

IES CHAN DO MONTE

C.S. de Desarrollo de Aplicaciones Multiplataforma

UNIDAD 1 Parte 2: Gestión de Ficheros XML

Índice

1.	Introducción	2
1.1	¿Qué es XML?.....	2
1.2	Estructura de un documento XML	2
1.3	¿Qué es un XML Parser?	6
1.4	APIS XML.....	7
1.5	APIs XML en Java.....	10

1. Introducción

XML (eXtensible Markup Language) es clave en el desarrollo de software, especialmente en entornos empresariales. Muchas aplicaciones lo utilizan para:

 **Presentación:** XML permite guardar los datos por separado del diseño visual, lo que facilita mostrar la misma información en distintos formatos según el dispositivo o el contexto.

Ejemplo: una reserva de avión puede mostrarse en HTML (web), XHTML (móvil) o PDF (impresión).

 **Comunicación:** permite que distintas aplicaciones se entiendan sin depender del formato visual. Ejemplo: empresas que intercambian pedidos o facturas usando vocabularios XML comunes (eBusiness).

 **Interacción:** permite que dos componentes se comuniquen a distancia, invocando funciones o servicios a través de la red. Esto se hace con tecnologías como SOAP o XML-RPC.

 **Configuración:** se usa para guardar ajustes y parámetros que necesita una aplicación para funcionar correctamente. Es fácil de leer, modificar y compartir.

1.1 ¿Qué es XML?

XML significa Extensible Markup Language, es un metalenguaje extensible de etiquetas desarrollado por el W3C. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Ventajas principales

- **Extensible:** se pueden añadir nuevas etiquetas sin romper compatibilidad.
- **Estándar:** existen analizadores XML ya disponibles, lo que evita errores y acelera el desarrollo.
- **Compatible:** otros sistemas pueden entender y procesar fácilmente documentos XML. Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones

1.2 Estructura de un documento XML

El **documento XML** consta del **prólogo** (opcional) y el **cuerpo**.

- El prólogo puede constar de la **declaración XML**, la **declaración de tipo de documento** e **instrucciones de procesamiento**.
- El cuerpo forma el documento en sí, es decir, las **etiquetas y su contenido**.

El prólogo

■ La declaración XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- **version:** La versión del XML utilizada (habitualmente 1.0, ya disponible la 1.1)
- **encoding:** Se refiere al juego de caracteres utilizado.
- **standalone** (yes | no) : informa de si el documento es aislado, es decir, si no depende de declaraciones externas (como un DTD, p.e.). Por defecto es no

■ El tipo de documento

```
<!DOCTYPE HTML PUBLIC "-// /W3C/ /DTD HTML 3.2 Final/ /EN">
<!DOCTYPE documento SYSTEM "doc.dtd">
<!DOCTYPE clientes SYSTEM "http://www.servidor.com/defs/clientes.dtd">
<!DOCTYPE elementoRaíz [
    definición del DTD ]>
```

La declaración <!DOCTYPE> se utiliza en XML para:

- ✓ Indicar el **nombre del elemento raíz del documento**
- ✓ Especificar **dónde se encuentra la DTD** (Document Type Definition), que define las reglas del documento: qué elementos puede tener, en qué orden, con qué atributos, etc.,

✳️ Formatos posibles de DOCTYPE

1. DTD externa pública (PUBLIC)

Hace referencia a una DTD **estándar y compartida públicamente**, usando un identificador público.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

- HTML: nombre del elemento raíz
- PUBLIC: indica que se usa una DTD pública
- "-//W3C//DTD HTML 3.2 Final//EN": identificador público de la DTD

Este tipo se usaba mucho en documentos HTML antiguos.

2. DTD externa privada (SYSTEM)

Hace referencia a una DTD **externa y específica**, localizada mediante una ruta o URL.

```
<!DOCTYPE documento SYSTEM "doc.dtd">
<!DOCTYPE clientes SYSTEM "http://www.servidor.com/defs/clientes.dtd">
```

- documento o clientes: nombre del elemento raíz
- SYSTEM: indica que se usa una DTD externa privada
- "doc.dtd" o URL: ruta local o remota del archivo DTD

Muy común en proyectos XML personalizados.

