Enfoque adoptado a la resolución de la Tarea 2.2

Prueba que releva las nociones de procesado de datos adquiridas en la tarea 2, continuamos con acceso a ficheros. Esta vez vamos a crear archivos XML.

En este caso vamos a volver a trabajar con el mismo fichero de datos, concretamente con los ficheros generados en la tarea 2. Dado que el XML es algo que nunca había visto, he pasado por un pequeño proceso de aprendizaje en el que he visto como es la estructura de los archivos XML, a parte de saber que **NO** es un lenguaje de marcas, si no un **meta-lenguaje,** nos permite definir lenguajes de marcado adecuados a usos determinados. Muchas personas creen que el XML viene del HTML pero realmente es justo lo contrario al ser un SGML Standard Generalized Markup Language).

Entonces como he dicho el XML sirve para representar información estructurada de modo que esta información pueda ser almacenada, procesada, visualizada… Ahora es cuando doy paso al ejercicio:

Enfoque:

Además de generar un TXT por cada bloque de información recibido, debo generar un archivo XML para cada fichero TXT generado, estos ficheros XML siguen una estructura especificada sobre papel y debo de aprender a tratar este tipo de ficheros.

También he visto que como en el HTML, XML puede tener DOM y que JAVA puede trabajar el XML mediante el DOM. Aquí entran en juego algunas clases que voy a aprender a mirar cómo **Document, DocumentBuilder,** y alguna que otra más, y a tratar un DOM desde Java, así cómo crear atributos, etiquetas, asignar valores... del XML.

Mi objetivo es crear archivos XML haciendo uso del DOM, sin embargo es necesario crear el documento con la **clase DocumentBuilder**, para después crear cada elemento con la **clase Element**. Finalmente uso la **clase Transformer** para generar un archivo de texto con el contenido del XML.

Ahora mi problema es "**dinamizar**" la forma en la que se generan los archivos ya que voy a rellenar los archivos XML en base a una serie de ficheros y cada uno puede tener más o menos información.

Parsers

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

Mi gran duda era afrontar como generar una estructura XML sin ningún elemento "puente" o algo que me diera las herramientas para programar su estructura.

He leido que hay parsers que trabajan también bajo los esquemas de un DOM y es lo que he aplicado a esta práctica.

Primera etapa: Elaboración del DOM, para hacer esto he usado una serie de librerías que implementan esto, ahora hablo de ellas:

Después de haber creado una instancia de DocumentBuilder y DocumentBuilderFactory:

1. Declaro un DOM con el método docBuilder.**newDocument**(): Este método obtiene una nueva instancia de un objeto de documento DOM para construir nuestro árbol DOM.

2. Una vez tenemos nuestro lienzo (documento DOM) para crear su estructura lo primero que debemos hacer es **crear** el elemento (etiqueta) RAIZ del cual van a salir el resto de etiquetas, en mi caso ese elemento/etiqueta es <FICHERO>, y esto lo escribo en forma de sentencia de la siguiente manera:

   Element fichero = documento.**createElement**("FICHERO");

   Nota: Como indicamos en negrita, lo único que hemos hecho es **crear** la etiqueta, aún no hemos indicado que sea para nuestro documento DOM, eso lo haremos cuando finalicemos por completo la estructura de la etiqueta, lo cual vamos a trabajar ahora en el siguiente paso.

3. La etiqueta <FICHERO> posee atributos según las especificaciones de la tarea, estos funcionan como elemento identificador e informativo de la misma, para crear un atributo escribimos las siguientes sentencias:

```
// atributos de la etiqueta fichero
//FECHA_CREACION (primero genero la fecha)
Date date = new Date(); //objeto de tipo fecha
DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy"); //objeto formateador de fechas

Attr atribFechaCreacion = documentoPrueba.createAttribute("FECHA_CREACION");
atribFechaCreacion.setValue(dateFormat.format(date));
fichero.setAttributeNode(atribFechaCreacion);
```

   Vamos a fijarnos concretamente en las líneas que he subrayado de amarillo:
   - Con **Attr** creamos un atributo.
   - Con .**setValue**(valor) decimos que valor llevará ese atributo (la variable que lleve cambiará o no dependiendo de si pretendemos dinamizar la información de ese fichero generado aún más).
   - Y por último con <etiqueta>.**setAttributeNode**(atributoQueHemosCreado) decimos que ese atributo pertenece a esa etiqueta, cosa que aún no habíamos indicado y que se debe de aplicar una vez hemos configurado ese atributo y su valor, de ahí el orden de realización.

