Recuerda que **un esquema** (o **schema**) es una especificación XML que dicta los componentes permitidos de un documento XML y las relaciones entre los componentes. Por ejemplo, un esquema identifica los elementos que pueden aparecer en un documento XML, en qué orden deben aparecer, qué atributos pueden tener, y qué elementos son subordinados (esto es, son elementos hijos) para otros elementos. Un documento XML no tiene por qué tener un esquema, pero si lo tiene, debe atenerse a ese esquema para ser un documento XML válido.

Para leer ficheros XML y acceder a su contenido y estructura, se utiliza **un procesador de XML o parser**.

- El procesador lee los documentos y proporciona acceso a su contenido y estructura.
- Son independientes del lenguaje de programación y existen versiones particulares para Java, C#, VisualBasic, etc.
- Algunos de los procesadores más empleados son DOM (Modelo de Objetos de Documento) y SAX (API Simple para XML). Utilizan dos enfoques muy diferentes:
  - **DOM**: un procesador XML que utilice este planteamiento almacena toda la estructura del documento en memoria, en forma de árbol con nodos padre, nodos hijo y nodos finales (que son aquellos que no tienen descendientes). Una vez creado el árbol, se van recorriendo los diferentes nodos y se analiza a qué tipo particular pertenecen. Tiene su origen en el W3C. Este tipo de procesamiento necesita más recursos de memoria y tiempo sobre todo si los ficheros XML a procesar son bastantes grandes y complejos.

  - **SAX**: un procesador que utilice este planteamiento lee un fichero XML de forma secuencial y produce una secuencia de eventos (comienzo/fin del documento, comienzo/fin de una etiqueta, etc) en función de los resultados de la lectura. Cada evento invoca a un método definido por el programador. Este tipo de procesamiento prácticamente no consume memoria, pero por otra parte, impide tener una visión global del documento por el que navegar.

Hay otros "parsers" conocidos, además de SAX y DOM, como por ejemplo JDOM y JAXB que permiten trabajar con XML y Java.

7.1 ACCESO A FICHEROS XML CON DOM

Para poder trabajar con DOM en Java necesitamos:

- El paquete **javax.xml.parsers** del API estándar de Java que proporciona un par de clases abstractas que toda implementación DOM para Java debe extender. Estas clases proprocionan métodos para cargar documentos desde una fuente de datos (fichero, InputStream, etc). Las dos clases fundamentales que permiten construir un procesador o parser XML son:
  - ***DocumentBuilderFactory***
  - ***DocumentBuilder***

- El paquete **javax.xml.transform** que permite especificar una fuente y un resultado, ya que DOM no define ningún mecanismo para generar un fichero XML a partir de un árbol DOM. La fuente y el resultado pueden ser ficheros, flujos de datos o nodos DOM entre otros.
  - ***Source.*** Un objeto que implementa esta interfaz contiene la información necesaria para actuar como fuente de entrada (fuente XML o instrucciones de transformación).
  - ***Result***. Un objeto que implementa esta interfaz contiene la información necesaria para construir un árbol de resultados de transformación.
  - ***TransformerFactory***: Permite crear instancias de la clase ***Transformer,*** de manera que se pueda transformar un árbol de origen en un árbol de resultados.

- El paquete **org.w3c.dom** (contenido en el SDK de Java) que proporcina clases e interfaces para representar los diferentes elementos del árbol DOM y manejarlos. Algunas de ellas son:
  - ***DOMImplementation.*** Proporciona métodos para realizar una serie de operaciones que son independientes de cualquier instancia particular del modelo de objeto de documento.
  - ***Document***. Representa un ejemplar de documento XML. Permite crear nuevos nodos en el documento.
  - ***Element***. Cada elemento del documento XML tiene un equivalente en un objeto de este tipo. Expone propiedades y métodos para manipular los elementos del documento y sus atributos.
  - ***Node***. Representa a cuaquier nodo del documento.
  - ***NodeList***. Contiene una lista con los nodos hijos de un nodo.

- ***Attr.*** Permite acceder a los atributos de un nodo.
- ***Text.*** Son los datos carácter de un elemento.
- ***CharacterData***. Representa a los datos carácter presentes en el documento. Proporcina atributos y métodos para manipular los datos de caracteres.
- ***DocumentType***. Proporciona información contenida en el etiqueta <!DOCTYPE>

Para **crear un fichero XML a partir de un fichero de datos**, por ejemplo a partir del fichero de acceso aleatorio de empleados creado en ejemplos anteriores (AleatorioEmple.dat) haremos lo siguiente:

1. Creamos una instancia de *DocumentBuilderFactory* para construir el parser o procesador XML. Como puede producir la excepción *ParserConfigurationException,* incluimos un bloque *try-catch*

```
        //obtenemos un procesador o parser XML
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
            builder = factory.newDocumentBuilder();
.....................
```

