ACCESO A DATOS

UNIDAD 06 – XML

INDICE

6.1 – Lenguaje XML

6.1 – LENGUAJE XML

- El nombre viene de extensible markup language o “lenguaje de marcado extensible”.
- Es un estándar del W3C (World Wide Web Consortium), organización para desarrollar y mantener la web.
- Fue introducido en 1998.

6.1 – LENGUAJE XML

- Se inventó para especificar tanto los contenidos como el aspecto visual de las páginas web.
- XML surgió como un lenguaje con sintaxis similar a HTML, pero para representar exclusivamente datos de cualquier tipo.
- Entre los principios de diseño de XML está el que sea fácil de leer e interpretar tanto por humanos como por máquinas, y esta es una de las claves de su éxito.

6.1 – LENGUAJE XML

- A cambio, contiene una gran verbosidad y por eso los documentos XML ocupan mucho espacio en relación con la cantidad efectiva de información que contienen.

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

- Un documento XML es un documento de texto escrito conforme a la sintaxis del lenguaje XML.
- Se dirá entonces que está *bien formado*, si es sintácticamente correcto, lo cual es redundante.

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

- A parte de la cabecera, el documento está formado por bloques dentro de otros bloques en una estructura jerárquica.

```
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

1. **Cabecera**: Se identifica el documento como XML con detalles como versión, etc.
2. **Elementos** (XML):
   - Todo lo contenido entre etiquetas del tipo <clientes> (apertura) y </clientes> (cierre).

```
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

2. **Elementos** (XML):
   - Puede tener una única etiqueta , como validez (se sabe que no hay etiqueta de cierre porque al final de la de apertura se encuentra /> y no >).
   - Dentro pueden contener:
     - Un texto (NADALES)
     - Otros elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

2. **Elementos** (XML):
   - Puede tener texto y otros elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<comunidades>
    <comunidad>EXTREMADURA
        <provincia CAPITAL="CÁCERES">CÁCERES</provincia>
        <provincia CAPITAL="BADAJOZ">BADAJOZ</provincia>
    </comunidad>
    <comunidad>ASTURIAS
        <provincia CAPITAL="OVIEDO">ASTURIAS</provincia>
    </comunidad>
</comunidades>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.1 – LENGUAJE XML