4. Y para finalizar con la creación de la etiqueta fichero solo nos queda indicar una cosa (la cual hemos mencionado en la **nota** del punto 2): Declaramos que fichero hereda del documento DOM, al heredar directamente del documento y no de una etiqueta "padre" <FICHERO> se convierte en etiqueta raíz. Esto traducido a sentencia sería algo como:
   - documentoDOM.**appendChild**(fichero);

Basta con que repitamos estos 4 pasos a modo de procedimiento para el resto de etiquetas pero heredando esta vez de la etiqueta **fichero** o un "hijo" de fichero creado con anterioridad.

Segunda etapa: Obtención de datos

Ya tengo generada mi estructura XML pero me falta algo igual de importante: los datos con los que rellenaré los ficheros XML generados y la dinamización de las etiquetas para incluir varios CAMPOS, tras pensar un poco vi acertado el uso de un bucle que reformulara la sentencia con un valor diferente y la agregara al DOM, al final esto dió resultado:

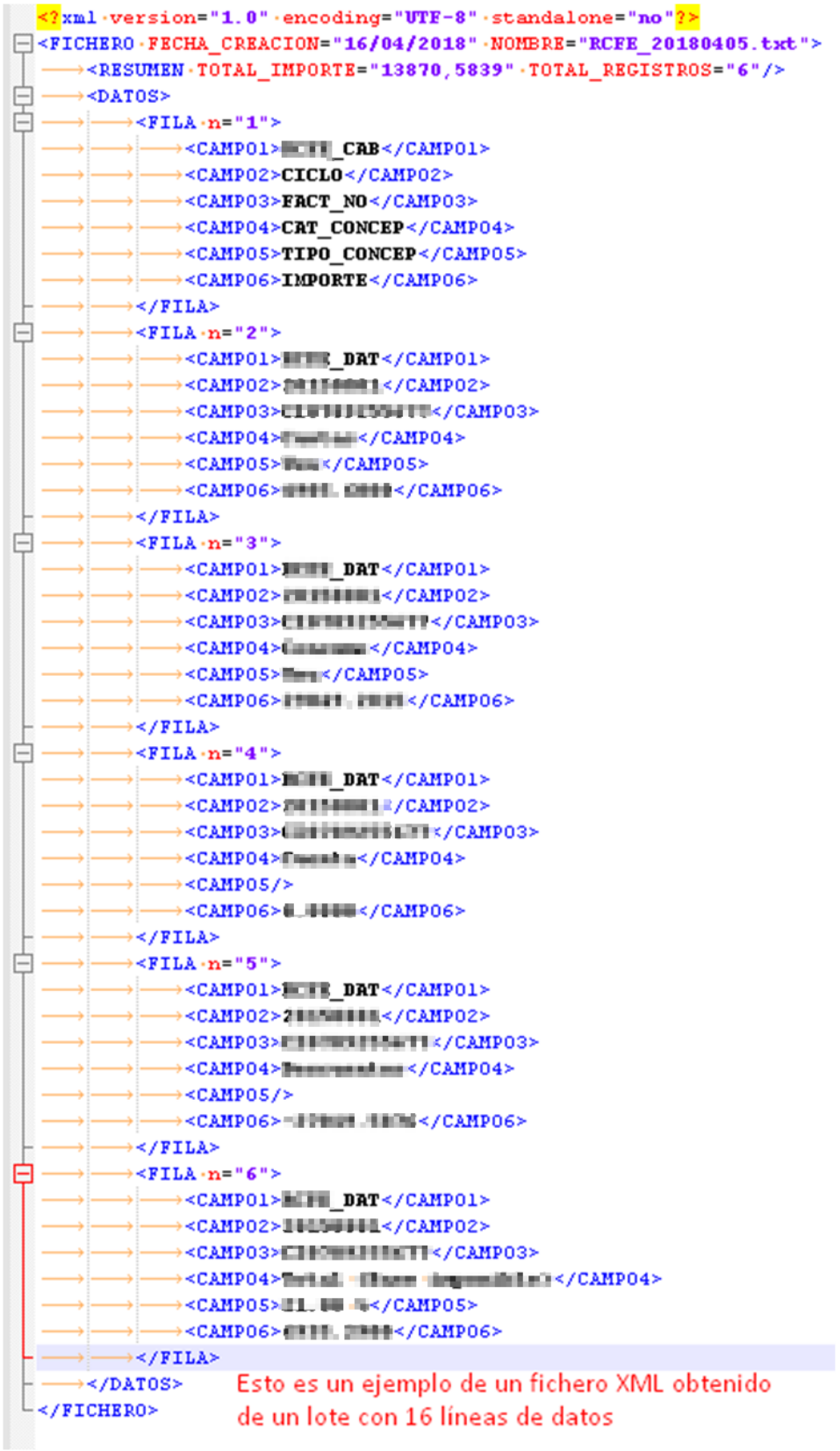
```
/*================== CAMPO ==================*/
// Declaramos elemento (etiqueta) <CAMPO></CAMPO> (<CAMPO> hereda de <FILA>)
for (int q = 0; q < palabras.length; q++) {
  listaPalabras.add(palabras[q]);

  Element campo = documentoPrueba.createElement("CAMPO" + (q + 1));
  campo.appendChild(documentoPrueba.createTextNode(listaPalabras.get(q)));
  //campo hereda de fila, aquí lo decimos.
  fila.appendChild(campo);
}
```