2. Creamos un documento vacío de nombre *document* con el nodo raíz de nombre *Empleados* y asignamos la versión del XML

```
//obtenemos una implementación para DOM
DOMImplementation implementation = builder.getDOMImplementation();
//creamos un documento vacío con el nodo raíz de nombre Empleados
//createDocument(String namespaceURI, String qualifiedName, DocumentType doctype)
Document document = implementation.createDocument(null, "Empleados", null);
document.setXmlVersion("1.0"); //asignamos la version de nuestro XML
```

3. Por cada registro del fichero de empleados se crea un nodo empleado con 4 hijos (id, apellido, dep, salario). Cada nodo hijo tendrá su valor. Para crear un elemento usamos el método ***createElement(String elemento)***, siendo *elemento* una cadena con el nombre indicado entre las etiquetas < >.

```
Element raiz = document.createElement("empleado"); //creamos el nodo empleado
document.getDocumentElement().appendChild(raiz);   //lo pegamos a la raiz
```

4. Se añaden los hijos a ese nodo (raíz). Para ello crearemos la función CrearElemento(), que recibe el nombre del nodo hijo y sus textos o valores que tienen que estar en formato String, el nodo al que se va a añadir (raíz) y el documento (*document*).

```
CrearElemento("id", Integer.toString(id), raiz, document); //añadir id
CrearElemento("apellido", apellidoS.trim(), raiz, document); //añadir apellido
CrearElemento("dep", Integer.toString(dep), raiz, document); //añadir dep
CrearElemento("salario", Double.toString(salario), raiz, document); //añadir salario
```

5. Para crear el nodo hijo (<id> o <apellido> o <dep> o <salario>) se escribe:

```
Element elem = document.createElement(datoEmple); //creamos hijo
```

6. Para añadir su valor o su texto se usa el método create*TextNode(String);*

```
Text text = document.createTextNode(valor); //damos valor
```

7. A continuación se añade el nodo hijo a la raiz (empleado) y su texto o valor al nodo hijo.

```
raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
elem.appendChild(text); //pegamos el valor
```

Al final se genera algo similar a esto por cada empleado:

```
<empleado>
    <id>1</id>
    <apellido>FERNANDEZ</apellido>
    <dep>10</dep>
    <salario>1000.45</salario>
</empleado>
```

Una vez creado el árbol, en los últimos pasos:

8. Se crea la fuente XML a partir del documento:

```
Source source = new DOMSource(document);
```

9. Se crea el resultado en el fichero asociado a Empleados.xml

```
Result result = new StreamResult(ficheroOut);
```

10. Se obtiene un TransformerFactory

```
Transformer transformer = TransformerFactory.newInstance().newTransformer();
```

11. Se realiza la transformación del documento a fichero

```
transformer.transform(source, result);
```

EJEMPLO 17. Código completo del ejemplo que crea un fichero XML a partir del fichero aleatorio de empleados AleatorioEmple.dat. [CrearEmpleadoXml.java]

```
import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class CrearEmpleadoXml {

  public static void main(String args[]) throws IOException {
  //Rutas de los ficheros en disco origen y destino
    File ficheroIn = new File("c:\\AD\\UD2\\AleatorioEmple.dat");
    File ficheroOut = new File("c:\\AD\\UD2\\Empleados.xml");

    RandomAccessFile file = new RandomAccessFile(ficheroIn, "r"); //Fichero de acceso aleatorio

    Source source; //Fuente para la transformación del XML
    Result result; //Resultado de la transformación XML

    int id, dep, posicion;
    Double salario;
    char apellido[] = new char[10], aux;

  //Para la creación del parser
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
      //obtenemos un procesador o parser XML
      DocumentBuilder builder = factory.newDocumentBuilder();
      //obtenemos una implementación para DOM
      DOMImplementation implementation = builder.getDOMImplementation();
      //creamos un documento vacío con el nodo raíz de nombre Empleados
      //createDocument(String namespaceURI, String qualifiedName, DocumentType doctype)
      Document document = implementation.createDocument(null, "Empleados", null);
      document.setXmlVersion("1.0"); //asignamos la version de nuestro XML

      posicion = 0; //para situarnos al principio del ficheroIn y comenzar lectura
      while (posicion < file.length()) { //mientras la posición esté dentro del fichero
        file.seek(posicion); //nos posicionamos
        id = file.readInt(); // obtengo id de empleado
        for (int i = 0; i < apellido.length; i++) {
          aux = file.readChar();//recorro uno a uno los caracteres del apellido
          apellido[i] = aux; //los voy guardando en el array
        }

        String apellidoS = new String(apellido);//convierto a String el array
        dep = file.readInt(); //obtengo dep
        salario = file.readDouble(); //obtengo salario
```

```
        if (id > 0) { //id validos a partir de 1 (las marcas a -1 no las añade)
         //crea un elemento nodo empleado
          Element raiz = document.createElement("empleado");
          //lo pegamos a la raiz
          document.getDocumentElement().appendChild(raiz);
          //añadimos los subelementos y sus valores
          CrearElemento("id", Integer.toString(id), raiz, document); //añadir id
          CrearElemento("apellido", apellidoS.trim(), raiz, document); //Apellido
          CrearElemento("dep", Integer.toString(dep), raiz, document); //añadir dep
          CrearElemento("salario", Double.toString(salario), raiz, document); //salario }
        }
        posicion = posicion + 36; //obtengo posición de acceso del siguiente empleado
      } //fin del while que recorre el ficheroIn

      //Una vez creada en memoria la estructura en árbol del XML
      //Creamos la fuente y el resultado de la transformación XML
      source = new DOMSource(document);
      result = new StreamResult(ficheroOut);


    //Objeto para realizar la transformación
      Transformer transformer = TransformerFactory.newInstance().newTransformer();
      transformer.transform(source, result);

    } catch (Exception ex) { //captura de las diferentes excepciones que se pueden originar
      System.err.println("Error: " + ex);

    }
    file.close();
  }//fin de main

/* Inserción de los datos del empleado */
  static void CrearElemento(String datoEmple, String valor, Element raiz, Document document) {
    Element elem = document.createElement(datoEmple); //creamos hijo
    Text text = document.createTextNode(valor); //damos valor
    raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
    elem.appendChild(text); //pegamos el valor
  }
} //fin de la clase
```

