 4:

```
<Alumno>
  <Nombre>Iker </Nombre>
  <Nombre>Manuel </Nombre>

</Alumno>
```

Solución: son válidos el 1, 2 y 4.

– **Cero o una repetición:** indica que el elemento puede no aparecer o bien aparecer una única vez. Se utiliza el operador **interrogación**.



Ejemplo en el DTD: <!ELEMENT Libro (Titulo?)>

Ejemplo en xml:

```
<Libro>
  <Titulo>Don Quijote de La Mancha</Titulo>

</Libro>

<Libro>

</Libro>
```



PRACTICA 10.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (DNI?, Nombre+)>
<!ELEMENT DNI (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <Nombre>Juan Ruiz </Nombre>

</Alumno>
```

Documento 2:

```
<Alumno>
  <DNI>55324412G </DNI>
  <Nombre>Pedro Vargas</Nombre>

</Alumno>
```

Documento 3:

```
<Alumno>
  <DNI>78645534H</DNI>
  <DNI>44312211F</DNI>

</Alumno>
```

Documento 4:

```
<Alumno>
  <DNI>55313424Y </DNI>

</Alumno>
```


Solución: son válidos el 1 y el 2.



PRACTICA 11.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumno (#PCDATA | Nombre?)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumno>
  <Nombre>Juan Ruiz </Nombre>
</Alumno>
```

Documento 2:

```
<Alumno>
  Carlos Morales
  <Nombre>Pedro Vargas</Nombre>
</Alumno>
```

Documento 3:

```
<Alumno>
  Carlos Morales
</Alumno>
```

Documento 4:

```
<Alumno>
</Alumno>
```

Solución: son válidos el 1, 3 y el 4.



PRACTICA 12.- Sea el siguiente DTD indica qué documentos xml son válidos y cuáles no.

```
<!ELEMENT Alumnos (Dni | Nombre)*>
<!ELEMENT Dni (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Documento 1:

```
<Alumnos>
  <Nombre>Juan Ruiz </Nombre>
</Alumnos>
```

Documento 2:

```
<Alumno>
  Carlos Morales
</Alumno>
```

Documento 3:

```
<Alumno>
  <Dni>45666777V</Dni>
</Alumno>
```

Documento 4:

```
<Alumno>
</Alumno>
```

Documento 5:

```
<Alumno>
  <Dni>45666777V</Dni>
  <Dni>43227377A</Dni>
  <Nombre>Eva Lopez </Nombre>
</Alumno>
```

Documento 6:

```
<Alumno>
  <Nombre>Ana Suarez</Nombre>
  <Dni>66555789H</Dni>
  <Nombre>Luis Perez</Nombre>
</Alumno>
```

Solución: son válidos todos menos el dos.



PRÁCTICA 13.- Si modificamos el DTD de la práctica anterior indica qué documentos xml del ejercicio anterior son válidos y cuáles no.

```
<!ELEMENT Alumnos (Dni | Nombre)+>
<!ELEMENT Dni (#PCDATA)>
<!ELEMENT Nombre (#PCDATA)>
```

Solución: todos válidos menos el 2 y el 4.



PRACTICA 14.- Sea el siguiente DTD:

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC , DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

¿Es válido el siguiente documento?

```
<!DOCTYPE XXX SYSTEM "XXX.dtd">
<XXX>
<AAA>
<CCC/><DDD/>
</AAA>
<BBB>
<CCC/><DDD/>
</BBB>
</XXX>
```

Solución: sí.

¿Cuál de los elemento AAA ó BBB no es obligatorio?.

¿Y el siguiente documento xml será válido?

```
<!DOCTYPE XXX SYSTEM "XXX.dtd">
<XXX>
<AAA/>
<BBB>
<CCC/><DDD/>
</BBB>
</XXX>
```

Solución: sí.

¿Y este otro documento xml?

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
<AAA/><AAA/>
<BBB>
<CCC/><DDD/>
</BBB>
</XXX>
```

Solución: no es válido, el AAA sólo puede aparecer cero o una vez.