ESTRUCTURA DE UN DOCUMENTO XML

3. **Atributos** :
   - Aparecen en las etiquetas de apertura de los elementos y consisten en la asignación de un valor a un nombre.

```
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.2 – DOM

6.2 – DOM

- DOM es un modelo que define interfaces de programación para acceder a los contenidos de un documento XML y modificarlo.
- En DOM, un documento XML se representa como árbol y los distintos componentes del documento son nodos.

6.2 – DOM

ESTRUCTURA DE UN DOCUMENTO XML

- En el árbol DOM hay distintos tipos de nodos.
- Nodos para los elementos (clientes, cliente, apellidos, CP, validez)
- Nodos para los textos, identificados con #text, que contienen el valor para los elementos, incluyendo los que no tienen valor (clientes, cliente y validez)

6.2 – DOM

ESTRUCTURA DE UN DOCUMENTO XML

- Los atributos en DOM no son nodos del árbol. Están en una lista asociada al nodo de su elemento.
- Cada entrada de esta lista contiene el nombre el nombre de un atributo y su valor.
- Existen nodos #text que parecen no contener texto, pero que contendrán saltos de línea y espacios incluidos en el fichero para facilitar su lectura.
- Estos que estos nodos se podrán ignorar.

6.2 – DOM

PARSERS O ANALIZADORES, SERIALIZACIÓN Y DESERIALIZACIÓN

- Un *parser* o analizador sintáctico es un programa que permite verificar que un fichero está redactado conforme a la sintaxis del lenguaje.
- En nuestro caso y para XML, además permite acceder a los contenidos del fichero.
- Un *parser* DOM permite verificar, y en su caso obtener un documento XML

6.2 – DOM

PARSERS O ANALIZADORES, SERIALIZACIÓN Y DESERIALIZACIÓN

- El proceso de *parsing* se denomina a veces, *deserialización*, ya que a partir de un documento (una serie o secuencia de datos), se obtiene una estructura de datos no secuencial.
- El proceso de obtener una estructura secuencial, como un fichero, a partir de una no secuencial, se suele denominar *serialización*.

6.2 – DOM

PARSERS O ANALIZADORES, SERIALIZACIÓN Y DESERIALIZACIÓN

- Un *parser DOM* en Java es muy sencillo de utilizar.
- Con *DOM* se dispone de todo el documento en memoria con lo que permite un acceso muy rápido a sus contenidos.
- Con *DOM* se puede tanto consultar como modificar los contenidos de un documento.
- También permite serializar un árbol DOM (generar fichero con los contenidos.)

6.2 – DOM

PARSERS O ANALIZADORES, SERIALIZACIÓN Y DESERIALIZACIÓN

- Para archivos muy grandes, mantener todo el documento en memoria se puede hacer intratable.
- En estos casos se usa un parser que no tenga que almacenar todo el documento en memoria, como un *parser SAX*.
- Java incluye soporte para XML y varias tecnologías relacionadas en un API JAXP (Java API for XML Procesing), entre ellas DOM y SAX

6.2 – DOM

DOM con Java

- Todo lo necesario para trabajar con DOM está en el paquete org.w3c.dom (de la organización W3C)
- Un documento DOM puede crearse a partir de un documento XML, mediante parsing.
- También puede crearse directamente en memoria, empezando con un documento vacío y añadiendo elementos.

6.2 – DOM

DOM con Java

- Las principales interfaces de DOM son:
- **Document** – un documento XML (permite crear nuevos nodos).
- **Element** – un elemento en XML
- **Node** – cualquier nodo del documento.
- **NodeList** –lista con nodos hijos de un nodo
- **NamedNodeMap** – semejante a **NodeList** pero se usa para listas de atributos (**Attr**) donde se identifican por el nombre.
- **Attr** – Atributo de un nodo.
- **Text** – Datos carácter de un elemento.
- **CharacterData** – Datos carácter en el documento.
- **DocumentType** – Información en <!DOCTYPE>

6.2 – DOM

DOM con Java

- Las principales interfaces de DOM son:

Método	Funcionalidad
Métodos de la interfaz *Document*	
String getXmlEncoding()	Devuelve la codificación. Podría ser, por ejemplo “UTF-8”
String getXmlVersion()	Devuelve la versión de XML. Por ejemplo “1.0”

6.2 – DOM

DOM con Java

Método	Funcionalidad
Métodos de la interfaz *Node*	
short getNodeType()	Devuelve una constante para saber el tipo de nodo: (DOCUMENT_NODE (raíz del documento), ELEMENT_NODE (un elemento), TEXT_NODE (texto), …)
String getNodeName()	Devuelve el nombre de un nodo.
String getNodeValue()	Devuelve el valor de un nodo.
NodeList getChildNodes()	Devuelve lista de nodos hijos. Relevante para nodos de tipo ***Element***.
NamedNodeMap getAttributes()	Si el nodo es ***Element***, devuelve lista de atributos, sino **null**. Devuelve ***NamedNodeMap*** en lugar de ***NodeList*** porque el orden no es relevante. Se identifican por su nombre.
Node getParentNode()	Devuelve el nodo padre si existe, o null.

6.2 – DOM

Parsing DOM

- El paquete *javx.xml.parsers* proporciona las funcionalidades para DOM (también para SAX).
- La Clase *DocumentBuilder* implementa un parser DOM.
  - Pertenecen al paquete *javax.xml.parsers.DocumentBuilder*

- https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/DocumentBuilder.html

6.2 – DOM

Parsing DOM

- Se obtiene una instancia de *DocumentBuilder* con la clase *DocumentBuilderFactory* mediante el método *newInstance()*.
- A la vez, se obtiene una instancia de *DocumentBuilderFactory* mediante su método de clase *newDocumentBuilder()*.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setIgnoringComments(true);
dbf.setIgnoringElementContentWhitespace(true);

try {
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document domDoc = db.parse(new File(nomFich));
```

6.2 – DOM

Parsing DOM

- Métodos de *DocumentBuilderFactory*

Método	Funcionalidad
static DocumentBuilderFactory newInstance ()	Crea una instancia de la clase
DocumentBuilder newDocumentBuilder ()	Devuelve un *parser* DOM
void setIgnoringComments (boolean ignoreComm)	Ignora comentarios. Si no se ignoran, se incluyen como nodos dentro del árbol DOM.
void setIgnoringElementContentWhitespace (boolean ignoreComm)	Ignora textos vacíos en el contenido de los elementos si el *parser* está en modo de validación, sino no tiene efecto.