3. DTD interna (incrustada en el propio XML)

La DTD se **define directamente dentro del documento XML**, entre corchetes [].

```
<!DOCTYPE elementoRaiz [
  <!ELEMENT elementoRaiz (#PCDATA)>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT autor (#PCDATA)>
]>
```

- elementoRaiz: nombre del elemento raíz
- Dentro de []: definiciones de elementos y atributos

Útil para documentos pequeños o pruebas locales, sin depender de archivos externos.

▣ Instrucciones de procesamiento

Las instrucciones de procesamiento permiten **pasar información a aplicaciones externas** para que sepan cómo tratar el documento XML.

- Suelen ir en el **prólogo**, pero pueden aparecer **frente a los elementos**.
- Sintaxis:

```
<?nombreAplicacion instrucciones?>
```

- El nombre **no puede empezar por "xml"** (ni en mayúsculas ni minúsculas), porque está reservado.

Ejemplos comunes

- Aplicar una hoja de estilo:

```
<?xmlstylesheet type="text/css" href="estilos.css"?>
```

- Incluir código PHP:

```
<?php echo "Hola mundo"; ?>
```

El cuerpo

Consta de las etiquetas XML necesarias para estructurar el documento junto con su contenido. Hay que recordar que deberá existir, al menos, un elemento raíz, para que el documento se considere bien formado.

Elementos:

Las partes que componen el cuerpo de un documento XML son los elementos, y se forman con etiquetas: textos reconocidos por el analizador y delimitados por los caracteres <>.

- Se escriben entre etiquetas <nombre> y </nombre>
- Pueden contener texto, otros elementos o estar vacíos
- Pueden incluir atributos

Ejemplo:

```
<poema fecha="Abril de 1915" lugar="Granada">
    <verso>En el silencio de la noche...</verso>
</poema>
```

Atributos

Los elementos pueden tener **atributos**, que son una manera de incorporar **características o propiedades a los elementos** de un documento.

- Se colocan en la etiqueta de apertura
- No pueden repetirse
- El orden **no importa**

Ejemplo:

```
<evento fecha="2025-10-13" lugar="Poio" />
```

Comentarios

Sirven para **añadir notas o explicaciones** que el procesador XML ignora.

- No se permite la secuencia -- dentro del comentario
- Se pueden usar <, > o & sin que se interpreten como etiquetas

```
<!-- Esto es un comentario a>b & c<d -->
```

Entidades

Las entidades **permiten reutilizar contenido** o **referenciar recursos externos** dentro de un documento XML.

El procesador XML las sustituye automáticamente por su contenido o las trata como objetos externos.

- Se declaran en la DTD mediante el uso de <!ENTITY>
- Para referenciar una entidad dentro del documento XML se hace de la siguiente manera:

```
&nombreDeLaEntidad
```

Tipos de entidades XML:

▪ Entidades generales internas

- **¿Qué son?** Abreviaturas definidas dentro del propio documento XML.
- **¿Para qué sirven?** Para reutilizar texto y evitar repeticiones.
- **¿Cómo se usan?**

```
<!DOCTYPE saludo [
    <!ENTITY firma "Atentamente, Pepa">    ]>
<saludo>
    <mensaje>Gracias por tu colaboración.</mensaje>
    <despedida>&firma;</despedida>
</saludo>
```

&firma; se sustituye por **"Atentamente, Pepa"**

▪ Entidades generales externas analizadas

- **¿Qué son?** Referencias a contenido XML externo que **sí se analiza**.
- **¿Para qué sirven?** Para insertar contenido desde otro archivo o URL.

