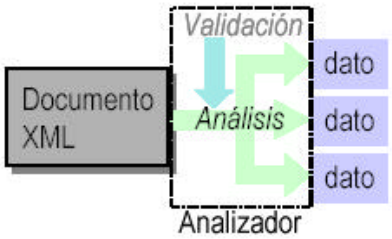
```
    } catch (IOException ex) {
      //puede provocarla '.parse()'
      System.out.println("Error de acceso al fichero origen");
    }

  }
}//fin PruebaSaxl

//clase de la que extendemos para gestionar el analizador XML
class GestionContenido extends DefaultHandler {
  public GestionContenido() {
    super();
  }
//métodos que sobreescribimos para nuestro tratamiento
   @Override
  public void startDocument() {
    System.out.println("Comienzo del Documento XML");
  }
   @Override
  public void endDocument() {
    System.out.println("Final del Documento XML");
  }
  @Override
  public void startElement(String uri, String nombre,
          String nombreC, Attributes atts) {

    System.out.println("\tPrincipio Elemento: " + nombre);
  }
  @Override
  public void endElement(String uri, String nombre, String nombreC) {
    System.out.println("\tFin Elemento: " + nombre);
  }
   @Override
  public void characters(char[] ch, int inicio, int longitud)
          throws SAXException {

    String car = new String(ch, inicio, longitud);
    car = car.replaceAll("[\t\n]", "");//quitar saltos de linea y tabulador
    System.out.println("\tCaracteres: " + car);
  }
}//fin GestionContenido
```

7.3 ANALIZADORES XML Y VALIDACIÓN

Los analizadores permiten seleccionar diferentes opciones, entre ellas si ha de comprobarse o no la validez de un documento xml respecto a su definición (el DTD). Esta validación añade una comprobación y por tanto una garantía más, pero también consume más recursos y ralentiza el procesamiento del documento xml.



¿Cómo validar XML con Java?

De entre las diferentes APIs para Java que proporcionan mecanismos para validar documentos XML veremos ejemplos con DOM y con SAX.

1. Para validar es necesario proporcionar un **ErrorHandler (manejador de errores)** para que nuestra aplicación pueda recibir los errores de validación. El que utilizaremos aquí es muy simple: muestra los errores y las advertencias en la Salida del Sistema, y continúa hasta que el documento XML ha sido completamente analizado, o hasta que se produce un error fatal.

Para construir el manejador hay que:

- implementar la interfaz ***ErrorHandler*** de SAX, y
- sobreescribir los métodos que se indican a continuación

```
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXParseException;

/**
 * clase para recibir los errores de validación de un documento XML
 */
public class SimpleErrorHandler implements ErrorHandler {

   @Override
   /* notificación de una advertencia */
   public void warning(SAXParseException e) {
      System.out.println(e.getMessage());
   }

   @Override
   /* notificación de un error recuperable */
   public void error(SAXParseException e) {
      System.out.println(e.getMessage());
   }

   @Override
   /* notificación de un error no recuperable */
   public void fatalError(SAXParseException e) {
      System.out.println(e.getMessage());
   }
}
```

2. **¿Qué se puede validar?**
   - Lo más sencillo, si es un documento bien formado
   - Si es un documento válido (DTD o esquema XSD)

Comprobación de documento bien formado

```
<!DOCTYPE contacts SYSTEM "contacts.dtd">
<contacts xsi:noNamespaceSchemaLocation="contacts.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <contact title="citizen">
      <firstname>Edwin</firstname>
      <lastname>Dankert</lastname>
   </contact>
</contacts>
```

Un documento se dice que está bien formado cuando cumple con las reglas **Wellformed XML**. Con la ayuda de nuestro manejador de errores, podemos comprobarlo fácilmente tanto con DOM como con SAX mediante

```
.setErrorHandler(new SimpleErrorHandler());
```

Aunque en el documento XML se han especificado tanto un doctype "contacts.dtd", como un esquema “contacts.xsd”, ahora no queremos comprobar esto, sino sólo que está bien formado. Por eso ponemos

```
.setValidating(false);
```

DOM [Wellformed.java **]**

```
public static void main(String[] args) {

  String rutaXML = "...\\ValidateExample\\contacts.xml";

  try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating(false);
    DocumentBuilder builder = factory.newDocumentBuilder();

      builder.setErrorHandler(new SimpleErrorHandler());

     builder.parse(new InputSource(rutaXML));

  } catch (ParserConfigurationException e) {
    System.out.println(e);
  } catch (SAXException e) {
    System.out.println(e);
  } catch (IOException e) {
    System.out.println(e);
  }
}
```