6.2 – DOM

Parsing DOM

- Métodos de *DocumentBuilder*

Método	Funcionalidad
Abstract Document newDocument ()	Devuelve un nuevo documento DOM vacío inicialmente.
Document parse (File f)	Construye un documento DOM a partir de los contenidos de un fichero (File)
Document parse (InputStream is)	… a partir de un InputStream
Document parse (String uri)	… a patir de un URI.

6.2 – DOM

Ejemplo DOM

- El siguiente programa obtiene un documento DOM a partir de un fichero XML y muestra sus contenidos.
- El *parsing* son cinco líneas de código para crear y configurar un *DocumentBuilderFactory*, crear un *DocumentBuilder* e invocar su método *parse()*.

```
57      public static void main(String[] args) {
58          String nomFich;
59          if (args.length < 1) {
60              System.out.println("Indicar por favor nombre de fichero");
61              return;
62          } else {
63              nomFich = args[0];
64          }
65
66          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setIgnoringComments(true);
            dbf.setIgnoringElementContentWhitespace(true);
68
69
70          try {
71              DocumentBuilder db = dbf.newDocumentBuilder();
72              Document domDoc = db.parse(new File(nomFich));
73              muestraNodo(domDoc, 0, System.out);
74          } catch (FileNotFoundException | ParserConfigurationException
75                  | SAXException e) {
76              System.err.println(e.getMessage());
            } catch (Exception e) {
                e.printStackTrace();
79          }
80      }
81  }
```

6.2 – DOM

Ejemplo DOM

- El grueso del programa está en el método *muestaNodo*.
- Para mostrar el árbol entero se le pasa el nodo raíz.
- El parámetro *nivel* se usa para mostrar los nodos con la indentación apropiada.
- Una vez mostrado el nodo se obtiene una lista de nodos hijo, y se llama nuevamente a *muestraNodo* para cada uno, incrementando *nivel* en uno.
- *muestraNodo* escribe a un *PrintStream* que se le pasa como parámetro, esto permite de manera sencilla dirigir la salida a un fichero cualquiera, en lugar de a *Sytem.out*.

```
20  public static void muestraNodo(Node nodo, int nivel, PrintStream ps) {
21      if (nodo.getNodeType() == Node.TEXT_NODE) { // Ignora textos vacíos
22          String text = nodo.getNodeValue();
23          if (text.trim().length() == 0) {
24              return;
25          }
26      }
27      for (int i = 0; i < nivel; i++) {    // Indentación
28          ps.print(INDENT_NIVEL);
29      }
30      switch (nodo.getNodeType()) {  // Escribe información de nodo según tipo
31          case Node.DOCUMENT_NODE:  // Documento
32              Document doc = (Document) nodo;
33              ps.println("Documento DOM, versión: " + doc.getXmlVersion()
34                      + ", codificación: " + doc.getXmlEncoding());
35              break;
36          case Node.ELEMENT_NODE:    // Elemento
37              ps.print("<" + nodo.getNodeName());
38              NamedNodeMap listaAtr = nodo.getAttributes();  // Lista atributos
39              for (int i = 0; i < listaAtr.getLength(); i++) {
40                  Node atr = listaAtr.item(i);
41                  ps.print(" @" + atr.getNodeName() + "[" + atr.getNodeValue() + "]");
42              }
43              ps.println(">");
44              break;
45          case Node.TEXT_NODE:    // Texto
46              ps.println(nodo.getNodeName() + "[" + nodo.getNodeValue() + "]");
47              break;
48          default:    // Otro tipo de nodo
49              ps.println("(nodo de tipo: " + nodo.getNodeType() + ")");
50      }
51      NodeList nodosHijos = nodo.getChildNodes();    // Muestra nodos hijos
52      for (int i = 0; i < nodosHijos.getLength(); i++) {
53          muestraNodo(nodosHijos.item(i), nivel + 1, ps);
54      }
55  }
```

