AAD

Acceso a Datos
UD02T03 – Manejo de archivos XML desde JAVA (DOM)

IES Ciudad Escolar
Desarrollo de Aplicaciones Multiplataforma

Índice

- XML
- Estructura de un documento XML
- Validación de documentos XML
- DOM
- SAX
- Xpath

XML

- XML es el acrónimo de E**X**tensible **M**arkup **L**anguage.

- XML es un lenguaje de marcas que permite definir y almacenar datos de forma compartible.

- XML es utilizado en multitud de contextos, sobre todo en:

  - **Intercambio de datos entre sistemas:** permite la posibilidad de intercambiar datos de forma estructurada entre diferentes sistemas al tratarse de un formato de texto plano y ser un lenguaje estandarizado, haciendo que esta transferencia sea muy ágil e independiente de la plataforma utilizada.

  - **Base de datos,** XML también permite guardar datos de forma estandarizada para luego poder ser tratados por multitud de lenguajes diferentes. Su manejo es mucho más sencillo que bases de datos como MySQL y mucho más rico que utilizar ficheros de texto planos.

XML

- Un archivo de XML es un **documento basado en texto** que se puede guardar con la extensión .xml.

- Cualquier archivo XML incluye los siguientes componentes:

  - **Declaración XML:** Un documento XML comienza con alguna información sobre el propio XML. Por ejemplo, podría mencionar la versión XML que sigue. Esta apertura se denomina declaración XML. EJ: <?xml version="1.0" encoding="UTF-8"?>

  - **Elementos XML:** Resto de etiquetas que consten en un documento XML. Los elementos XML pueden contener las siguientes características:
    - Texto
    - Atributos: Añaden información adicional sobre los elementos.
    - Otros elementos

Validación de documentos XML: DTD

- Un DTD es un conjunto de reglas que permiten especificar nuestro propio conjunto de elementos y atributos.
- Se trata también de un documento de texto, pero en este caso, su **sintaxis NO es XML**.
- Un documento XML se considera válido si dispone de su correspondiente DTD y además está estructurado de acuerdo a las reglas definidas en el DTD.
- El DTD aunque se puede indicar en el propio XML, lo habitual es disponer de él en un fichero independiente.

```
<!ELEMENT alumnos (alumno+)>
<!ELEMENT alumno (expediente,nombre,edad)>
<!ELEMENT expediente (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT edad (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF8" ?>
<!DOCTYPE alumnos SYSTEM "alumnos.dtd">
<alumnos>
  <alumno>
    <expediente>11111</expediente>
    <nombre>carlitos</nombre>
    <edad>21</edad>
  </alumno>
  <alumno>
    <expediente>22222</expediente>
    <nombre>menganito</nombre>
    <edad>28</edad>
  </alumno>
  <alumno>
    <expediente>33333</expediente>
    <nombre>fermin</nombre>
    <edad>24</edad>
  </alumno>
  <alumno>
    <expediente>44444</expediente>
    <nombre>julito</nombre>
    <edad>23</edad>
  </alumno>
</alumnos>
```

