David Torres Aranha

Actividad 1.11: Tratamiento ficheros XML

Investiga sobre las clases de DOM para el tratamiento de ficheros XML

¿Para qué se utiliza DOM en JAVA?

DOM (Document Object Model) es una API estándar del W3C que permite leer, recorrer, modificar y crear documentos XML desde Java. Su funcionamiento consiste en cargar todo el fichero XML en memoria y representarlo como un árbol jerárquico de nodos (elementos, atributos, texto, comentarios...), de modo que se pueda acceder a cualquier parte del documento de forma directa.

Su principal ventaja es la facilidad para navegar y manipular el documento. Su principal inconveniente es el consumo de memoria con ficheros muy grandes, ya que se carga el documento completo (para esos casos existen alternativas como SAX o StAX). En Java se encuentra en el paquete `org.w3c.dom`, junto con `javax.xml.parsers` para el análisis y `javax.xml.transform` para la escritura.

¿En qué otros ámbitos, además de en programación con JAVA, se utiliza el enfoque DOM?

DOM es un estándar independiente del lenguaje y de la plataforma, por lo que se utiliza en muchos otros ámbitos:

- **Desarrollo web (JavaScript):** el navegador construye el DOM de las páginas HTML y JavaScript lo manipula (`document.getElementById()`, `createElement()`, `appendChild()`...).
- **Otros lenguajes de programación:** Python (`xml.dom.minidom`), C# / .NET (`XmlDocument`), PHP (`DOMDocument`), C++, Ruby, etc.
- **Documentos HTML y XHTML**, además de XML.
- **Pruebas automatizadas de interfaces web** (por ejemplo, Selenium), que localizan elementos recorriendo el DOM.
- **Documentos SVG**, que también se manipulan mediante DOM.

Lectura de ficheros XML con DOM:

¿Cuáles son las clases principales utilizadas para leer y cargar un fichero XML usando DOM?

- `DocumentBuilderFactory` (`javax.xml.parsers`): fábrica que se utiliza para obtener el constructor del parser (`DocumentBuilderFactory.newInstance()`).
- `DocumentBuilder` (`javax.xml.parsers`): analiza el fichero XML mediante el método `parse(File)` y genera el documento.
- `Document` (`org.w3c.dom`): representa el documento XML completo cargado en memoria (el árbol DOM). Con `getDocumentElement()` se obtiene el elemento raíz.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(new File("informacion-IES.xml"));
```

¿Qué clases se usan para acceder a los nodos asociados a las etiquetas del fichero XML? ¿Y para acceder a la información que tienen?

Para acceder a los nodos de las etiquetas:

- `Node`: interfaz base de cualquier componente del árbol. Métodos habituales: `getNodeName()`, `getNodeType()`, `getChildNodes()`.
- `Element`: representa una etiqueta del XML. Permite buscar etiquetas con `getElementsByTagName("nombre")`.
- `NodeList`: lista de nodos que devuelven métodos como `getElementsByTagName()` o `getChildNodes()`. Se recorre con `getLength()` e `item(i)`.

Para acceder a la información que contienen:

- `Element`: `getAttribute("nombre")` devuelve el valor de un atributo.
- `Node`: `getTextContent()` devuelve el texto contenido en una etiqueta; `getNodeValue()` devuelve el valor del nodo (útil en nodos de texto y atributos).
- `Attr` y `NamedNodeMap`: representan los atributos de un elemento (`getAttributes()` devuelve un `NamedNodeMap` con objetos `Attr`).
- `Text`: nodo que contiene el texto de una etiqueta.

Escritura de ficheros XML con DOM:

¿Cuáles son las clases principales utilizadas para escribir un fichero XML?

Primero se crea el documento en memoria y después se vuelca a un fichero:

- `DocumentBuilderFactory` y `DocumentBuilder`: el método `newDocument()` crea un `Document` vacío.
- `Document`: se construye el árbol XML dentro de él.
- `TransformerFactory` (`javax.xml.transform`): fábrica para crear el transformador.
- `Transformer`: realiza la transformación del árbol DOM a un fichero XML (`transform(source, result)`). Permite configurar la salida con `setOutputProperty(OutputKeys.INDENT, "yes")`.
- `DOMSource`: origen de la transformación; encapsula el `Document` en memoria.
- `StreamResult`: destino de la transformación (un fichero, un `File` o la consola).

```
Transformer transformer = TransformerFactory.newInstance().newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.transform(new DOMSource(doc), new StreamResult(new
File("salida.xml")));
```

¿Qué clases se usan para crear una etiqueta, el contenido asociado a una etiqueta o el contenido asociado a un atributo?

Todos estos elementos se crean mediante métodos de la clase `Document`, y después se enlazan al árbol con `appendChild()`:

- **Crear una etiqueta:** `doc.createElement("nombre")` devuelve un objeto de tipo `Element`.
- **Crear el contenido de una etiqueta:** `doc.createTextNode("texto")` devuelve un objeto de tipo `Text`. También se puede usar `elemento.setTextContent("texto")`.
- **Crear un atributo y su contenido:** `elemento.setAttribute("nombre", "valor")` o, de forma explícita, `doc.createAttribute("nombre")` que devuelve un objeto `Attr`, al que se le asigna el valor con `setValue("valor")` y se añade con `elemento.setAttributeNode(attr)`.

```
Element instituto = doc.createElement("instituto");
instituto.setAttribute("nombre", "IES Virgen de la Paloma");
Element titulo = doc.createElement("titulo");
titulo.appendChild(doc.createTextNode("Programación"));
instituto.appendChild(titulo);
doc.appendChild(instituto);
```