INFORME

PRÁCTICA 3

PTI

Eduard González Moreno
Miguel Guerrero López
06/04/2018

Entorno y configuración

En esta práctica no hemos necesitado ningún servidor ya que todo se ha hecho localmente. Solo hemos necesitado la terminal y hacer el install de java en caso de que no estuviera instalado previamente. Para la práctica se ha realizado una aplicación de terminal que nos permitía introducir los datos y ver los resultados por pantalla.

Resolución

Para la resolución de esta práctica se nos pedía que elaboramos una funcionalidad a la aplicación de terminal.
La primera era la operación reset, que no era más que crear un archivo xml con la siguiente estructura:

<?xml version="1.0" encoding="UTF-8"?>
<carrental>
</carrental>

Para solucionar este problema hemos hecho lo siguiente:
En el main de la aplicación java:

```
if(command.equals("reset")) outputDocumentToFile(createDocument());

En la función createDocument():
public static Document createDocument() {
        // Create the root element
        Element carElement = new Element("carrental");
        //create the document
        Document myDocument = new Document(carElement);
        return myDocument;
Que retorna un documento con la estructura que pedían anteriormente.
outputDocumentToFile():
public static void outputDocumentToFile(Document myDocument) {
        //setup this like outputDocument
        try {
            // XMLOutputter outputter = new XMLOutputter("  ", true);
            XMLOutputter outputter = new
XMLOutputter(Format.getPrettyFormat());

            //output to a file
            FileWriter writer = new FileWriter("carrental.xml");

            outputter.output(myDocument, writer);
            writer.close();
```

```
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
```

La segunda operación consistía en hacer un new rental y guardarlo en el archivo carrental.xml.

```
public static void nuevo(Document myDocument){
      Element root = myDocument.getRootElement();
      Element carRental = newElemento();
      root.addContent(carRental);
      outputDocumentToFile(myDocument);

public static Element newElemento(){
            Element carElement = new Element("rental");
            carElement.setAttribute(new Attribute("id",
"25"));//Pendiente de hacer un id correcto
            System.out.print("Marca del vehiculo:");
            String input = System.console().readLine();
            Element make = new Element("make");
            make.addContent(input);
            carElement.addContent(make);
            ////////////////////////////////////
            System.out.print("Modelo:");
            input = System.console().readLine();
            Element modelo = new Element("model");
            modelo.addContent(input);
            carElement.addContent(modelo);
            ////////////////////////////////////
            System.out.print("Numero de dias:");
            input = System.console().readLine();
            Element nofdays = new Element("nofdays");
            nofdays.addContent(input);
            carElement.addContent(nofdays);
            ////////////////////////////////////
            System.out.print("Numero de vehiculos:");
            input = System.console().readLine();
            Element nofunits = new Element("nofunits");
            nofunits.addContent(input);
            carElement.addContent(nofunits);
            ////////////////////////////////////
            System.out.print("Descuentos:");
            input = System.console().readLine();
            Element des = new Element("discount");
            des.addContent(input);
            carElement.addContent(des);
```

```
            return carElement;
```

En esta solución creamos un nuevo elemento, que será el nuevo rental. Introducimos los parámetros por terminal, el programa los recoge y los añade al fichero carrental.xml
Ejemplo del nuevo xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<car vin="123fhg5869705iop90">
  <!--Description of a car-->
  <make>Toyota</make>
  <model>Celica</model>
  <year>1997</year>
  <color>green</color>
  <license state="CA">1ABC234</license>
</car>
```

La tercera función era el List, que no es más que leer el fichero donde están los carrental y mostrarlos por pantalla. La función que hemos hecho es la siguiente:

```
public static Document readDocument() {
        try {
            SAXBuilder builder = new SAXBuilder();
            Document anotherDocument = builder.build(new
File("carrental.xml"));
            return anotherDocument;
        } catch(JDOMException e) {
            e.printStackTrace();
        } catch(NullPointerException e) {
            e.printStackTrace();
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
        return null;
    }
```

Después de la función del read hacemos el output a la terminal del documento que hemos leído.

La siguiente funcionalidad de la aplicación consistía era transformar en html el carrental.xml utilizando una plantilla xslt.

```
public static void executeXSLT(Document myDocument) {
            try {
                  TransformerFactory tFactory =
TransformerFactory.newInstance();
            // Make the input sources for the XML and XSLT documents
            org.jdom2.output.DOMOutputter outputter = new
org.jdom2.output.DOMOutputter();
```

```
            org.w3c.dom.Document domDocument =
outputter.output(myDocument);
            javax.xml.transform.Source xmlSource = new
javax.xml.transform.dom.DOMSource(domDocument);
            StreamSource xsltSource = new StreamSource(new
FileInputStream("carrental.xslt"));
                  //Make the output result for the finished document
            StreamResult xmlResult = new StreamResult(System.out);
                  //Get a XSLT transformer
                  Transformer transformer =
tFactory.newTransformer(xsltSource);
                  //do the transform
                  transformer.transform(xmlSource, xmlResult);
         } catch(FileNotFoundException e) {
             e.printStackTrace();
         } catch(TransformerConfigurationException e) {
             e.printStackTrace();
            } catch(TransformerException e) {
             e.printStackTrace();
         } catch(org.jdom2.JDOMException e) {
             e.printStackTrace();
         }
      }
```

Este es el código de la función, a continuación mostraremos un ejemplo del html:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>RENTALS</title>
</head>
<body>
<table border="0">
        MAKE=Mazda<br>
        Modelo=1<br>
        Numero dias=2<br>
        Numero vehiculos=3<br>
        Descuento=5<br>
</table>
<br>
<hr>
<br>
<table border="0">
        MAKE=Toyota<br>
        Modelo=Yaris<br>
        Numero dias=5<br>
        Numero vehiculos=50<br>
        Descuento=100<br>
```

```
</table>
<br>
<hr>
<br>
</body>
</html>
```

La última función trata sobre validar el xml con un XSD schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="carrental">
    <xs:complexType>
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="rental">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="make" type="xs:string" />
                        <xs:element name="model" type="xs:string" />
                        <xs:element name="nofdays"
type="xs:unsignedByte" />
                        <xs:element name="nofunits"
type="xs:unsignedByte" />
                        <xs:element name="discount"
type="xs:unsignedByte" />
                    </xs:sequence>
                    <xs:attribute name="id" type="xs:unsignedInt"
use="required" />
                </xs:complexType>
            </xs:element>
        </xs:sequence>
     </xs:complexType>
    </xs:element>
</xs:schema>
```

La función que utiliza este esquema es la siguiente:

```
public static void validate(){
           try {
           SAXBuilder builder = new
SAXBuilder(XMLReaders.XSDVALIDATING);
           Document anotherDocument = builder.build(new
File("carrental.xml"));
           System.out.println("Root: " +
anotherDocument.getRootElement().getName());
           } catch(JDOMException e) {
            e.printStackTrace();
```

```
        } catch(NullPointerException e) {
            e.printStackTrace();
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }


    }
```

Evaluación de la solución

Para evaluar la solución utilizamos la terminal y el navegador para ver el archivo html que creamos con la aplicación de terminal. También podemos ver el contenido de los ficheros xml para comprobar que se han generado correctamente. Adjuntamos algunos ejemplos que sacamos al hacer la práctica.

Ventajas tecnológicas

La idea de esta práctica era un poco ver las diferencias entre el xml y Json. La principal diferencia quizás sea en que XML es más complicado de entender y tiene un formato más estricto que JSON. Pero como ventaja XML tiene los schemas que facilitan mucho el trabajo y se pueden crear estructura complejas y luego reutilizarlas.