# UA02. PRIMER ACCESO A DATOS

## 1.1. XML con DOM

La API DOM se caracteriza por ser un **analizador basado en modelos de carga de documentos con estructuras en árbol**, el cual guarda en memoria la información del XML.

Entre las características principales de este analizador podemos destacar que nos permitirá:

- tener los **datos en orden**,
- **navegar** por ellos en ambas direcciones,
- disponer de una **API de lectura y escritura** de datos,
- la **manipulación** del fichero XML.

Lo único negativo que cabe destacar es que el parser DOM tiene un **procesado de información bastante lento**, lo que provoca que consuma y ocupe mucho espacio en memoria del programa al cargar o tratar el fichero XML.

Si observamos el ejemplo, podemos ver que, para empezar, declaramos un fichero XML. Tendremos que indicarle la ruta del fichero. El fichero tendrá esta estructura con estos datos:

```xml
<?xml version= "1.0" encoding="UTF-8"?>
<coches>
    <coche>
        <marca>Seat</marca>
        <modelo>Ibiza</modelo>
        <color>rojo</color>
        <matriculacion>2019</matriculacion>
    </coche>
</coches>
```

A continuación, es necesario crear una **nueva instancia** del objeto DOM que cargará el archivo XML en la memoria.

```
try {
        //Indicaremos la ruta del fichero xml
        //Src es el nombre raiz de nuestro proyecto, main es la
        //Esta es la ruta relativa.
        File arxXml = new File("src/coches.xml");
        //Creamos los objetos que nos permitiran leer el ficherc
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newI
        DocumentBuilder db = dbf.newDocumentBuilder();
        //Le pasamos el XML
        Document doc = db.parse(arxXml);
        doc.getDocumentElement().normalize();
        System.out.println("Elemento raiz:" + doc.getDocumentEle
        NodeList nodeList = doc.getElementsByTagName("coche");
        //Creamos un bucle para leer los datos del xml y los mos
        for (int itr = 0; itr < nodeList.getLength(); itr++) {
            Node node = nodeList.item(itr);
            if (node.getNodeType() == Node.ELEMENT_NODE){
                Element eElement = (Element) node;
                System.out.println("Marca: "+ eElement.getElemer
                System.out.println("Modelo: "+ eElement.getEleme
                System.out.println("Color: "+ eElement.getElemer
                System.out.println("Matriculacion: "+ eElement.g
            }
        }
```

```
25      } catch (Exception e) {
26          e.printStackTrace();
27      }
```

En primer lugar, vamos a utilizar una nueva instancia de **DocumentBuilderFactory** (línea 7), esta clase nos va a permitir obtener un analizador sintáctico que produce la jerarquía de objetos DOM a partir de documentos XML.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

Una vez creada la instancia de esta clase, necesitaremos definir una clase **DocumentBuilder** (línea 8). Esta es necesaria para posteriormente definir el documento que nos permitirá parsear el XML.

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

Después crearemos una nueva instancia de la clase **Document**, que nos permitirá almacenar nuestro documento XML (línea 10). La clase Document representa un documento XML y nos proporcionará el acceso al contenido del documento XML.

```
Document doc = db.parse(arxXml);
```

Seguidamente, necesitaremos obtener el nodo raíz a través del método **getDocumentElement()**, así obtendremos los datos de esa etiqueta (línea 11).

```
System.out.println("Elemento raiz:" +
doc.getDocumentElement().getNodeName());
```

A continuación, tenemos que detectar cuántos elementos contiene el XML, es decir, cuántos nodos hay definidos. Para ello, utilizamos el método **getElementByTagName()** al cual le pasaremos el nombre del nodo que queremos sustraer, en este caso, "coche".

```
NodeList nodeList = doc.getElementsByTagName("coche");
```

Para almacenar los datos obtenidos, definiremos el **NodeList**, que almacenará todos los datos encontrados haciendo la llamada al método **getElementByTagName()** (línea 13).

Si necesitamos acceder a todos los nodos desde el inicio del fichero, podemos llamar recursivamente a este método: **getChildElement()**.

En el ejemplo vemos también que definimos la clase **Node**, esta nos servirá para asignar todos los datos de cada elemento "coche" encontrado en el XML (línea 16).

```
Node node = nodeList.item(itr);
```

Para obtener el valor del texto, usaremos el método **getElementByTagName()**.

```
System.out.println("Marca: "+
eElement.getElementsByTagName("marca").item(0).getTextContent());
```

## Ejemplo sin conocer estructura

```
1  for (int i = 0; i < nList.getLength(); i++) {           // Recor
2      Node node = nList.item(i);                          // Obtie
3      if (node.getNodeType() == Node.ELEMENT_NODE) {      // Si es
4          Element eElement = (Element) node;              // ... s
5          if(node.hasChildNodes()) {                      // Si el
6              NodeList nl = node.getChildNodes();         // ... s
7              for(int j=0; j<nl.getLength(); j++) {       // ... y
8                  Node nd = nl.item(j);                   // Se de
9                  System.out.print(nd.getTextContent());  // ... y
10             }
11         }
12     }
13 }
```