- **¿Cómo se usan?**

```
<!ENTITY intro SYSTEM "http://www.miservidor.com/intro.xml">
...
<documento>
  &intro;
</documento>
```

El **procesador** carga y analiza el contenido de intro.xml

- **Entidades generales externas no analizadas**

- **¿Qué son?** Referencias a archivos externos **no XML** (imagen, vídeo...).
- **¿Para qué sirven?** Para vincular recursos binarios sin analizarlos.
- **¿Cómo se usan?**

```
<!NOTATION gif SYSTEM "image/gif">
<!ENTITY logo SYSTEM "http://www.miservidor.com/logo.gif" NDATA gif>
...
<imagen archivo=&logo;" tipo="gif" />
```

El **procesador** no interpreta el contenido, solo lo referencia

Entidades parámetro (internas y externas)

- **¿Qué son?** Fragmentos reutilizables **solo dentro deL DTD**
- **¿Para qué sirven?** Para agrupar definiciones repetidas
- **¿Cómo se usan?**

```
<!ENTITY % elemento-alf "<!ELEMENT ALF (#PCDATA)>">
...
%elemento-alf;
```

Se usa % en lugar de & y solo dentro de la DTD

- **Entidades predefinidas**

- **¿Qué son?** Caracteres especiales representados por entidades estándar
- **¿Para qué sirven?** Para evitar conflictos con la sintaxis XML
- **¿Cómo se usan?**

```
<código>
  if x < 4 then x := x + 1;
</código>
```

< se interpreta como <

- **Secciones CDATA**

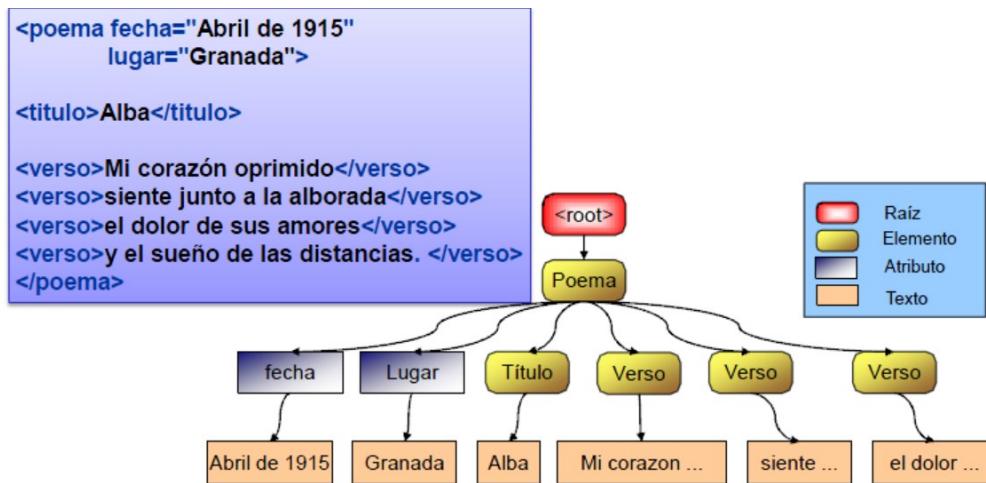
- **¿Qué son?** Bloques de texto que el procesador **no interpreta**
- **¿Para qué sirven?** Para incluir código o texto literal sin usar entidades
- **¿Cómo se usan?**

```
<![CDATA[
  if x < 4 then x := x + 1;
]]>
```

El **contenido** se muestra tal cual, sin analizar etiquetas ni símbolos

Árbol del documento XML

Sólo puede haber un único elemento raíz. Cualquier documento XML **puede representarse como un árbol**



1.3 ¿Qué es un XML Parser?