6.2 – DOM

Ejemplo DOM

- Resto de código de ejemplo

```
// Parsing DOM y visualización de documento DOM generado
package dom_parser;

import java.io.File;
import java.io.PrintStream;
import java.io.FileNotFoundException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.NamedNodeMap;
import org.xml.sax.SAXException;

public class DOM_Parser {

    private static final String INDENT_NIVEL = "  ";  //
```

```
public static void muestraNodo(Node nodo, int nivel, PrintStream ps) {
    if (nodo.getNodeType() == Node.TEXT_NODE) { // Ignora textos vacíos
        String text = nodo.getNodeValue();
        if (text.trim().length() == 0) {
            return;
        }
    }
    for (int i = 0; i < nivel; i++) {    // Indentación
        ps.print(INDENT_NIVEL);
    }
    switch (nodo.getNodeType()) {  // Escribe información de nodo según tipo
        case Node.DOCUMENT_NODE:  // Documento
            Document doc = (Document) nodo;
            ps.println("Documento DOM, versión: " + doc.getXmlVersion()
                    + ", codificación: " + doc.getXmlEncoding());
            break;
        case Node.ELEMENT_NODE:    // Elemento
            ps.print("<" + nodo.getNodeName());
            NamedNodeMap listaAtr = nodo.getAttributes();  // Lista atributos
            for (int i = 0; i < listaAtr.getLength(); i++) {
                Node atr = listaAtr.item(i);
                ps.print(" @" + atr.getNodeName() + "[" + atr.getNodeValue() + "]");
            }
            ps.println(">");
            break;
        case Node.TEXT_NODE:    // Texto
            ps.println(nodo.getNodeName() + "[" + nodo.getNodeValue() + "]");
            break;
        default:    // Otro tipo de nodo
            ps.println("(nodo de tipo: " + nodo.getNodeType() + ")");
    }
    NodeList nodosHijos = nodo.getChildNodes();    // Muestra nodos hijos
    for (int i = 0; i < nodosHijos.getLength(); i++) {
        muestraNodo(nodosHijos.item(i), nivel + 1, ps);
    }
}
```

6.2 – DOM

Ejemplo DOM

- Salida del programa junto con fichero xml de entada.

```
run:
Documento DOM, versión: 1.0, codificación: UTF-8
  <clientes>
    <cliente @DNI[78901234X]>
      <apellidos>
        #text[NADALES]
      <CP>
        #text[44126]
    <cliente @DNI[89012345E]>
      <apellidos>
        #text[ROJAS]
      <validez @estado[borrado] @timestamp[1528286082]>
    <cliente @DNI[56789012B]>
      <apellidos>
        #text[SAMPER]
      <CP>
        #text[29730]
BUILD SUCCESSFUL (total time: 0 seconds)
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<clientes>
    <cliente DNI="78901234X">
        <apellidos>NADALES</apellidos>
        <CP>44126</CP>
    </cliente>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
    <cliente DNI="56789012B">
        <apellidos>SAMPER</apellidos>
        <CP>29730</CP>
    </cliente>
</clientes>
```

6.2 – DOM

ACTIVIDADES PROPUESTAS

a) Modifica el programa anterior para que escriba la salida a un fichero con nombre parsing_dom.txt. Como mejora puedes utilizar la clase SimpleDateFormat para hacer que en el nombre de fichero se incluyan la fecha y la hora, de manera similar a como se hacía cuando trabajábamos con ficheros.

6.2 – DOM

Creación de documentos con DOM

- Se trata de crear desde cero un documento DOM, y escribirlo a un fichero XML.
- Se pueden crear nodos de distintos tipos con algunos métodos de la interfaz *Document*.
- Se usarán métodos de la interfaz *Node* para añadirlos al documento.

6.2 – DOM

Creación de documentos con DOM

- Métodos de la interfaz *Document* par crear nuevos nodos.

Método	Funcionalidad
Element createElement (String nombreEtiqueta)	Crea nuevo elemento con el nombre dado.
Text createTextNode (String texto)	Crea nuevo nodo de tipo texto

6.2 – DOM

Creación de documentos con DOM

- Los atributos se añaden con métodos de la interfaz *Element*.

Método	Funcionalidad
void setAttribute (String name, String value)	Añade atributo con el valor dado.