SAX [Wellformed.java**]**

```
  public static void main(String[] args) {

  String rutaXML = "...\\ValidateExample\\contacts.xml";

  try {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(false);

    SAXParser parser = factory.newSAXParser();
    XMLReader reader = parser.getXMLReader();

    reader.setErrorHandler(new SimpleErrorHandler());
    reader.parse(new InputSource(rutaXML));

  } catch (ParserConfigurationException e) {
    System.out.println(e);
  } catch (SAXException e) {
    System.out.println(e);
  } catch (IOException e) {
    System.out.println(e);
  }
}
```

Comprobación de documento válido (DTD)

```
<!ELEMENT contacts (contact*)>
<!ATTLIST contacts xsi:noNamespaceSchemaLocation CDATA #IMPLIED>
<!ATTLIST contacts xmlns:xsi CDATA #IMPLIED>

<!ELEMENT contact (firstname,lastname)>
<!ATTLIST contact title (MR|MS|MRS) "MS">

<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

Intencionadamente, el atributo "**title**" en el documento "contacts.xml" tiene un valor que no está permitido de acuerdo con este DTD. De manera que cuando se utiliza esta DTD para validar el documento, se obtiene el error:

```
El atributo "title" con el valor "citizen" debe tener un valor de la lista "MR
MS MRS ".
```

Activamos la validación con dtd:

```
.setValidating(true);
```

DOM [ValidateInternalDTD.java]

```
    public static void main(String[] args) {

    String rutaXML = "...\\ValidateExample\\contacts.xml";

    try {
       DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
       factory.setValidating(true);

       DocumentBuilder builder = factory.newDocumentBuilder();

       builder.setErrorHandler(new SimpleErrorHandler());
       builder.parse(new InputSource(rutaXML));

    } catch (ParserConfigurationException e) {
       System.out.println(e);
    } catch (SAXException e) {
       System.out.println(e);
    } catch (IOException e) {
       System.out.println(e);
    }
  }
```

SAX [ValidateInternalDTD.java]

```
  public static void main(String[] args) {

    String rutaXML = "...\\ValidateExample\\contacts.xml";
    try {
       SAXParserFactory factory = SAXParserFactory.newInstance();
       factory.setValidating(true);
       SAXParser parser = factory.newSAXParser();
       XMLReader reader = parser.getXMLReader();
       reader.setErrorHandler(new SimpleErrorHandler());
       reader.parse(new InputSource(rutaXML));
    } catch (ParserConfigurationException e) {
       System.out.println(e);
    } catch (SAXException e) {
       System.out.println(e);
    } catch (IOException e) {
       System.out.println(e);
    }
  }
```

Comprobación de documento válido (XSD)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="contacts">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="contact"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="contact">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:NCName"/>
        <xs:element name="lastname" type="xs:NCName"/>
       </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Observa que intencionadamente, el documento `contacts.xml` contiene un atributo adicional que no se ha definido en el esquema XML. De manera que cuando se utiliza este esquema para validar el documento, se obtiene el error: `cvc-complex-type.3.2.2: No está permitido que el atributo 'title' aparezca en el elemento 'contact'`

De nuevo:

```
factory.setValidating(false);
```

DOM [ValidateExternalSchema.java]

```
public static void main(String[] args) {

   String rutaXML = "...\\ValidateExample\\contacts.xml";
   String rutaXSD = "...\\ValidateExample\\contacts.xsd";
   try {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      factory.setValidating(false);

      SchemaFactory schemaFactory = SchemaFactory.newInstance(
               "http://www.w3.org/2001/XMLSchema");

      factory.setSchema(schemaFactory.newSchema(
               new Source[]{new StreamSource(rutaXSD)}));

      DocumentBuilder builder = factory.newDocumentBuilder();
      builder.setErrorHandler(new SimpleErrorHandler());
      builder.parse(new InputSource(rutaXML));
   } catch (ParserConfigurationException e) {
      System.out.println(e);
   } catch (SAXException e) {
      System.out.println(e);
   } catch (IOException e) {
      System.out.println(e);
   }
}
```

SAX [ValidateExternalSchema.java]