- Un parser XML (anализатор) es el componente que:
 - Verifica que el documento esté bien formado
 - Permite leer y manipular los datos XML
 - Puede validar el documento contra una DTD o un esquema
- ¿Por qué es útil?
 - No necesitas escribir código desde cero para leer XML
 - Existen librerías estándar en Java, Python, C#, etc.
 - Facilita la integración de XML en aplicaciones web, móviles o de escritorio

Ejemplo práctico:

Supón que tienes este XML:

```

<usuario>
    <nombre>Pepa</nombre>
    <rol>Docente</rol>
</usuario>
  
```

Con un parser, puedes acceder a los datos así:

```
parser.getElement("usuario").getChild("nombre").getText(); // "Pepa"
```

▪ Tipos de parser

Tipo de parser	¿Valida?	¿Qué verifica?
💡 Sin validación	✗ No	Solo que el documento esté bien formado
✓ Con validación	✓ Sí	Bien formado + cumple reglas de DTD/XSD(esquema)

Ambos tipos permiten acceder a los datos, pero el segundo añade control de calidad estructural.

Los parser pueden ser software independiente o venir en librería de clases.

El parser XML es la herramienta principal de cualquier aplicación XML. Mediante el parser no solamente podemos comprobar si nuestros documentos son bien formados, sino que también podemos incorporarlos a nuestras aplicaciones, de manera que estas puedan manipular y trabajar con documentos

El parser o analizador de XML es el programa que verifica que el documento XML esté bien formado, lo lee y hace que los datos que contiene estén accesibles de una manera u otra. Además, el parser puede tener otras funcionalidades, como validar el documento contra una DTD o un schema.

1.4 APIS XML

- ✓ Existen diversas **formas de procesar documentos XML**, dependiendo del modo en que se acceda a la información y del tipo de aplicación que se deseé desarrollar.
- ✓ Cada **enfoque dispone de un API estándar** (Application Programming Interface) que define cómo debe comportarse el analizador (parser). Estas especificaciones permiten que distintas compañías implementen sus propias versiones compatibles con el estándar.
- ✓ Cada una ofrece un enfoque distinto para leer, procesar, modificar o generar XML, dependiendo del tipo de aplicación y del tamaño del documento.
- ✓ Tradicionalmente se han utilizado dos APIs principales para procesar XML: **SAX y DOM**. Sin embargo, en la actualidad se reconocen cuatro enfoques complementarios ampliamente utilizados en Java: **SAX, DOM, StAX y JAXB**.
 - **SAX** (API Simple para XML): procesa los documentos XML mediante **eventos**. El parser recorre el documento secuencialmente y lanza métodos cuando detecta elementos, atributos o texto. No guarda el documento en memoria.
 - **DOM** (Modelo de Objetos del Documento): procesa los documentos XML **como árboles**. El parser carga todo el documento en memoria y lo representa como una estructura jerárquica de nodos, que se puede recorrer, modificar y guardar.
 - **StAX** (Streaming API for XML): procesa los documentos XML **como un flujo secuencial controlado por el programador**, permitiendo leer y escribir paso a paso. El parser no lanza eventos automáticamente, sino que el código solicita cada fragmento cuando lo necesita.
 - **JAXB** (Java Architecture for XML Binding): convierte automáticamente documentos **XML en objetos Java y viceversa**. El programador anota las clases Java para definir cómo deben representarse en XML, lo que simplifica el trabajo con estructuras orientadas a objetos.

SAX (Simple API for XML)

- Enfoque general:
 - **Procesamiento por eventos** (lectura secuencial del documento).
 - ✓ SAX lee el archivo XML **de arriba hacia abajo, sin cargarlo completamente en memoria**.
 - ✓ A medida que el parser avanza por el documento, **detecta eventos** (como el inicio o fin de un elemento, texto, atributos o comentarios) y **llama automáticamente a los métodos** del manejador de eventos que el programador haya definido.
 - Es un modelo **basado en eventos y callbacks**:
 - ✓ cuando **ocurre un evento**, SAX “avisa” al programa mediante una **llamada a un método** (por ejemplo, startElement() o characters()).
 - Una vez procesado un fragmento, **la memoria se libera inmediatamente**, lo que permite trabajar con archivos XML muy grandes sin problemas de rendimiento.
- **Ventajas:**
 - **Rápido y eficiente en memoria**.
 - Solo procesa la parte del documento que se está leyendo en cada momento, sin almacenar el resto.
 - Ideal para procesar XML muy grandes o cuando solo se necesita una parte del contenido.
 - **Perfecto para tareas de lectura secuencial**, como validaciones, búsquedas o conteos de elementos.
- **Desventajas:**
 - **No permite modificar el XML ni recorrerlo libremente**, ya que no se construye una estructura en memoria (no hay árbol que recorrer).
 - **No se puede retroceder ni acceder a elementos anteriores**: la lectura es estrictamente hacia adelante.
 - Requiere más código y una lógica más compleja, ya que el programador debe implementar manualmente los manejadores de eventos (métodos que responden a cada tipo de evento).