6.2 – DOM

Creación de documentos con DOM

- Algunos métodos de la interfaz *Node*.
- Suelen lanzar la excepción *DOMException* en caso de fallo.

Método	Funcionalidad
Node appendChild (Node nuevoHijo)	Añade nodo como último nodo hijo, y devuelve el nodo añadido.
Node insertBefore (Node nuevoHijo, Node refHijo)	Añade nuevo hijo antes del indicado.

6.2 – DOM

Creación de documentos con DOM

- Vamos a ver como Serializar un documento DOM (generar un documento de XML textual a partir de él).
- El siguiente programa crea un documento DOM con parte de los contenidos del documento de XML de ejemplo.
- Después lo serializa en un string mediante un ***StringWriter***.

```
1  // Creación de árbol DOM añadiendo elementos y serialización una vez creado
2  package creardocumentodom;
3
4  import java.io.StringWriter;
5
6  import org.w3c.dom.Document;
7  import org.w3c.dom.Element;
8
9  import javax.xml.parsers.DocumentBuilderFactory;
10 import javax.xml.parsers.DocumentBuilder;
11 import javax.xml.parsers.ParserConfigurationException;
12
13 import javax.xml.transform.dom.DOMSource;
14 import javax.xml.transform.TransformerFactory;
15 import javax.xml.transform.Transformer;
16 import javax.xml.transform.OutputKeys;
17 import javax.xml.transform.stream.StreamResult;
18
19 public class CrearDocumentoDOM {
20
21     public static void main(String[] args) {
22
23         try {
24             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

6.2 – DOM

Creación de documentos con DOM

- La clase *Transformer* permite serializar documentos DOM.
- Se obtienen instancias de *Transformer* mediante el método *newTransformer()* de la clase *TransformerFactory*.
- Las instancias de *TransformerFactory* se obtienen, a su vez, mediante su método *newInstance()*.
- Se puede controlar las opciones para la serialización mediante setOutputProperty.
- Se realiza la serialización mediante el método *transform (Source xmlSource, Result outputTarget)*.

```
21  public static void main(String[] args) {
22
23      try {
24          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
25          DocumentBuilder db = dbf.newDocumentBuilder();
26          Document doc = db.newDocument();
27          doc.setXmlVersion("1.0");
28
29          Element elClientes = doc.createElement("clientes");
30          doc.appendChild(elClientes);
31
32          Element elCliente = doc.createElement("cliente");
33          elCliente.setAttribute("DNI", "89012345E");
34          Element elApell = doc.createElement("apellidos");
35          elApell.appendChild(doc.createTextNode("ROJAS"));
36          elCliente.appendChild(elApell);
37          Element elValidez = doc.createElement("validez");
38          elValidez.setAttribute("estado", "borrado");
39          elValidez.setAttribute("timestamp", "1528286082");
40          elCliente.appendChild(elValidez);
41
42          elClientes.appendChild(elCliente);
43
44          DOMSource domSource = new DOMSource(doc);
45          Transformer transformer = TransformerFactory.newInstance().
46                  newTransformer();
47          transformer.setOutputProperty(OutputKeys.METHOD, "xml");
48          transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
49          transformer.setOutputProperty(OutputKeys.INDENT, "yes");
50          transformer.setOutputProperty(
51                  "{http://xml.apache.org/xslt}indent-amount", "4");
52
53          StringWriter sw = new StringWriter();
            StreamResult sr = new StreamResult(sw);
55          transformer.transform(domSource, sr);
56          System.out.println(sw.toString());
57      } catch (ParserConfigurationException e) {
58          System.err.println(e.getMessage());
        } catch (Exception e) {
            e.printStackTrace();
61      }
```

6.2 – DOM

Creación de documentos con DOM

- Salida del programa junto con fichero xml de entada.

