**Práctica XML**

# **Introducción**

## **JDOM**

La práctica pide procesar archivos XML con Java. Pese a que XML ha sido sustituido en gran parte por JSON, XML aún sigue siendo la primera opción en algunos escenarios.

Para realizarla, se usará la librería JDOM de Java, que permite un rápido desarrollo de aplicaciones XML. JDOM puede usarse como alternativa al paquete *org.w3c.dom* para manipular programáticamente documentos XML. No es un un reemplazo a DOM, ya que pueden coexistir.

Para entender el por qué de la necesidad de JDOM, se han de ver las limitaciones de W3C DOM:

* Independencia del lenguaje: DOM no fue pensado para Java, y trabajar con él en Java puede ser complejo.
* Jerarquías estrictas: La API DOM sigue directamente la especificación XML. En XML, todo es un nodo, por lo que la interfaz está basada en nodos casi todos los método retornan un nodo. Esto hace que trabajar conjuntamente con Java sea complejo.
* Interface driven: La API DOM consiste en interfaces únicamente (con la excepción de la clase *Exception*).

En contraste, las ventajas de usar JDOM son:

* JDOM es específica para Java: La API usa la clase incluida en Java *String* siempre que sea posible, de forma que el texto siempre está disponible como String.
* Sin jerarquías: En JDOM, un elemento XML es una instancia de *Element*, un atributo XML es una instancia de *Attribute* y un documento XML en sí es una instancia de *Document*.
* Class driven: A causa de que los objetos de JDOM son instancias directas de clases como *Document*, *Element* y *Attribute*, crear uno es tan fácil como usar el operador *new*.

Crear un documento es muy simple:

Element carElement = new Element("car");

Document myDocument = new Document(carElement);

Añadir un atributo puede realizarse de la siguiente manera:

carElement.addAttribute(new Attribute("vin", "123fhg5869705iop90"));

Añadir elementos también es muy simple:

Element make = new Element("make");

make.addContent("Toyota");

carElement.addContent(make);

Un elemento puede tener un comentario:

carElement.addContent(new Comment("Description of a car"));

Se puede acceder a un elemento hijo con:

Element yearElement = carElement.getChild("year");

Y se puede eliminar un elemento hijo con:

boolean removed = carElement.removeChild("year");

## **XLST**

XLST (Extensible Stylesheet Language Transformations) es un lenguaje para transformar documentos XML a otros documentos XML u otros formatos como HTML.

El procesador XSLT toma uno o más documentos fuente XML, uno o más documentos plantilla XSLT y los procesa para obtener un documento resultante mediante el paradigma básico de *pattern-matching*.

# **Acciones**

Para la realización de la práctica se han tenido que implementar varias acciones. Entre las cuales encontramos:

* Reset: crea un documento XML inicializado.
* New: pide los datos al usuario y añade un alquiler al fichero XML.
* List: muestra por pantalla el fichero XML .
* Xslt: transforma el fichero XML a HTML usando una plantilla xslt.

## **Reset**

Esta función es la encargada de crear (o vaciar e inicializar) el documento XML que contendrá todos los datos para la práctica.

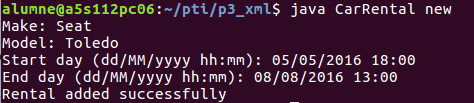
Para ello, se llama a la función createDocument(), la cual crea un elemento llamado carrental, y a partir de este se crea el documento. Una vez creado el documento, este se pasa a la función outputDocumentToFile(Document) que usando XMLOtputter, escribe el documento JDOM en el ficher carRental.xml.

carRentalReset.png

**Figura1: Ejecución de la llamada reset**

## **New**

La función *“addNewRental()”* será la encargada de ralizar la tarea de pedir los datos necesarios al usuario para realizar un nuevo alquiler (Fabricante, Modelo, Fecha de inicio del alquiler y fecha de fin del alquiler). Se ayuda al usuario a introducir los datos poniendo, por ejemplo, el formato de fecha esperado. Una vez se hayan introducido las dos fechas, se comprueba que sean correctas (es decir, que la fecha de inicio del alquiler sea anterior a la fecha de fin del alquiler).