```
   String rutaXML = "...\\ValidateExample\\contacts.xml";
   String rutaXSD = "...\\ValidateExample\\contacts.xsd";

   try {
      SAXParserFactory factory = SAXParserFactory.newInstance();
      factory.setValidating(false);
      factory.setNamespaceAware(true);
      SchemaFactory schemaFactory = SchemaFactory.newInstance(
               "http://www.w3.org/2001/XMLSchema");
      factory.setSchema(schemaFactory.newSchema(
               new Source[]{new StreamSource(rutaXSD)}));
      SAXParser parser = factory.newSAXParser();
      XMLReader reader = parser.getXMLReader();
      reader.setErrorHandler(new SimpleErrorHandler());
      reader.parse(new InputSource(rutaXML));
   } catch (ParserConfigurationException e) {
      System.out.println(e);
   } catch (SAXException e) {
      System.out.println(e);
   } catch (IOException e) {
      System.out.println(e);
   }
}
```

7.4 ANALIZADORES XML Y VINCULACIÓN (BINDING)

JAXB (Java Architecture for XML Binding) simplifica el acceso a documentos XML representando la información

obtenida de los documentos XML en un programa en formato Java, o sea, proporciona a los desarrolladores de aplicaciones Java, una forma rápida para vincular esquemas XML a representaciones Java.

JAXB proporciona métodos para, a partir de documentos XML, obtener árboles de contenido (generados en código Java), para después operar con ellos o manipular los mismos en una aplicación Java y generar documentos XML con la estructura de los iniciales, pero ya modificados.

Algunos conceptos a tener en cuenta:

- **Parsear** un documento XML consiste en "escanear" el documento y dividirlo o separarlo lógicamente en piezas discretas. El contenido parseado está entonces disponible para la aplicación.
- **Binding**: Binding o vincular un esquema (schema) significa generar un conjunto de clases Java que representan el esquema.
- **Compilador de esquema** o schema compiler: liga un esquema fuente a un conjunto de elementos de programa derivados. La vinculación se describe mediante un lenguaje de vinculación basado en XML.
- **Binding runtime** framework: proporciona operaciones de unmarshalling y marshalling para acceder, manipular y validar contenido XML usando un esquema derivado o elementos de programa.
- **Marshalling**: es un proceso de codificación de un objeto en un medio de almacenamiento, normalmente un fichero. Proporciona a una aplicación cliente la capacidad para convertir un árbol de objetos Java JAXB a ficheros XML. Por defecto, el marshaller usa codificación UTF-8 cuando genera los datos XML.
- **Unmarshalling**: proporciona a una aplicación cliente la capacidad de convertir datos XML a objetos Java JAXB derivados.

JAXB permite mapear clases Java a representaciones en XML y viceversa.

JAXB proporciona dos principales características:

- La capacidad de serializar (marshalling) objetos Java a XML.
- Lo inverso, es decir, deserializar (unmarshalling) XML a objetos Java.

O sea que **JAXB permite almacenar y recuperar datos en memoria en cualquier formato XML, sin la necesidad de implementar un conjunto específico de rutinas XML** de carga y salvaguarda para la estructura de clases del programa.

El compilador de JAXB (schema compiler) permite generar una serie de clases Java que podrán ser llamadas desde nuestras aplicaciones a través de métodos sets y gets para obtener o establecer los datos de un documento XML.

El funcionamiento esquemático al usar JAXB sería:

- Crear un esquema (fichero .xsd) que contendrá las estructura de las clases que deseamos utilizar.