Validación de documentos XML: XSD

- Un XSD o XML Schema Definition es la evolución natural del sistema de validación de documentos XML.
- Su sintaxis permite más capacidades de validación y resulta más coherente.
- Hoy en día es la arquitectura predominante para la validación, ya que se usa en XQuery y sobre todo en los servicios Web.
- Se trata también de un documento de texto, pero en este caso, **su sintaxis SÍ es XML**.
- Un documento XML se considera válido si dispone de su correspondiente XSD y además está estructurado de acuerdo a las reglas definidas en el DTD.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xsd:element name="alumnos">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="alumno" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="expediente"/>
              <xsd:element name="nombre"/>
              <xsd:element name="edad"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```xml
<?xml version="1.0" encoding="UTF8" ?>
<alumnos
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="alumnos.xsd">
  <alumno>
    <expediente>11111</expediente>
    <nombre>carlitos</nombre>
    <edad>21</edad>
  </alumno>
  <alumno>
    <expediente>22222</expediente>
    <nombre>menganito</nombre>
    <edad>28</edad>
  </alumno>
  <alumno>
    <expediente>33333</expediente>
    <nombre>fermin</nombre>
    <edad>24</edad>
  </alumno>
  <alumno>
    <expediente>44444</expediente>
    <nombre>julito</nombre>
    <edad>23</edad>
  </alumno>
</alumnos>
```

DOM vs SAX

DOM : Document Object Model

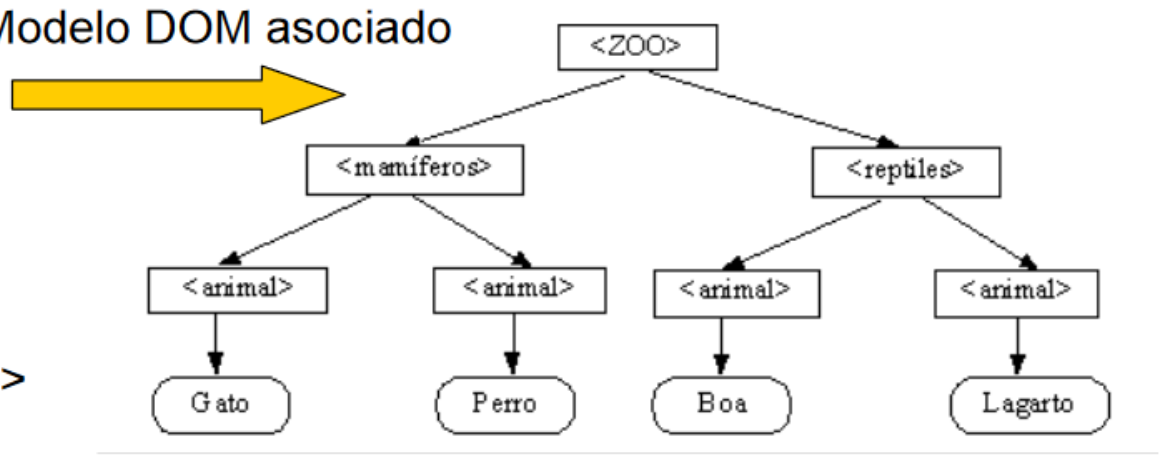
- DOM es un estándar de acceso y manipulación de documentos XML, es decir, nos permite obtener, cambiar, añadir y borrar elementos XML.

- DOM se basa en un conjunto de interfaces que permiten la representación y manipulación tanto de la estructura como del contenido de un documento XML.

- Permite la carga de un fichero XML en memoria almacenándolo en una estructura de árbol resultando útil sobre todo cuando el fichero XML no es muy grande.

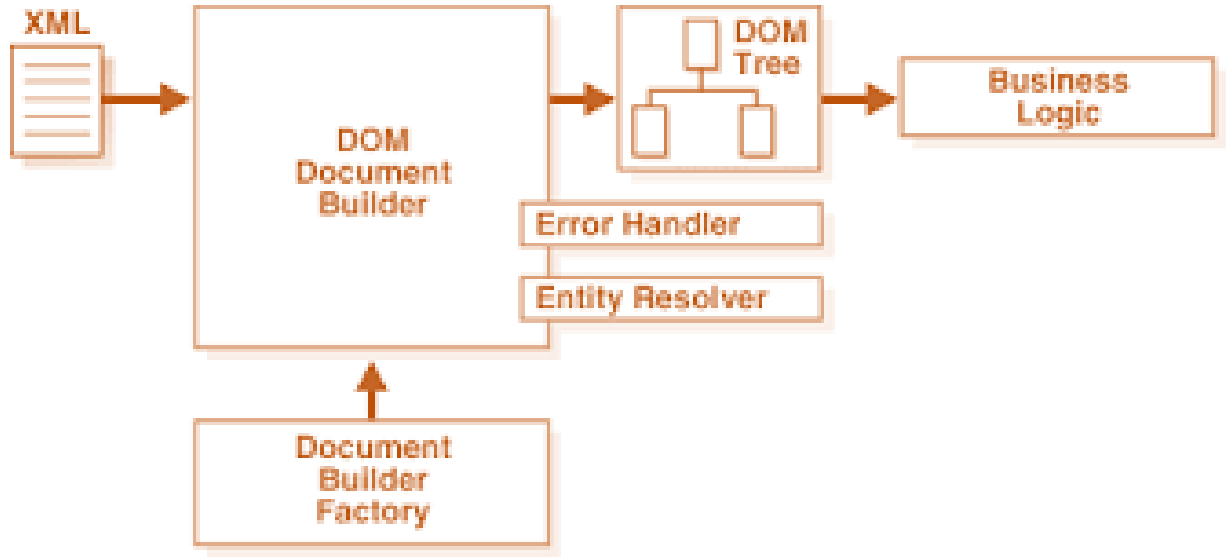
```
<zoo>
  <mamiferos>
    <animal> Gato </animal>
    <animal> Perro </animal>
  </mamiferos>
  <reptiles>
    <animal> Boa </animal>
    <animal> Lagarto </animal>
  </reptiles>
</zoo>
```

Modelo DOM asociado



DOM : Document Object Model

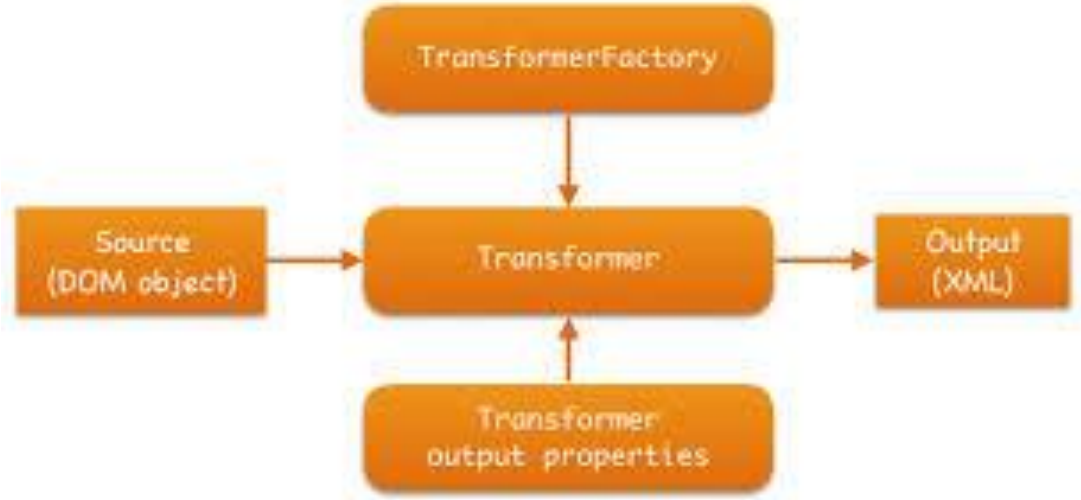
- La **operativa** común en el uso de DOM para "parsear" un fichero XML es:




```
1. File ficheroXML = new File("E:\\alumnos.xml");

2. DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

3. DocumentBuilder db = dbf.newDocumentBuilder();

4. Document document = db.parse(ficheroXML);
```

DOM : Document Object Model

- La **operativa** común en el volcado a fichero de una estructura DOM es:



Save from DOM Document to XML

```
1. TransformerFactory tf = TransformerFactory.newInstance();
2. Transformer t = tf.newTransformer();
3. DOMSource source = new DOMSource(document);
   4.a StreamResult result = new StreamResult(new FileWriter(new File("E:\\salidaDOM.xml")));
   4.b StreamResult result = new StreamResult(System.out);
   4.c StreamResult result = new StreamResult(new FileOutputStream("E:\\salidaDOM.xml"));
5. t.transform(source,result);
```

opciones

DOM : Document Object Model

- En el modelo DOM para XML, se considera un **nodo** a cualquier cosa que se encuentra dentro del documento:
  - Todo el documento en su conjunto es un **nodo documento**.
  - Cada elemento XML es un **nodo elemento**.
  - El texto de los elementos XML son **nodos texto**.
  - Cada atributo es un **nodo atributo**.
  - Los comentarios son **nodos comentario**.
- Para manipular los nodos en Java se dispone de la clase **Node**, que se encuentra en el paquete org.w3c.dom

DOM : Métodos para navegación por los nodos

```
Node getFirstChild()
```

Retorna el **primer nodo** hijo de este nodo. Si no lo hay, retorna *null*.