PRACTICA 15.- Sea el siguiente DTD:

```
<!ELEMENT XXX (AAA+ , BBB+)>
<!ELEMENT AAA (BBB | CCC )>
```

```
<!ELEMENT BBB (#PCDATA | CCC)*>
<!ELEMENT CCC (#PCDATA)>
```

¿Será válido este documento?

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
```

```
<XXX>
```

```
<AAA>
```

```
<CCC>Exactamente un elemento.</CCC>
```

```
</AAA>
```

```
<AAA>
```

```
<BBB>
```

```
<CCC/>
```

```
<CCC/>
```

```
<CCC/>
```

```
</BBB>
```

```
</AAA>
```

```
<BBB/>
```

```
<BBB>
```

Esta es <CCC/> una combinación <CCC/> de <CCC> elementos CCC
</CCC> y texto <CCC/>.

```
</BBB>
```

```
<BBB>
```

Sólo texto.

```
</BBB>
```

```
</XXX>
```

Solución: sí. ¿Pero a que sería más fácil si escribimos respetando la indentación?



PRACTICA 16.- Sea el siguiente DTD:

```
<!ELEMENT XXX (AAA+)>
```

```
<!ELEMENT AAA EMPTY>
```

¿Es válido el siguiente documento?

```
<XXX>
```

```
  <AAA /> <AAA />
```

```
</XXX>
```



PRACTICA 17.- Describe con tus palabras qué elementos se están definiendo y qué tipo de hijos pueden contener en las siguientes declaraciones de DTD:

1.- <!ELEMENT LIBRO (Autor, Editorial)>

2.- <!ELEMENT Autor (#PCDATA)>

3.- <!ELEMENT PELICULA (Actor|Actriz|Director)+>

4.- <!ELEMENT PELICULA ((Actor | Actriz)*, Director, Maquillaje?)>

5.- <!ELEMENT PELICULA (#PCDATA | Actor)*>

6.- <!ELEMENT PELICULA (Titulo, Genero, (Actor | Actriz | Narrador)*)>

7.- <!ELEMENT FICHA (Nombre+, Apellido+, Direccion*, foto?, TelFijo*|TelMovil*)>



PRACTICA 18.- Realiza una DTD apropiada para representar el siguiente documento xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<Agenda>
  <Persona>
    <Nombre> Eva </Nombre>
    <Apellido> Lopez </Apellido>
    <Email> evalopez@minik.org </Email>
    <Oficina> 2.1 B18 </Oficina>
    <Telefono> 5555555 </Telefono>
    <Movil> 5557777 </Movil>
  </Persona>
</Agenda>
```

4.3.- Definición de atributos.

Para especificar los atributos de una etiqueta utilizamos la directiva **!ATTLIST**, lista de atributos:

<!ATTLIST Elemento Atributo Tipo Modificador>



Los atributos de un elemento pueden incluirse en una o más declaraciones **<!ATTLIST ...>**. Si se hace en la misma declaración, basta con separarlos con un espacio (espacio, tabulador, retorno de carro).

Recuerda que los atributos deben encerrarse siempre entre comillas dobles. En el siguiente ejemplo definimos el atributo numero en la etiqueta capitulo:

```
<libro>
  <capitulo numero="1">
    <subcapitulo>
      ;
    </subcapitulo>
  </capitulo>
</libro>
```

Según la definición sintáctica de la página anterior primero colocamos el nombre del elemento (etiqueta), después el nombre del atributo y después tendremos que definir el tipo y el modificador. En cuanto al tipo existen varias posibilidades:

- **Atributo cadena de texto:** basta con especificar la palabra CDATA.
Ejemplo: **<!ATTLIST Autor Nacionalidad CDATA>**
Ejemplo en xml: `<Autor Nacionalidad="ruso">Chejov</Autor>`
- **Atributo con lista de valores acotada:** basta con especificar los distintos valores entre paréntesis y separados por el operador barra vertical (se lee ó):
Ejemplo: **<!ATTLIST Pelicula Genero (Ficcion | Terror | Humor)>**
Ejemplo en xml: `<Pelicula Genero="Terror">Saw III</Pelicula>`

- **Atributo con ID:** el identificador ya lo hemos utilizado en temas anteriores. El ID debe ser único para cada etiqueta del documento, y sirve para identificarla desde otros lenguajes como JavaScript, CSS, etc. Sólo podrá existir un atributo ID por cada elemento. Sólo puede contener valores válidos como NMToken y empezar por una letra.