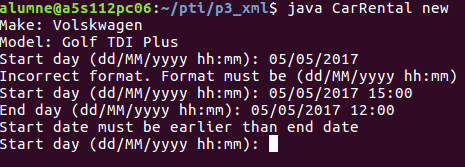


## 

**Figura 2: Ejecución de la llamada new y entrada de datos correctos**

# El programa pedirá cada una de las dos fechas todas las veces que sean necesarias hasta que el usuario consiga introducir dos correctas. En cada error, el programa imprimirá por pantalla un mensaje indicando al usuario que el formato indicado no es correcto, mostrando además el formato correcto. dateErrorFormat.png

**Figura 3: Mensaje de error indicando un error en el formato de la fecha introducida y mostrando el formato correcto**



**Figura 4: Mensaje de error indicando que la fecha inicial debe ser anterior a la final**

Para introducir todos los datos de forma adecuada, se llamará a una función *“askData()”*, que creará un elemento “*rental”*. A este elemento se le añadirá un atributo *“ID”* que se obtendrá de la siguiente forma:

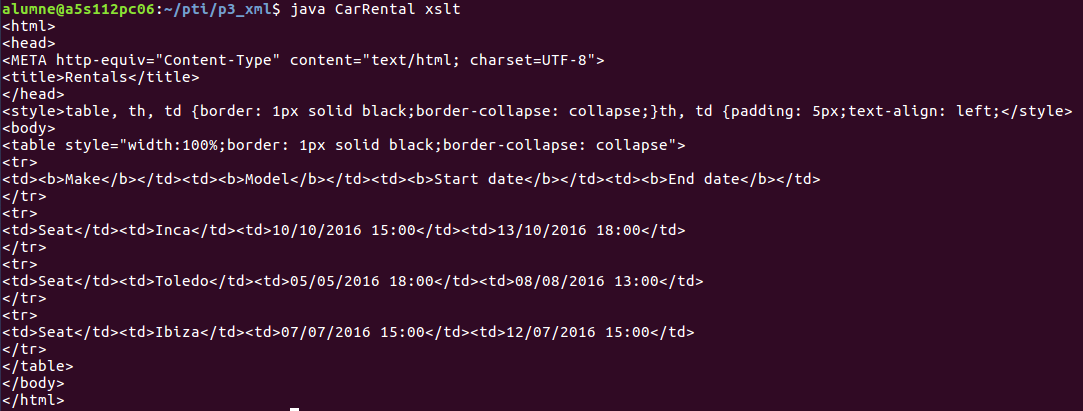
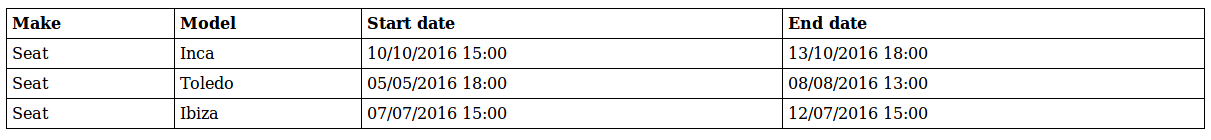
long epoch = System.currentTimeMilis()/1000;

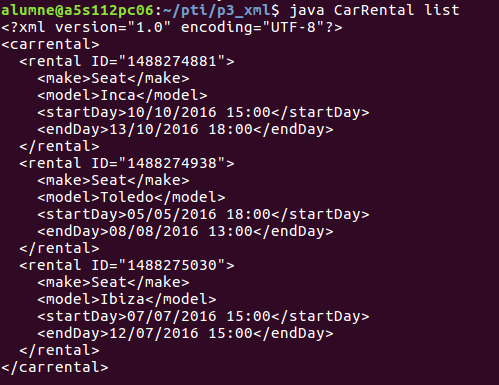
Se crea un elemento en el que se guarda la información de cada parámetro del alquiler (Fabricante, Modelo, etc). Estos se guardarán como elementos hijos del elemento *“rental”*. Una vez asignados todos, se retornará el elemento *“rental”* a la función *“addNewRental()”*, que lo añadirá como hijo al elemento root del *Document*.

Tras esto, se escribirá el *Document* mediante la función *“outputDocumentToFile(Document)”*, con lo que se finalizará el proceso de añadir un nuevo alquiler.

## **List**

Esta función simplemente muestra por pantalla el contenido del ficher carrental.xml. Para ello se usa la funcion readDocument, que, usa SAXBuilder para generar un Document a partir de un objeto File de Java. Una vez se ha creado este documento, se pasa a la función outputDocument, que lo imprime por pantalla. Para ello, igual que en el apartado de Reset, se usa XMLOutputter para escribir el documento por el canal de salida correspondiente. En este caso el terminal.