```
run:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<clientes>
    <cliente DNI="89012345E">
        <apellidos>ROJAS</apellidos>
        <validez estado="borrado" timestamp="1528286082"/>
    </cliente>
</clientes>

BUILD SUCCESSFUL (total time: 1 second)
```

6.2 – DOM

ACTIVIDADES PROPUESTAS

b) Crea un programa que realice el parsing DOM del fichero de ejemplo con los datos de varios clientes. Una vez hecho esto, debe buscar un cliente cualquiera por DNI y cambiarle los apellidos. Después debe hacer una copia del fichero añadiendo al nombre ".bak". Seguido, debe utilizar la clase **Transformer** para serializar el documento DOM al fichero, sobreescribiendo sus contenidos. Tanto el DNI como el nuevo valor para apellidos se pueden asignar en variables de programa.

6.3 – SAX

6.3 – SAX

- Acceso a datos con SAX (Simple API for XML) es otra tecnología para poder acceder a XML desde lenguajes de programación.
- Aunque SAX tiene el mismo objetivo que DOM, esta aborda el problema desde una óptica diferente.
- Por lo general se usa SAX cuando la información almacenada en los documentos XML es clara, está bien estructurada y no se necesita hacer modificaciones.

6.3 – SAX

- Las principales características de SAX son:
  - Alternativa para leer documentos XML de manera secuencial.
  - El documento solo se lee una vez. (A diferencia de DOM, el programador no se puede mover por el documento a su antojo.)
  - Una vez que se abre el documento, se recorre secuencialmente el fichero hasta el final. Cuando llega al final se termina el proceso.
  - SAX, a diferencia de DOM, no carga el documento en memoria, sino que lo lee directamente desde el fichero. Esto lo hace especialmente útil cuando el fichero XML es muy grande.

6.3 – SAX

- SAX sigue los siguientes pasos básicos:
  1. Se le dice al *parser* SAX qué fichero quiere que sea leído de manera secuencial.
  2. El documento XML es traducido a una serie de eventos.
  3. Los eventos generados pueden controlarse con métodos de control llamados *callbacks*.

6.3 – SAX

- El proceso se puede resumir de la siguiente manera:
  - SAX abre un archivo XML y coloca un puntero en al comienzo del mismo.
  - Cuando comienza a leer el fichero, el puntero va avanzando secuencialmente.
  - Cuando SAX detecta un elemento propio de XML entonces lanza un evento. Un evento puede deberse a:
    - Que SAX haya detectado el comienzo del documento XML.
    - Que se haya detectado el final del documento XML.
    - Que se haya detectado una etiqueta de comienzo de un elemento.
    - Que se haya detectado una etiqueta de final de un elemento.
    - Que se haya detectado un atributo.
    - Que se haya detectado una cadena de caracteres que puede ser un texto.
    - Que se haya detectado un error (en el documento, de I/O, etc.).
  - Cuando SAX devuelve que ha detectado un evento, entonces este evento puede ser manejado. Esta clase puede ser extendida y los métodos de esta clase pueden ser redefinidos (sobrecargados) por el programador para conseguir el efecto deseado cuando SAX detecta los eventos.
  - Cuando SAX detecta un evento de error o un final de documento entonces se termina el recorrido

RESUMEN

- ✓ XML es un leguaje formal. XML es un estándar del W3C. Los documentos en lenguaje XML son documentos de texto con estructura jerárquica que se utilizan para guardar información de todo tipo.
- ✓ DOM es un modelo que define interfaces de programación para acceder a los contenidos de un documento de XML, y para modificarlos. En el modelo DOM un documento SML se representa como un árbol con distintos tipos de nodos.
- ✓ Un *parser* o analizador sintáctico para un lenguaje formal como XML permite verificar que un documento es conforme a la sintaxis del lenguaje y, si es el caso, proporciona acceso a sus contenidos.

RESUMEN

- ✓ Un *parser* DOM permite obtener un árbol DOM a partir de un documento de XML.
- ✓ Un *parser* SAX es un tipo de *parser* para XML basado en eventos. No devuelve una estructura de datos como un *parser* DOM. En lugar de ello, genera diversos tipos de eventos conforme va analizando el documento, y estos eventos pueden gestionarse en métodos manejadores de eventos.
- ✓ La validación de un documento XML consiste en verificar que es conforme a un conjunto de restricciones adicionales relativas a los elementos que contiene, sus atributos y los valores para todos ellos. Estas restricciones se pueden especificar en DTD y en esquemas de XML. Se puede validar un documento de XML durante el *parsing* y en ese caso el *parser* lanzará excepciones ante cualquier error de validación.

Acceso a Datos

FIN DE LA UNIDAD

GRACIAS