DOM (Document Object Model)

■ Enfoque general

- Procesamiento en memoria (modelo en forma de árbol).
 - ✓ DOM **carga todo el documento XML en memoria**, construyendo una estructura jerárquica (**un árbol de nodos**) que representa todos sus elementos, atributos, textos y comentarios.
 - ✓ Cada **nodo del árbol corresponde a una parte del XML** (por ejemplo, una etiqueta, un atributo o un texto).
 - ✓ Esta estructura **permite acceder, recorrer y modificar cualquier parte del documento** de manera sencilla.
 - ✓ El **parser DOM devuelve un objeto de tipo Document**, que contiene todos los nodos del árbol y que **puede manipularse mediante las clases y métodos del API DOM**.

■ Ventajas

- **Acceso completo y aleatorio a toda la información** del XML.
- Se puede recorrer el árbol libremente, acceder a cualquier elemento y modificarlo.
- Permite **leer, modificar, crear y guardar** documentos XML.
- Muy intuitivo: su modelo de árbol resulta fácil de entender y utilizar.
- Ideal cuando se **necesita trabajar con la estructura completa** del XML (por ejemplo, editar configuraciones o generar nuevos documentos).

■ Desventajas

- **Alto consumo de memoria**, ya que todo el documento se carga completo en memoria, incluso si solo se necesita una pequeña parte.
- **Menor rendimiento con XML muy grandes** o complejos.



StAX (Streaming API for XML)

■ Enfoque general

- **Procesamiento por flujo controlado (lectura o escritura “pull”).**
 - ✓ Fue introducida en Java 6 como una alternativa a SAX y DOM.
 - ✓ Permite **procesar documentos XML de forma secuencial**, pero con una diferencia clave respecto a SAX: En StAX, es el programador quien **controla cuándo y cómo se leen los eventos**, en lugar de que el parser los envíe automáticamente.
 - No **carga todo el documento en memoria**, sino que lee los datos a medida que el programa los solicita.
 - Por eso se conoce como un modelo de extracción (“pull parsing”), mientras que SAX utiliza un modelo de empuje (“push parsing”).
 - ✓ También **permite escribir XML**, lo que lo convierte en una herramienta flexible tanto para lectura como para generación de documentos.

■ Ventajas

- **Eficiente y con bajo consumo de memoria**, ya que no necesita almacenar el documento completo.
- **Mayor control del flujo de lectura**: el programador decide cuándo avanzar y qué procesar.
- Permite tanto leer como escribir XML (a diferencia de SAX, que solo lee).
- Ideal para procesar documentos grandes.
- Su código es más limpio y fácil de mantener que el de SAX.

■ Desventajas

- No permite recorrer el documento de forma aleatoria, ya que **trabaja secuencialmente**.
- No se construye una estructura completa del XML, por lo que **no es adecuado para modificar todo el documento**.
- Menos cómodo que DOM si se necesita editar o reestructurar el XML.