Para leer un documento XML:

1. Creamos una instancia de ***DocumentBuilderFactory*** para construir el parser y cargamos el documento con el método ***parse()***
2. Obtenemos la lista de nodos con nombre *empleado* de todo el documento.
3. Recorremos la lista de nodos, y para cada nodo se obtienen sus etiquetas y sus valores mediante el método construido a tal fin ***getNodo()***.

EJEMPLO 18. Código completo del ejemplo que realiza la lectura del documento XML con los datos de empleados Empleados.xml. [LecturaEmpleadoXml.java]

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class LecturaEmpleadoXml {
  public static void main(String[] args) {
    File fileIn = new File("c:\\ad\\ud2\\Empleados.xml");

    Document document;
    DocumentBuilder builder;
    NodeList empleados;
    Node emple;
```

```
    //Para la creación del parser
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

  try {
      builder = factory.newDocumentBuilder();//obtenemos un procesador o parser XML
      document = builder.parse(fileIn); //cargamos el documento desde fileIn(Empleados.xml)
      document.getDocumentElement().normalize(); //obtiene los elementos permitiendo su acceso
      //imprime el nombre del nodo raíz
      System.out.println("Elemento raíz: " + document.getDocumentElement().getNodeName());

      //crea una lista de nodos con todos los nodos empleado
      empleados = document.getElementsByTagName("empleado");

      //recorre la lista de nodos
      for (int i = 0; i < empleados.getLength(); i++) {
        emple = empleados.item(i); //obtiene un nodo
        // si es un nodo tipo elemento
        if (emple.getNodeType() == Node.ELEMENT_NODE) {
          Element elemento = (Element) emple; //obtiene los elementos del nodo
          System.out.println("ID: " + getNodo("id", elemento));
          System.out.println("Apellido: " + getNodo("apellido", elemento));
          System.out.println("Departamento: " + getNodo("dep", elemento));
          System.out.println("Salario: " + getNodo("salario", elemento));
        }
      }
    } catch (ParserConfigurationException ex) {
      //puede provocarla '.newDocumentBuilder()'
      System.out.println("Error de construcción del Lector");
    } catch (IOException ex) {
      //puede provocarla '.parse()'
      System.out.println("Error de acceso al fichero origen");
    } catch (SAXException ex) {
      //también puede provocarla '.parse()'
      System.out.println("Error de conversión del Lector");
    }
  }//fin de main

  /** Método para obtener la información de un nodo*/
  private static String getNodo(String etiqueta, Element elem) {

    NodeList nodo = elem.getElementsByTagName(etiqueta).item(0).getChildNodes();
    Node valornodo = (Node) nodo.item(0);
    return valornodo.getNodeValue();//devuelve el valor del nodo
  }
}//fin de la clase
```

7.2 ACCESO A FICHEROS XML CON SAX

SAX (API Simple para XML) es un conjunto de clases e interfaces que ofrecen una herramienta muy útil para el procesamiento de documentos XML.

- Permite analizar los documentos de forma secuencial (es decir, no carga en memoria todo el fichero como hace DOM), esto implica poco consumo de memoria aunque los documentos sean de gran tamaño, en contraposición, impide tener una visión global del documento que se va a analizar.
- SAX es más complejo de programar que DOM.
- Es una API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio parser de XML.

La **lectura de un documento XML** produce eventos que ocasiona la llamada a métodos. Los eventos son encontrar:

- la etiqueta de inicio y fin del documento (startDocument() y endDocument())
- la etiqueta de inicio y fin de un elemento (startElement() y endElement())
- los caracteres entre etiquetas (characters()),
- etc.