Ejemplo: <!ATTLIST Pelicula id ID>

Ejemplo en xml: <Pelicula ID="2839" Genero="Terror">Saw III</Pelicula>
<Pelicula ID="5300" Genero="Humor">SreckII</Pelicula>

- **Atributo con referencia:** son los atributos IDREF e IDREFS. Los que hacen es que referencian a otro elemento de la página con un id concreto. En el caso de IDREFS se puede poner una lista de ID separados por espacio.

Ejemplo: <!ATTLIST Cliente alquiler IDREF>

Ejemplo en xml: <Cliente alquiler="2839">Juan Perez</Cliente>

Y si en la DTD cambiamos y usamos IDREFS podemos hacer:

<Cliente alquiler="2839 5399">Pedro Gamez</Cliente>

- **Atributo de Entidad:** su valor debe coincidir con una o más entidades no analizadas. Las dejaremos para más adelante.
- **NMTOKEN, NMTOKENS:** su valor ha de ser una cadena de tipo token. Mientras que los atributos de tipo CDATA pueden contener cualquier carácter (siempre que esté dentro de las reglas del estándar xml), los atributos de tipo NMTOKEN y NMTOKENS no:
 - Si es del tipo NMTOKEN sólo puede contener letras, dígitos, punto [.], guión [-], subrayado [_] y dos puntos [:] .
 - Los del tipo NMTOKENS pueden contener los mismos caracteres que NMTOKEN más espacios en blanco. Un espacio en blanco consiste en uno o más espacios, retornos de carro o tabuladores.

Y por último podemos agregar el modificador. En cuanto al modificador existen también varias posibilidades:

- **Atributos obligatorios:** su presencia en la etiqueta es obligatoria. Usan la palabra reservada #REQUIRED.

Ejemplo: <!ATTLIST Alumno Dni CDATA #REQUIRED>

Ejemplo en xml: <Alumno Dni="43122866X">Juan Perez</Alumno>

- **Atributos opcionales:** su presencia en la etiqueta no es obligatoria. Usan la palabra reservada #IMPLIED.

Ejemplo: <!ATTLIST Alumno Dni CDATA #IMPLIED>

Ejemplo en xml: <Alumno>Juan Perez</Alumno>

<Alumno Dni="123444332F"> Juan González </alumno>

- **Atributos con valor por defecto:** es una manera de indicar el valor que toma el atributo cuando no se especifica nada.

Ejemplo: <!ATTLIST Pelicula Genero (Terror|Humor) "Humor">

Ejemplo en xml: <Pelicula>Srek II</Pelicula> → es una película de Humor
<Pelicula Genero="Terror">Saw III</Pelicula> → es de terror

Ejemplo: <!ATTLIST Autor Nacionalidad CDATA "Española">

Si no indicamos nacionalidad por defecto será española.

- **Atributos fijos:** se incluya o no se incluya el atributo, los procesadores siempre obtendrán este mismo valor.

Ejemplo: `<!ATTLIST Autor Nacionalidad CDATA #FIXED "Española">`

Tanto si se indica como si no en xml cualquier valor, el procesador que analice el documento xml considerará españoles a todos los autores.

Todos los atributos de una etiqueta pueden declararse en la misma línea, o dentro de la misma directiva ATTLIST. La separación entre ellos será un espacio en blanco, o tabulador o retorno de carro, lo que queramos. Observa el ejemplo y describe con tus palabras la naturaleza de cada atributo:



PRACTICA 19.- Comenta con tus propias palabras qué se define en el siguiente extracto de una DTD:

```
<!ATTLIST curso
  director CDATA #REQUIRED
  horario (mañana | tarde | noche) #IMPLIED
  matricula (ordinaria | extraordinaria) "Ordinaria"
  tasas CDATA #FIXED "53€">
```