JAXB (Java Architecture for XML Binding)

■ Enfoque general

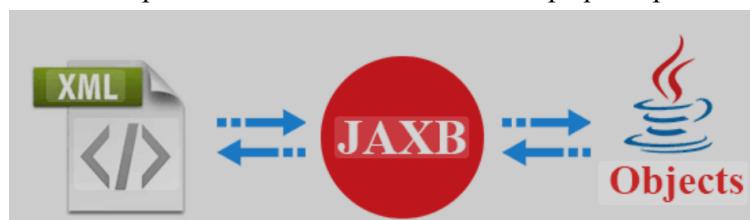
- **Procesamiento mediante mapeo entre objetos Java y XML.**
 - ✓ JAXB es una API que **permite convertir objetos Java en XML y viceversa** de manera automática.
 - ✓ En lugar de recorrer o analizar manualmente los elementos del XML (como en SAX, DOM o StAX), **JAXB trabaja directamente con objetos Java**.
 - ✓ El proceso de vinculación (“binding”) **se basa en anotaciones** dentro de las **clases Java**. (@XmlRootElement, @XmlElement, @XmlAttribute, etc.), que indican cómo se corresponde cada atributo o clase con el XML.
 - ✓ JAXB puede realizar dos operaciones principales:
 - **Marshalling**: convertir un objeto Java en un documento XML.
 - **Unmarshalling**: leer un XML y generar a partir de él un objeto Java con los datos correspondientes.

■ Ventajas de JAXB

- Muy **fácil de usar y mantener**.
- Permite trabajar con XML sin tener que analizarlo manualmente.
- **Totalmente orientado a objetos**.
El desarrollador solo maneja clases Java, no nodos ni eventos.
- Permite tanto leer como escribir XML.
- Perfecto para servicios web (SOAP o REST) y aplicaciones que intercambian datos estructurados en XML.
- Menor cantidad de código, ya que las conversiones se realizan automáticamente.

■ Desventajas de JAXB

- Menos flexible si el XML tiene una estructura muy variable o no está bien definida.
- Requiere que el documento XML tenga una **estructura predecible y estable**.
- No ideal para procesamiento parcial o cuando solo interesa una pequeña parte del XML.



1.5 APIs XML en Java

Existen diversas **APIs para manejar XML** desde una **aplicación Java**

- **JAXP** (Java API for XML Processing) es el API estándar de Java y la más popular, incluye:
 - **SAX** (Analizador basado en eventos)
 - **DOM** (Analizador tipo árbol)
 - **Transformer** (transformador de documentos XML): las API XSLT define en `javax.xml.transform` permiten escribir datos XML en un archivo o convertirla en otras formas.
- JAXP es un componente estándar de la plataforma Java. Una implementación del JAXP 1.3 está incluido en J2SE 5.0 y una implementación de JAXP 1.4 es en Java SE 6.0 y OpenJDK7.
- **Stax** (Streaming API for XML) es una API para leer y escribir documentos XML. Introducido en Java 6.0 y considerado como superior a SAX y DOM.
- **Arquitectura Java para XML Binding (JAXB)** es un estándar de Java que define cómo los objetos Java se convierte a XML y viceversa (especificar mediante un conjunto estándar de asignaciones). Permite mapear objetos entre documentos XML.
- Otras APIs para XML en Java son: **JDOM**, **dom4j**, **XMPPull**.
 JDOM: un API para leer, escribir, crear y manipular XML cómodamente desde Java. Al ser un **API específico para Java** es mucho más intuitivo y sencillo que los anteriores, pero no está pensado para otros lenguajes.

Una visión general de los Paquetes

Las bibliotecas que definen las APIs XML son las siguientes:

- **javax.xml.parsers**: El API JAXP, que proporcionan una interfaz común para los diferentes vendedores SAX y DOM analizadores.
- **org.w3c.dom**: Define el Documento de clase (un DOM), así como clases para todos los componentes de un DOM
- **org.xml.sax**: Define el API SAX básico.
- **javax.xml.transform**: define las API XSLT que permiten transformar XML en otras formas
- **javax.xml.stream**: Clases de StAX para lectura y escritura de XML.
- **javax.xml.bind**: Clases de JAXB para mapeo objeto ↔ XML.