- Compilar con el JAXB compiler (bien con un IDE como NetBeans o desde línea de comandos con el comando xjc) ese fichero .xsd, de modo que nos producirá los POJOs (Acrónimo de Plain Old Java Object. Se usa por los programadores Java para enfatizar el uso de clases sencillas, que no dependen de un framework en especial), o sea, una clase por cada uno de los tipos que hayamos especificado en el fichero .xsd. Esto nos producirá los ficheros .java
- Compilar esas clases java.
- Crear un documento XML: validado por su correspondiente esquema XML, o sea el fichero .xsd, se crea un árbol de objetos.
- Ahora se puede parsear el documento XML, accediendo a los métodos *gets y sets* del árbol de objetos generados por el proceso anterior. Así se podrá modificar o añadir datos.
- Después de realizar los cambios que se estimen, se realiza un proceso para sobrescribir el documento XML o crear un nuevo documento XML

Construir una aplicación JAXB

Para construir una aplicación JAXB necesitamos tener un esquema XML.

Tras obtener el esquema XML, seguimos los **siguientes pasos para construir la aplicación JAXB**:

1. **Escribir el esquema**: es un documento XML que contiene la estructura que se tomará como indicaciones para construir las clases. Estas indicaciones pueden ser, por ejemplo, el tipo primitivo al que se debe unir un valor de atributo en la clase generada.
2. **Generar los ficheros fuente de Java**: para esto usamos el compilador de esquema, ya que éste toma el esquema como entrada de información. Cuando se haya compilado el código fuente, podremos escribir una aplicación basada en las clases que resulten.
3. **Construir el árbol de objetos Java**: con nuestra aplicación, se genera el árbol de objetos java, también llamado árbol de contenido, que representa los datos XML que son validados con el esquema. Hay dos formas de hacer esto:
    a. Instanciando las clases generadas.
    b. Invocando al método `unmarshall` de una clase generada y pasarlo en el documento. El método `unmarshall` toma un documento XML válido y construye una representación de árbol de objetos.
4. **Acceder al árbol de contenido** usando nuestra aplicación: ahora podemos acceder al árbol de contenido y modificar sus datos.
5. **Generar un documento XML** desde el árbol de contenido. Para poder hacerlo tenemos que invocar al método `marshall` sobre el objeto raíz del árbol.

Aunque estos pasos que acabamos de comentarte te parezcan algo complicados, vamos a ver un ejemplo sencillo realizado en NetBeans, en el que clarificaremos todo esto, comprobando que no es tan difícil como parece.

Ejemplo de uso de JAXB en NetBeans

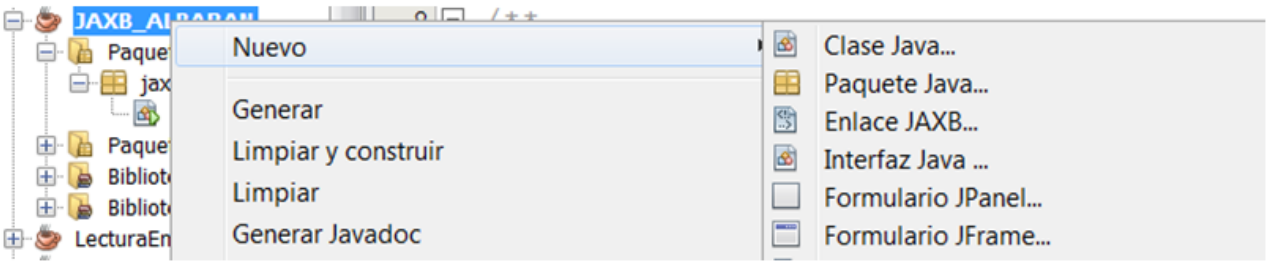
Vamos a suponer una estructura de un archivo que podría usarse en un almacén distribuidor de medicamentos. El fichero albaran.xsd tiene la estructura que el almacén usa cuando envía un pedido de medicamentos que le ha hecho una farmacia.

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
 <xsd:element name="pedido" type="PedidoType"/>
 <xsd:element name="comentario" type="xsd:string"/>

 <xsd:complexType name="PedidoType">
  <xsd:sequence>
   <xsd:element name="facturarA" type="Direccion"/>
   <xsd:element ref="comentario" minOccurs="0"/>
   <xsd:element name="articulos" type="Articulos"/>
  </xsd:sequence>
  <xsd:attribute name="fechaPedido" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="Direccion">
  <xsd:sequence>
   <xsd:element name="nombre" type="xsd:string"/>
   <xsd:element name="calle" type="xsd:string"/>
   <xsd:element name="ciudad" type="xsd:string"/>
   <xsd:element name="provincia" type="xsd:string"/>
   <xsd:element name="codigoPostal" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="pais" type="xsd:NMTOKEN" fixed="Espana"/>
 </xsd:complexType>


 <xsd:complexType name="Articulos">
  <xsd:sequence>
   <xsd:element name="articulo" minOccurs="1" maxOccurs="unbounded">
```