Con esta implementación conseguí lo que quería, los valores los obtiene de una lista (listaPalabras en la imagen superior) en las que previamente he almacenado todos los valores de cada línea mediante el método .**split**() aprendido en la primera tarea.

Finalmente uso la **clase Transformer** para generar un archivo de texto con el contenido del XML.

```
// escribimos el contenido en un archivo .xml
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(documentoPrueba);

String nombreFichero = s.getName().substring(0, 5) + ciclo + s.getName().substring(4, s.get

StreamResult result = new StreamResult(new File(rutaDir + "/" + nombreFichero + ".xml"));
//(s.getName().length() - 4)
//StreamResult result = new StreamResult(new File("archivo.xml"));

// Si se quiere mostrar por la consola...
// StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);

System.out.println("- " + s.getName() + " guardado en TXT y XML!");
```

Resultado final:



Como se aprecia cada atributo posee un valor el cual es dinámico y calculado durante la elaboración del XML. Aquí menciono cuales son y si tengo algo que objetar:

- **FECHA_CREACION="17/04/2018"** - Fecha actual y su formato (DD/MM/AAAA) obtenido gracias a tres librerías:
  - java.util.Date;
  - java.text.DateFormat;
  - java.text.SimpleDateFormat;

- **NOMBRE="RCFE_20180417.txt"** - El nombre del fichero lo obtengo con la clase **File** y el método **.getName()** que me da un String del nombre del fichero completo.

- **TOTAL_IMPORTE="13870,5839"** - Esto es algo más laborioso ya que he tenido que hacer un buffer específico que me usara el método **.split( )** en cada línea, y me buscara en la **cabecera** (**primera línea**) si existe la palabra **IMPORTE** como columna de cabecera, en ese caso me debe decir en qué posición de la cabecera se encuentra lo cual lo obtiene con el método **.indexOf()** ya que previamente toda esa información la almacenaba en una lista. De no existir la palabra **IMPORTE** NO me crearía el atributo, en caso de existir, ya solo le quedaría sumar cada valor sobre una variable que recibe exactamente las cifras que yo previamente le he pasado **gracias a que tengo el índice de su posición en la lista,** todo esto es obtenido de forma automática. Por último cabe decir que ese valor **TOTAL_IMPORTE** generalmente no me da una cifra con pocos decimales, así que como paso final uso esta librería:
  - java.text.DecimalFormat
  - Esta librería me formatea el valor para que se redondee a **4 decimales** o a los que yo previamente le haya indicado (**#.####**).

- **TOTAL_REGISTROS="6"** - Esta variable también la he tenido que obtener como TOTAL_IMPORTE (aislando un bucle que me obtuviera solamente ese valor en concreto), ya que he tenido que hacer un buffer específico que me contara todas las líneas del fichero antes de leer y procesar toda su información.

- **n="1"** - Variable autoincrementada cada vez que se lee una nueva línea, en cada iteración pongo su valor.

Como última aclaración, cada XML debía llevar un nombre prediseñado que consiste en poner **tres datos entre "_"** por ejemplo **AVPG_20150801_20180417**.xml :

- **AVPG** - Extensión del bloque objetivo obtenido mediante **.substring()** leyendo la primera línea del fichero.
- **20150801** - Esto es el **CICLO** el cual se encuentra dentro del bloque, el lote entero posee el mismo ciclo por lo que dedicando un buffer específico lo obtengo de la segunda línea del fichero (Ya que la primera es el **CAB**), lo guardo como variable y lo uso a posteriori para ponerselo a todos los ficheros XML generados.
- **20180417** - Fecha actual en formato AAAA/MM/DD obtenida con las librerías:
  - java.util.Date;
  - java.text.DateFormat;
  - java.text.SimpleDateFormat;

Nada más que decir de momento.