PRACTICA 20.- Comenta con tus propias palabras qué se define en el siguiente extracto de una DTD:

```
<!ELEMENT pelicula(titulo, writer+, productor+, director+, actor*,
comentarios?)>
<!ATTLIST pelicula
  tipo (drama | comedy | adventure | sci-fi | mystery | horror | romance |
documentary) "drama"
  clasificacion (G | PG | PG-13 | R | X) "PG"
  review (1 | 2 | 3 | 4 | 5) "3"
  año CDATA #IMPLIED
>
```

Frecuentemente un mismo objeto se puede diseñar como un atributo o un elemento sin pérdida de semántica pero existen criterios para decantarnos por una opción o por otra:

Cuándo nos decantaremos por un atributo en lugar de hacer una etiqueta:

- Normalmente se trata de objetos cuya existencia no tiene sentido fuera del objeto al que describen (p.e.adjetivos), metadatos, identificadores únicos, el idioma, ...
- En general, todo aquello por lo que existe mayor interés en filtrarlo que en mostrarlo.

Ventajas del uso de atributos:

- Más fácil de procesar por el software (mayor eficiencia).

- Más legible, los atributos están próximos al elemento al que pertenecen.

Nos decantaremos por crear elementos (etiquetas) en los siguientes casos:

- Siempre que se quiera definir sub-elementos.
- Los elementos permiten crear vínculos.
- Siempre que queramos repetir el mismo elemento con distinto valor (los atributos tienen un único valor como máximo).
- Siempre que el contenido sea mayor que una palabra (oraciones, párrafos, ...) y sobre todo si se quiere mostrar el texto en cuestión.
- Tienen entidad propia independientemente del resto de elementos.

Los documentos que priman a los atributos pueden ser más breves al no tener una etiqueta propia que abrir y cerrar. No obstante no permitirán tantas búsquedas y extracciones de información.



PRACTICA 21.- Observando la siguiente DTD indica si es válido el código xml posterior:

```
<!ELEMENT mietiqueta (#PCDATA)>
<!ATTLIST mietiqueta
  AtribA CDATA #IMPLIED
  AtribB NMTOKEN #REQUIRED
  AtribC NMTOKENS #REQUIRED>
```

Documento 1:

```
<mietiqueta AtribB="12.5">Texto </mietiqueta>
```

Documento 2:

```
<mietiqueta AtribB="12.5" AtribC="3:16">Texto </mietiqueta>
```

Documento 3:

```
<mietiqueta AtribA="hola" AtribB="12.5" AtribC="3 6">Texto </mietiqueta>
```

Documento 4:

```
<mietiqueta AtribA="17:5" AtribB="17,5" AtribC="2 3">Texto </mietiqueta>
```

Documento 5:

```
<mietiqueta AtribA="aa" AtribB="17 5" AtribC="hola">Texto </mietiqueta>
```

Solución: son válidos el uno, el dos y el tres.

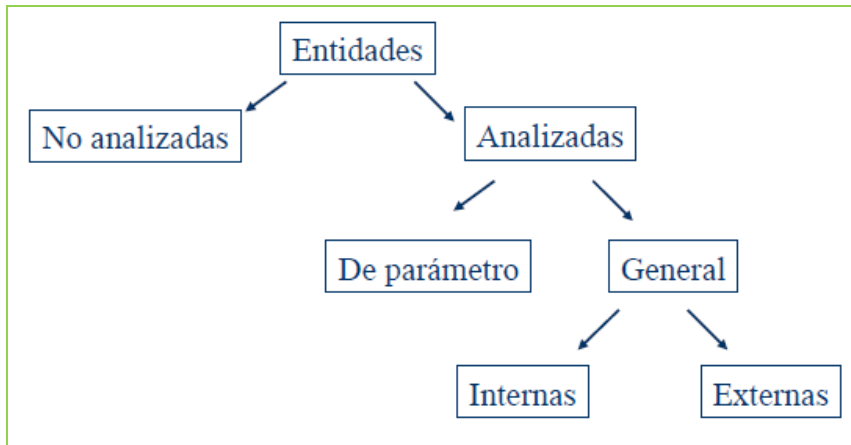
4.4.- Definición de entidades generales.

En XML para hacer referencia a objetos como ficheros, páginas web, imágenes, etc, se utilizan las **entidades**, que se declaran en la DTD mediante el uso de "<!ENTITY>".