```
Node getNextSibling()
```

Retorna el nodo inmediatamente **siguiente** (hermano) a este nodo. Si no lo hay, retorna *null*.

```
Node getParentNode()
```

Retorna el **padre** de este nodo. Todos los nodos, excepto los de tipo *Attr*, *Document*, *DocumentFragment*, *Entity*, y *Notation* pueden tener un padre. Sin embargo, será *null* si el nodo está recién creado y no ha sido añadido todavía al árbol, o si ha sido eliminado del árbol.

```
NodeList getChildNodes()
```

Retorna una lista de la clase *NodeList* que contiene todos los **nodos hijo** de este nodo. Si no hay nodos hijo, la lista estará vacía.

DOM : Métodos para navegación por los nodos

La **clase NodeList** dispone de los siguientes métodos que permiten obtener los nodos que componen una de esas listas:

```
int getLength()
```

Retorna el **número de nodos** de la lista.

```
Node item(int index)
```

Retorna el **nodo** de la lista que ocupa la posición *index*.

DOM : Métodos para gestionar los nodos

La **clase NodeList** dispone de los siguientes métodos que permiten obtener los nodos que componen una de esas listas:

```
Node appendChild(Node hijoNuevo)
```

Añade el nodo *hijoNuevo* al final de la lista de hijos de este nodo. Si el *hijoNuevo* ya está en el árbol, se elimina previamente. Retorna el nodo que se ha añadido.

```
Node insertBefore(Node hijoNuevo, Node hijoReferencia)
```

Inserta el nodo *hijoNuevo* antes del nodo *hijoReferencia* que ya existe. Si el nodo *hijoReferencia* es *null*, se inserta el nodo *hijoNuevo* al final de la lista de nodos hijo. Retorna el nodo que se ha insertado.

```
Node removeChild(Node hijoViejo)
```

Elimina, de la lista de hijos, el hijo indicado por parámetro como *hijoViejo* y lo retorna.

```
Node replaceChild(Node hijoNuevo, Node hijoViejo)
```

Sustituye el nodo *hijoViejo* que se encuentre en la lista de nodos hijo por el nodo *hijoNuevo*.

DOM : Métodos para manipular la info del nodo

```
short getNodeType()
```

Retorna un código identificativo del **tipo de nodo** que se trata. Para trabajar cómodamente con esos códigos, la clase *Node* proporciona una serie de constantes para identificar los tipos de nodos:

Tipo de nodo	Constante
Attr	ATTRIBUTE_NODE
CDATASection	CDATA_SECTION_NODE
Comment	COMMENT_NODE
Document	DOCUMENT_NODE
DocumentFragment	DOCUMENT_FRAGMENT_NODE
DocumentType	DOCUMENT_TYPE_NODE
Element	ELEMENT_NODE
Entity	ENTITY_NODE
EntityReference	ENTITY_REFERENCE_NODE
Notation	NOTATION_NODE
ProcessingInstruction	PROCESSING_INSTRUCTION_NODE
Text	TEXT_NODE

DOM : Métodos para manipular la info del nodo

```
String getNodeName()
```

Retorna el **nombre** de nodos de determinados tipos, como los atributos (*Attr*) y elementos (*Element*).

```
String getNodeValue()
```

Retorna el **valor** de nodos de determinados tipos, como los atributos (*Attr*), comentarios (*Comment*) y textos (*Text*).

```
String getTextContent()
```

Retorna el **texto** del nodo y de sus descendientes.

```
void setNodeValue(String valorNodo)
```

Establece el valor de nodos de determinados tipos, como los atributos (*Attr*), comentarios (*Comment*) y textos (*Text*).