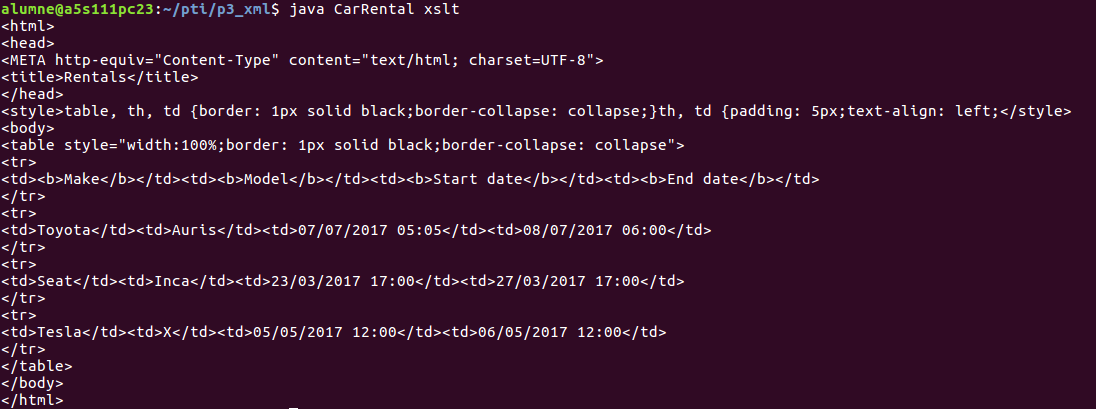


**Figura 5: Salida de la acción list**

## **XSLT**

Para crear el documento HTML a partir del documento XSLT se usa la clase TransformerFactory que permite, a partir de un fichero xslt y un document crearlo.

Para obtener el Document, se llama a a la función readDocument explicada anteriormente. Una vez leído este y transformado, se escribe por pantalla el resultado de esta transformación.

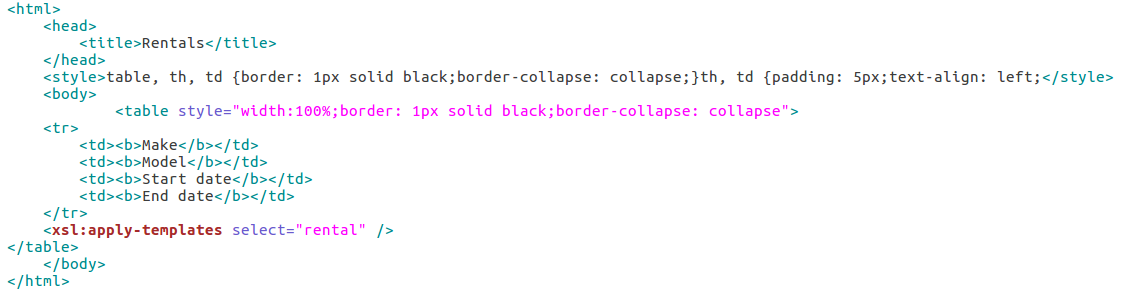
**Figura 6: Salida de la acción xslt**

Para ver la tabla resultante, podemos copiar la salida de la acción xlst y crear un nuevo documento HTML con ella. Ahora, se puede usar un navegador web para observar el resultado.

**Figura 7: Tabla resultante del comando xslt**

### **carrental.xslt**

Al usar el comando xslt es necesario usar una plantilla que sea capaz de entender el XML y traducirlo a HTML. Para ello, se ha definido esta plantilla dentro del fichero carrental.xslt. Su funcionamiento, una vez entendida la estructura de los alquileres es bastante sencilla.

Primeramente se busca el modelo carrental dentro del fichero XML. Para este, se define el estilo que va a tener la tabla (centrada y con las celdas con bordes) y sus columnas. Una vez se itera para cada modelo dentro de carrental (en este caso los rentals). Para cada uno de ellos se van a buscar todos sus atributos y hacer un match con el creador, el modelo, el dia de reserva y el de resolución. Para cada uno de ellos, se añade el atributo a la fila correspondiente.

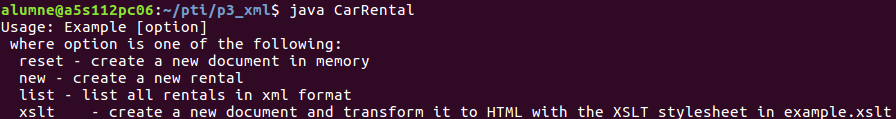
**Figura 8: Definición de la tabla (estilo y columnas) e iteración entre los rentals**

rentalIternation.png

**Figura 9: Creación de las filas para cada uno de los rentals**creacionColumna.png

**Figura 10: Iteración para cada uno de los atributos de los rentals y asignación de valores a cada una de las columnas**

## **Usage**

Si el usuario no introduce ninguna opción válida, se le va a mostrar por pantalla las opciones que están disponibles.

**Figura 11: Salida por pantalla del Usage**

## 3.2 Making it robust

En esta segunda versión, buscamos implementar un sistema más robusto capaz de soportar la caída de uno de los servidores.

# 4.Open questions

# 5.Personal opinion