A veces una entidad no es más que una manera abreviada de colocar un texto. Otras veces es una forma de referenciar un archivo. Cada tipo de entidad se declara siguiendo una sintaxis determinada.

Las entidades pueden ser:

- Internas o Externas
- Analizadas o No analizadas.
- Generales o de Parámetro.



Las entidades no analizadas se utilizan principalmente para referenciar archivos binarios (fotos, imágenes, pdfs, etc). Este tipo de archivos no deben ser analizados por un parser de XML puesto que no se trata de código XML.

Por otra parte las entidades analizadas sí deben ser revisadas por un parser. Pueden ser “de parámetro” o “generales”. Las entidades de parámetro se usan sólo dentro de la propia DTD. Las entidades generales se utilizarán dentro del contenido del documento XML que estemos creando y podrán ser “internas” o “externas”.

Para poder utilizar una entidad en un documento XML primero tendremos que **declararla**. Esto se hace especificándola dentro de la DTD:

<!ENTITY NombreEntidad ContenidoEntidad>



La parte ContenidoEntidad es el que determina el tipo de entidad que estamos declarando. Veamos algunos casos:

- **Entidad general interna:** el ContenidoEntidad es una cadena de texto. Cada vez que encontremos la entidad en el código XML se sustituirá por esta cadena. Veamos un ejemplo:

1.- Declaración:

<!ENTITY empresa “Minik Corporation 2010”>

2.- Uso: para usarlas en XML seguimos la sintaxis **&NombreEntidad**; ¿Te suena? Los caracteres especiales en HTML son entidades. < © ´...

Todos comienzan por un ampersand y terminan con un punto y coma.

Para usar nuestra entidad pondremos en nuestro código &empresa;



PRACTICA 22.- Crea mensajes.DTD. Guardará la gramática de un sistema de envío de mensajes. Cada mensaje está compuesto de un autor, uno o más destinatarios y un asunto. Todos contienen texto.

Además crea una entidad general interna que se llame “yo” y guarde tu nombre y apellidos. Crea otra entidad general interna que se llame “curso” que guarde el texto “Primero de Administración de Sistemas Informáticos en Red”.

A continuación crea un fichero practica21.xml que guarde algunos mensajes. Al final de cada asunto utiliza las entidades “yo” y “curso”.

- **Entidades generales externas analizadas:** el ContenidoEntidad será una referencia a un archivo externo XML que será analizado por un parser. Deben indicar la ruta al archivo al que apuntan y antes usaremos las palabras **SYSTEM** ó **PUBLIC**.
 - **SYSTEM:** se usa para archivos que están en local o en una unidad de red accesible desde nuestra máquina, es decir, en nuestro sistema de archivos.
 - **PUBLIC:** lo usaremos cuando sea un archivo publicado en un servidor web y sea accesible públicamente. No obstante PUBLIC es opcional, se puede poner siempre SYSTEM.

Ejemplos de declaración:

```
<!ENTITY datosanexo SYSTEM "anexo.xml" >
<!ENTITY tema1 SYSTEM "http://www.miservidor.com/tema1.xml">
<!ENTITY tema2 SYSTEM "http://www.miservidor.com/tema2.xml">
```

Y a la hora de usarlas usamos la sintaxis ya conocida:

```
<temario>
    &tema1;
    &tema2;
</temario>
<otrosdatos>
    &datosanexo;
</otrosdatos>
```

- **Entidades generales externas no analizadas:** el ContenidoEntidad será una referencia a un archivo externo binario (pdfs, fotos, películas de flash, etc).
 - En este caso en la sintaxis se utiliza la palabra reservada **NDATA**.

Ejemplo:

```
<!ENTITY mifoto SYSTEM "foto.jpeg" NDATA JPEG>
<!ELEMENT persona EMPTY>
<!ATTLIST persona nombre CDATA #REQUIRED
                foto ENTITY #IMPLIED>
```

Y para utilizarlo:

```
<persona nombre="Pedro Pérez" foto="&mifoto;"/>
```

Nota: no es necesario utilizar NDATA, se puede utilizar el sistema de entidad externa general y colocar la ruta hacia un archivo que no es de tipo xml si el navegador sabe cómo interpretar la extensión de ese archivo. Mira el ejemplo siguiente:

```
<!ENTITY logo SYSTEM "http://www.miservidor.com/logo.gif">
```

4.5.- Definición de entidades de parámetro.