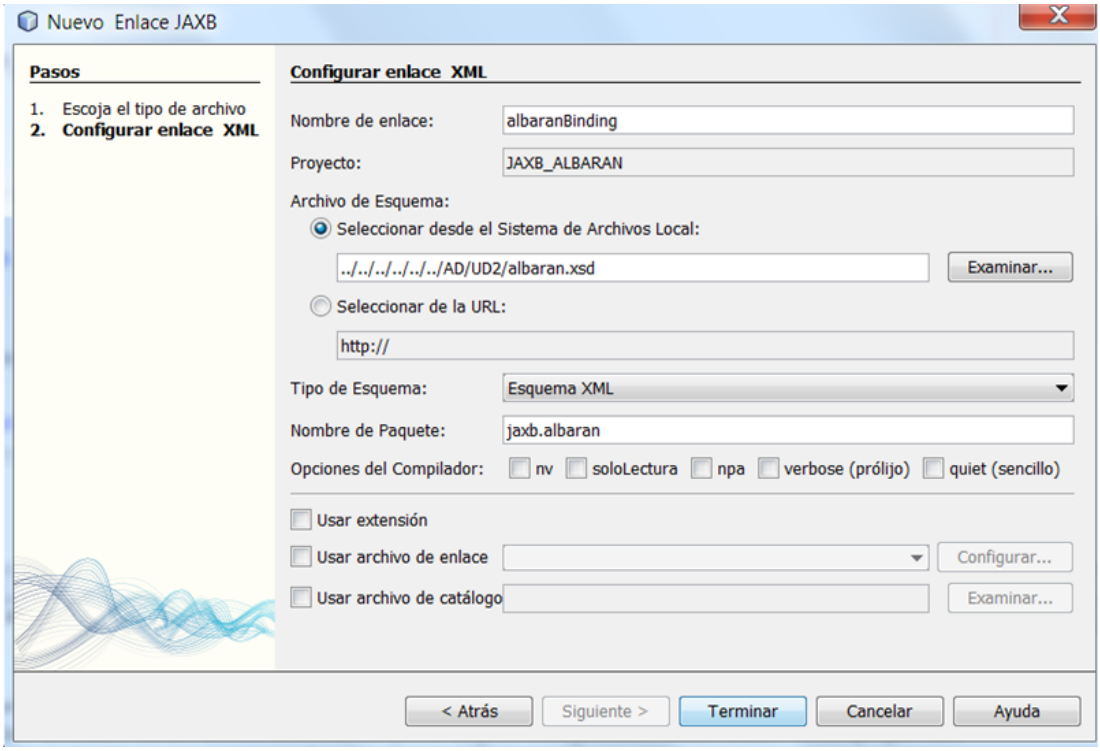
```
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="nombreProducto" type="xsd:string"/>
      <xsd:element name="cantidad">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="precio" type="xsd:decimal"/>
      <xsd:element ref="comentario" minOccurs="0"/>
      <xsd:element name="fechaEnvio" type="xsd:date" minOccurs="0"/>
     </xsd:sequence>
     <xsd:attribute name="codigo" type="PER" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:simpleType name="PER">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

El tipo denominado PedidoType contiene los elementos: facturarA, comentario, artículos y fecha del pedido.
El tipo de los artículos contiene: codigo, nombreProducto, precio, cantidad, fechaEnvio, comentario

- Una vez creado el fichero albaran.xsd (lo puedes descargar de la plataforma educativa), crea un proyecto Java Aplication, por ejemplo de nombre JAXB_albaran
- Sobre el proyecto, click derecho y selecciona Nuevo>> Enlace JAXB (o bien Otros>>XML >> Enlace JAXB

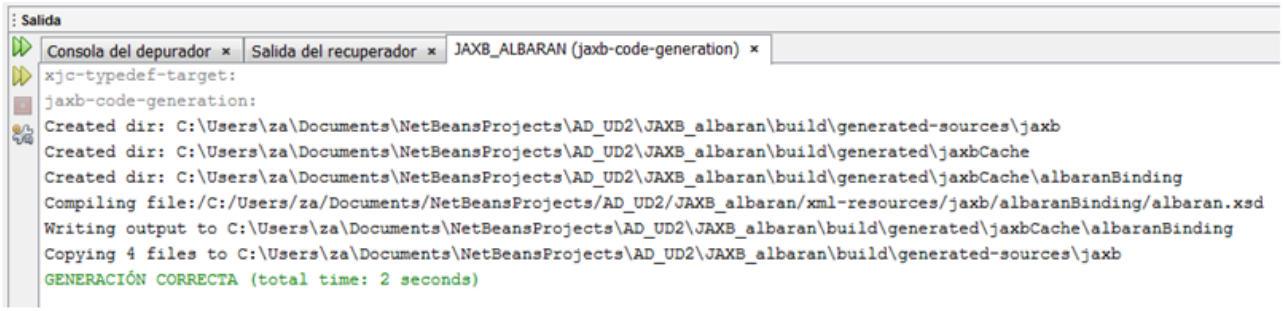


- **Rellena los siguientes datos:**
  - Nombre del enlace
  - Selecciona el fichero esquema .xsd creado enteriormente
  - Indica el tipo de esquema y nombre del paquete

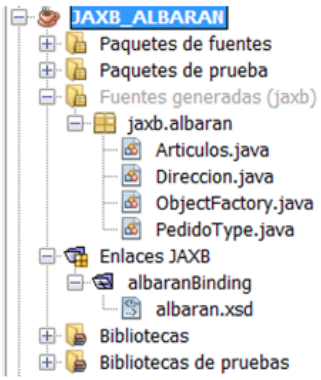


- Pulsa terminar

- Como hemos pulsado en Terminar, automáticamente se produce la compilación y se generan las clases a partir del fichero esquema, (el .xsd)



- En la ventana de proyectos vemos que se han creado las clases que representan a los tipos complejos del XML Schema **albaran.xsd**



- Creamos un fichero **albaran.xml** con los datos de un pedido para una farmacia. (por ejemplo con Notepad++)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <pedido xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="albaran.xsd">
  - <facturarA>
      <nombre>Farmacia Gil</nombre>
      <calle>Mayor 102</calle>
      <ciudad>Madrid</ciudad>
      <provincia>Madrid</provincia>
      <codigoPostal>20080</codigoPostal>
    </facturarA>
    <comentario>Lineas de pedido</comentario>
  - <articulos>
    - <articulo codigo="872-AA">
        <nombreProducto>Aspirina</nombreProducto>
        <cantidad>1</cantidad>
        <precio>3.60</precio>
        <comentario>Para el dolor de cabeza</comentario>
        <fechaEnvio>2010-10-18</fechaEnvio>
      </articulo>
    - <articulo codigo="926-BC">
        <nombreProducto>Gelocatil</nombreProducto>
        <cantidad>2</cantidad>
        <precio>7.20</precio>
        <fechaEnvio>2010-09-15</fechaEnvio>
      </articulo>
    </articulos>
</pedido>
```

