UA02. PRIMER ACCESO A DATOS

1.2. XML con SAX

SAX (**S**imple **A**PI for **X**ML) es un conjunto de clases e interfaces que ofrecen una herramienta muy útil para el procesamiento de documentos XML.

Permite analizarlos de forma secuencial (es decir, no carga todo el fichero en memoria como hace **DOM**), esto implica poco consumo de memoria aunque los documentos sean de gran tamaño.

En contraposición, impide tenes una visión global del documento que se va a analizar.

SAX es más complejo de programar que DOM, es un API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio *parser* de XML.

Para la implementación de SAX, primero necesitaremos tener un fichero XML. Usaremos este XML que contiene información sobre coches. Tendremos que tener en nuestro proyecto o en nuestro ordenador el fichero XML.

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

Para poder explicar mejor cómo estructurar una implementación de un ejemplo con SAX, vamos a desglosar el ejemplo por partes para poder explicarlo mejor.

En primer lugar, necesitamos definir la clase **SAXParserFactory** y crear una nueva instancia.

```
1 SAXParserFactory factory = SAXParserFactory.newInstance();
2 SAXParser saxParser = factory.newSAXParser();
```

Esta clase es la API encargada de proporcionar un SAX *parser*. Seguidamente, se deberá crear un SAX parser que se obtendrá gracias a la llamada del método **newSaxParser()**.

Como vemos en el siguiente ejemplo, para poder parsear el XML deberemos llamar al método **parse()**, al que se necesita pasarle por parámetro la ruta del fichero XML y un handler. Lo que hará es llamar a una clase auxiliar de apoyo que veremos cómo se crea a continuación.

```
1 SAXParserFactory factory = SAXParserFactory.newInstance();
2 SAXParser saxParser = factory.newSAXParser();
3 SaxHelper handler = new SaxHelper();
4 saxParser.parse("src/main/resources/xml/coches.xml", handler);
```

Un **handler** es una clase auxiliar que servirá para realizar los diferentes pasos de extracción de datos del XML. Deberemos crear una clase auxiliar llamada **SaxHelper** que extenderá de **DefaultHandler** y que nos proporcionará la implementación por defecto de los métodos necesarios para realizar el parser.

Para entender mejor en qué consiste, vamos a ver su implementación:

```
1
2 public class SaxHelper extends DefaultHandler{
3     ...
```

```
        }
```

La clase se llamará en este caso "SaxHelper", pero podemos asignarle el nombre que más nos convenga. Este método se llamará cuando se encuentre el principio de un elemento, este deberá extender siempre de la clase DefaultHandler. De este modo, nos podremos beneficiar de los métodos por defecto. Si no se implementa así, no nos funcionará el parser con SAX.

```
1 public class SaxHelper extends DefaultHandler{
2    boolean esMarca = false;
3    boolean esModelo = false;
4    boolean esColor = false;
5    boolean esMatriculacion = false;
6 }
```

A continuación, debemos crear tantas variables como atributos tiene nuestro XML. En nuestro caso, tenemos cuatro y hemos creado cuatro booleanos con sus nombres asociados. Estas variables nos servirán para saber si el elemento que estamos comprobando corresponde a cada atributo que queremos encontrar. Por tanto, tendremos uno para marca, modelo, color y matriculación.

Nuestra clase auxiliar, para funcionar correctamente, tendrá que implementar un método startElement, el cual se encargará de buscar los diferentes elementos y atributos de nuestro XML. Este método se llamará cuando se encuentre el principio de un elemento.

```
public void startElement(String uri, String localName, String element
   System.out.println("Inicio del elemento :" + elementos);
   switch (elementos) {
      case "marca":
         esMarca = true;
         break;
      case "modelo":
```

```
            esModelo = true;
            break;
        case "color":
            esColor = true;
            break;
        case "matriculacion":
            esMatriculacion = true;
            break;
        default:
            break;
    }
}
```

Si nos fijamos, el método recibe cuatro parámetros:

- URI: contendrá el namespace del XML si lo tiene. En este caso estará vacío.
- localName: puede ir vacío.
- Elementos: contendrá el nombre del elemento que se acaba de encontrar.
- Atributos: contendrá los atributos de cada elemento.

El método **startElement** se encarga de ir comprobando que el string "elementos" contenga uno de nuestros elementos del XML. Cuando encuentre uno de ellos, establecerá el valor **true** a la variable que corresponda según la etiqueta. A continuación, deberemos implementar el método **characters()**. Este método será llamado cuando se encuentra texto en el XML. Es un método que recibe por parámetro:

- Un array de tipo char que contendrá todos los caracteres de nuestro XML.
- Un int inicio que contendrá un int que indicará la posición en la que tiene que empezar a leer el array char anterior.
- Un int lenght que indicará el número de caracteres que tenemos que usar del array de caracteres.

```
public void characters(char ch[], int inicio, int length) throws SAXE
    if (esMarca) {
        System.out.println("Marca: " + new String(ch, inicio, length)
```

```
            esMarca = false;
            return;
        }
        if (esModelo) {
            System.out.println("Modelo: " + new String(ch, inicio,length)
            esModelo = false;
            return;
        }
        if (esColor) {
            System.out.println("Color: " + new String(ch, inicio, length)
            esColor = false;
            return;
        }
        if (esMatriculacion) {
            System.out.println("Matriculacion: " + new String(ch, inicio,
            esMatriculacion = false;
        return;
        }
    }
```

El método se encargará de comprobar qué elemento debe imprimir por consola, ya que en el método anterior hemos verificado qué elemento estábamos comprobando. Si la propiedad booleana correspondiente a ese elemento del XML es true, imprimirá por consola el elemento y el valor correspondiente, y volverá a setear a false. Esto se hace para indicar que ya no está en uso ese elemento.

Para finalizar con este helper, necesitaremos implementar el método endElement(). El método se llama cuando encuentra el final de un elemento. En este ejemplo, solo se imprimirá por consola el valor del elemento e indicará que hemos terminado con este bloque del XML.

```
    public void endElement(String uri, String localName, String elementos
        System.out.println("Fin del elemento: " + elementos);
    }
```

Este método recibe por parámetro un string con la URI, otro con localName y otro con el elemento XML finalizado. En este caso, solo tendrá valor el string "elementos" porque nuestro XML es más sencillo y estos campos están vacíos. Cabe decir que solo se rellenarán cuando el XML contenga esa información. Como vemos en el ejemplo, los objetos para acceder a los datos de un fichero son con SaxParserFactory y SaxParser.

Este es un ejemplo muy simplificado para observar el funcionamiento básico y los objetos. Es recomendable consultar GitLab para ver el ejemplo implementado, ya que la implementación de SAX es bastante complicada.