Se trata de un tipo especial de entidad que sólo puede usarse en la DTD, y no en los documentos de XML. Sirven para agrupar ciertos elementos del DTD que se repitan mucho, o bien subelementos que utilicen varios elementos.

Se diferencian las entidades parámetro de las generales, en que para hacer referencia a ellas, se usa el símbolo "%" en lugar de "&" tanto como para declararlas como para usarlas. La sintaxis es la siguiente:

```
<!ENTITY % NombreEntidad ContenidoEntidad>
```



Veamos mejor cómo se usan con un ejemplo, y recuerda que este mecanismo sólo lo utilizarás cuando lo necesites porque introduce cierta complejidad a la hora de entender una DTD.

Estamos creando los elementos mesa, silla y armario. Estas tres etiquetas podrán contener subetiquetas como alto, ancho y fondo con las dimensiones del mueble concreto. Observa el siguiente extracto XML:

```
<muebles>
  <silla>
    <alto> 90cm </alto>
    <ancho>60cm</ancho>
    <fondo>45cm</fondo>
  </silla>
  <armario>
    <alto> 180cm </alto>
    <ancho>120cm</ancho>
    <fondo>55cm</fondo>
  </armario>
</muebles>
```

Ahora observa cómo ahorramos trabajo creando una entidad llamada medidas:

```
<!ENTITY % medidas "alto, ancho, fondo">
```

en la propia DTD donde creamos nuestra entidad la podemos utilizar:

```
<!ELEMENT mesa (%dimensiones;)>
<!ELEMENT silla (%dimensiones;)>
<!ELEMENT armario (%dimensiones;)>
```



PRACTICA 23.- Sea la siguiente DTD:

```
<!ENTITY % contacto "direccion, telefono">
<!ELEMENT alumno (nombre, %contacto;)>
<!ELEMENT centro (denominacion, %contacto;)>
```

1º.- Termina de definir la DTD.

2º.- Crea un documento XML con dos alumnos y un centro.

5.- Herramientas.

El número de herramientas para trabajar con XML y DTDs aumenta cada día. Algunas son gratuitas, otras no. Entre estas últimas las hay que generan automáticamente DTDs a partir del análisis de un simple documento XML. Otras representan el árbol sintáctico de forma gráfica.

Algunos ejemplos:

- SoftQuad Xmetal. Está orientada al desarrollo de documentos validados que se van a ver en la red. Soporta el uso de CSS. Requiere que todos los documentos XML sean válidos.
- Adobe FrameMaker+SGML. Proporciona una GUI completa que manipula documentos SGML/XML. Permite abrir documentos no válidos y rectificarlos.
- IBM Xena. Herramienta desarrollada en Java. Proporciona una GUI para editar documentos válidos.
- Bluestone Visual-XML. Herramienta de creación XML orientada a la creación de soluciones de bases de datos empresariales que incluyen XML. Posee un asistente para la publicación de B.B., que genera DTD y plantillas de documentos XML a partir de tablas de bases de datos. Puede descargarse gratuitamente en: <http://www.bluestone.com/xml/visual-XML>.
- XML Notepad. Se trata de una herramienta poco potente de Microsoft, que sirve para realizar tareas de creación sencillas. Para validar documentos requiere tener instalado el IE5.0. Puede descargarse gratuitamente en: <http://msdn.microsoft.com/xml/notepad>
- XMLSPY <http://www.xmlspy.com/>. Es una herramienta muy famosa. La casa de software Altova es su productor. Es cara pero muy potente, automatiza casi todas las tareas.
- XML WRITER <http://xmlwriter.net/>



CICLO FORMATIVO DE GRADO SUPERIOR:

“DESARROLLO DE APLICACIONES WEB”



MÓDULO:

**Lenguajes de marcas y
sistemas de gestión de
información**

(4 horas semanales)