- Creamos una clase (ModificaAlbaPed.java) para modificar el fichero albaran.xml

```
public class ModificaAlbaPed {
public static void main(String[] args) {
  try {
    // Crear una instancia para poder manipulas las clases generadas, que están en el paqute jaxb.albaran
    JAXBContext jaxbContext = JAXBContext.newInstance("jaxb.albaran");

    // Crear un objeto de tipo Unmarshaller para convertir datos XML en un árbol de objetos Java
    Unmarshaller u = jaxbContext.createUnmarshaller();

    // La clase JAXBElement representa a un elemento de un documento XML en este caso a un elemento del
documento albaran.xml
    JAXBElement jaxbElement = (JAXBElement) u.unmarshal(new FileInputStream("C:\\albaran.xml"));

    // El método getValue() retorna el modelo de contenido (content model) y el valor de los atributos del
elemento
    PedidoType pedidoType = (PedidoType) jaxbElement.getValue();

    // Obtenemos una instancia de tipo PedidoType para obtener un Objeto de tipo Direccion
     Direccion direccion = pedidoType.getFacturarA();

    // Establecemos los datos
    direccion.setNombre("Jose Javier");
    direccion.setCalle("Zafiro 3");
    direccion.setCiudad("Molina");
    direccion.setProvincia("Murcia");
    direccion.setCodigoPostal(new BigDecimal("30500"));

    // Crear un objeto de tipo Marshaller para posteriormente convertir el árbol de objetos Java a datos XML
    Marshaller m = jaxbContext.createMarshaller();

    // Crear el resultado XML no formateado para lectura de las personas, con saltos de línea, etc
     m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
```

```
            // Escribir el elmento obtenido como primer parámetro por la salida estándar, el segundo parámetro
            m.marshal(jaxbElement, System.out);
        } catch (JAXBException je) {
            System.out.println(je.getCause());
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        }
    }
}
```

- Tras compilar y ejecutar la clase, se nos mostrará por la salida estándar el resultado, el contenido del fichero albaran.xml