1.9.2 Paso de esquemas XML (.xsd) a clases java.

El compilador de JAXB nos va a permitir generar una serie de clases Java a partir de un esquema xml. Un esquema XML describe la estructura de un documento XML. A los esquemas xml se le llama XSD (XML Schema Definition).

JAXB, compila el fichero xsd, creando una serie de clases para cada uno de los tipos que se haya especificado en el xsd. Esas clases serán clase POJO, y las podremos utilizar para crear, y modificar documentos XML utilizando JAVA.

Ejemplo3: Validar XML con XSD. Para validar un XML con un XSD, desde esta URL:

http://www.utilities-online.info/xsdvalidation/
https://www.freeformatter.com/xml-validator-xsd.html

<table>
<tr><th>XSD</th><th>XML</th></tr>
<tr><td>

DOCUMENTO CON UN EMPLEADO

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

 <xsd:element name="empleados" type="EmpleadosType" />


<xsd:complexType name="EmpleadosType">
  <xsd:sequence>
     <xsd:element name="codigo" type="xsd:integer"/>
     <xsd:element name="nombre" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

</td><td>

DOCUMENTO CON UN EMPLEADO

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

<empleados>
       <codigo>1</codigo>
       <nombre>weewrrewrew </nombre>
</empleados>
```

</td></tr>
<tr><td>

DOCUMENTO CON VARIOS EMPLEADOS

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

<xsd:element name="empleados" type="Clasedatosempleados" />

<xsd:complexType name="Clasedatosempleados">
  <xsd:sequence>
     <xsd:element name="emple" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
           <xsd:sequence>
                <xsd:element name="empno" type="xsd:integer" />
                <xsd:element name="apellido" type="xsd:string" />
                <xsd:element name="oficio" type="xsd:string" />
                <xsd:element name="salario" type="xsd:float" />
           </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

O también este XSD lo validaría.

</td><td>

DOCUMENTO CON VARIOS EMPLEADOS

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

<empleados>
     <emple>
       <empno>1</empno>
       <apellido>Gil</apellido>
       <oficio>Profesor</oficio>
       <salario>1000 </salario>
      </emple>
      <emple>
       <empno>2</empno>
       <apellido>Ramos</apellido>
       <oficio>Profesora </oficio>
       <salario>2000</salario>
      </emple>
      <emple>
       <empno>3</empno>
       <apellido>García</apellido>
       <oficio>Vendedor</oficio>
       <salario>2100</salario>
      </emple>

</empleados>
```

</td></tr>
</table>

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

<xsd:element name="empleados" type="DatosempleadosType" />

<xsd:complexType name="DatosempleadosType">
  <xsd:sequence>
     <xsd:element name="emple" type="Empleado"
                  minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Empleado">
  <xsd:sequence>
        <xsd:element name="empno" type="xsd:integer" />
        <xsd:element name="apellido" type="xsd:string" />
        <xsd:element name="oficio" type="xsd:string" />
        <xsd:element name="salario" type="xsd:float" />
   </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

DOCUMENTO CON UN DEPARTAMENTO Y VARIOS EMPLEADOS

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

<xsd:element name="datosempledepart" type="DatosempledepartType" />

<xsd:complexType name="DatosempledepartType">
  <xsd:sequence>
     <xsd:element name="departamento" type="Departamento" />
     <xsd:element name="empleados" type="Empleados" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Departamento">
  <xsd:sequence>
        <xsd:element name="codigodep" type="xsd:integer" />
        <xsd:element name="nombredep" type="xsd:string" />
        <xsd:element name="localidad" type="xsd:string" />
        <xsd:element name="presupuesto" type="xsd:decimal" />
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Empleados">
   <xsd:sequence>
     <xsd:element name="emple" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="empno" type="xsd:integer" />
            <xsd:element name="apellido" type="xsd:string" />
            <xsd:element name="oficio" type="xsd:string" />
            <xsd:element name="salario" type="xsd:float" />
          </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

DOCUMENTO CON UN DEPARTAMENTO Y VARIOS EMPLEADOS

```
<?xml version="1.0" encoding="UTF-8"?>
<datosempledepart>
  <departamento>
    <codigodep>1234</codigodep>
    <nombredep>Ventas</nombredep>
    <localidad>Toledo</localidad>
    <presupuesto>123.45</presupuesto>
  </departamento>
  <empleados>
    <emple>
      <empno>12345678</empno>
      <apellido>Rajoy</apellido>
      <oficio>Comercial</oficio>
      <salario>3.14159</salario>
    </emple>
    <emple>
      <empno>23456789</empno>
      <apellido>Iglesias</apellido>
      <oficio>Secretario</oficio>
      <salario>10</salario>
    </emple>
    <emple>
      <empno>01234567</empno>
      <apellido>Rivera</apellido>
      <oficio>Administrativo</oficio>
      <salario>20</salario>
    </emple>
  </empleados>
</datosempledepart>
```

Otra forma de presentar el documento de 1 departamento y varios empleados es esta:

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

    <xsd:element name="datosempledepart" type="DatosempledepartType" />

    <xsd:complexType name="DatosempledepartType">
           <xsd:sequence>
                  <xsd:element name="departamento" type="Departamento" />
                  <xsd:element name="empleados" type="ListaEmpleados" />
           </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Departamento">
           <xsd:sequence>
                  <xsd:element name="codigodep" type="xsd:integer" />
                  <xsd:element name="nombredep" type="xsd:string" />
                  <xsd:element name="localidad" type="xsd:string" />
                  <xsd:element name="presupuesto" type="xsd:decimal" />
           </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ListaEmpleados">
           <xsd:sequence>
                  <xsd:element name="emple" type="Empleado" minOccurs="1" maxOccurs="unbounded" />
           </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Empleado">
           <xsd:sequence>
                  <xsd:element name="empno" type="xsd:integer" />
                  <xsd:element name="apellido" type="xsd:string" />
                  <xsd:element name="oficio" type="xsd:string" />
                  <xsd:element name="salario" type="xsd:float" />
           </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Ejemplo4:

Partimos de un XSD, en el que vamos a representar a un artículo y sus ventas. El artículo puede tener varias ventas, mínimo 1 venta. Datos del artículo ***(DatosArtic)*** son: *codigo*, *denominacion*, *stock* y *precio.* Datos de cada venta ***(ventas)*** son: *numventa*, *unidades*, *nombrecliente* y *fecha*.

El fichero XSD se llama ***ventas_articulo.xsd,*** y el contenido es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

 <xsd:element name="ventasarticulos" type="VentasType"/>

 <xsd:complexType name="VentasType">
  <xsd:sequence>
   <xsd:element name="articulo" type="DatosArtic"/>
   <xsd:element name="ventas" type="Ventas"/>
  </xsd:sequence>
```

```
 </xsd:complexType>

 <xsd:complexType name="DatosArtic">
  <xsd:sequence>
   <xsd:element name="codigo" type="xsd:string"/>
   <xsd:element name="denominacion" type="xsd:string"/>
   <xsd:element name="stock" type="xsd:integer"/>
   <xsd:element name="precio" type="xsd:decimal"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Ventas">
  <xsd:sequence>
   <xsd:element name="venta" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="numventa" type="xsd:integer"/>
        <xsd:element name="unidades">
        <xsd:simpleType>
          <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxExclusive value="100"/>
          </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="nombrecliente" type="xsd:string"/>
      <xsd:element name="fecha" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

¡INTERESANTE!
Para saber más sobre XSD puedes consultar estas URLs:
http://www.w3schools.com/xml/schema_howto.asp
http://www.w3schools.com/xml/schema_schema.asp

Este XSD está formado por los siguientes elementos:

- El ***elemento principal*** (raíz del documento) se va a llamar ***ventasarticulos*** y su tipo de dato lo vamos a llamar ***VentasType***:

```
<xsd:element name="ventasarticulos" type="VentasType"/>
```

- El tipo ***VentasType***, type=*"VentasType"* formado por las ventas de un artículo, se llama *ventasarticulos*. Es el tipo del elemento principal `<xsd:element name="ventasarticulos" type="VentasType"/>`.

Es un tipo complejo, formado por dos elementos, esto van a ser etiquetas en el documento XML, cada uno tiene su tipo, un tipo es el ***artículo***, y el otro tipo son las ***ventas*** del artículo. Estos a su vez estarán compuestos por otras etiquetas *(<xsd:element..)*. En ***name*** se indica el nombre del elemento, y este nombre es el de las etiquetas que luego aparecen en el XML. En el ejercicio lo declaramos así:

```
<xsd:complexType name="VentasType">
  <xsd:sequence>
   <xsd:element name="articulo" type="DatosArtic"/>
   <xsd:element name="ventas" type="Ventas"/>
  </xsd:sequence>
 </xsd:complexType>
```

- El *element name* artículo lo utilizamos para representar los datos del artículo su tipo es ***type="DatosArtic"***. Este tipo está formado por las etiquetas del artículo. Es un tipo complejo y lo representamos así:

```
<xsd:complexType name="DatosArtic">
  <xsd:sequence>
   <xsd:element name="codigo" type="xsd:string"/>
   <xsd:element name="denominacion" type="xsd:string"/>
   <xsd:element name="stock" type="xsd:integer"/>
   <xsd:element name="precio" type="xsd:decimal"/>
  </xsd:sequence>
 </xsd:complexType>
```

Para indicar los tipos de datos que van a contener las etiquetas utilizamos:

type="xsd:string", para representar cadenas,
type="xsd:integer", para los Integer,
type="xsd:decimal", para los Float.

¡INTERESANTE!

Para saber más sobre tipos XSD podemos consultar las siguientes URLs:

http://www.w3schools.com/xml/schema_dtypes_string.asp
http://www.w3schools.com/xml/schema_dtypes_date.asp
http://www.w3schools.com/xml/schema_dtypes_numeric.asp
http://www.w3schools.com/xml/schema_dtypes_misc.asp
http://www.w3schools.com/xml/schema_example.asp

- El *element name* ventas lo declaramos para representar los datos de las ventas del artículo, su tipo es ***type="Ventas"*** y se llama ventas. Cada artículo podrá tener varias ventas (cada detalle de venta se llamará ***venta***). Es un tipo complejo y lo representamos así:

```
<xsd:complexType name="Ventas">
  <xsd:sequence>
   <xsd:element name="venta" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="numventa" type="xsd:integer"/>
       <xsd:element name="unidades">
         <xsd:simpleType>
           <xsd:restriction base="xsd:positiveInteger">
             <xsd:maxExclusive value="100"/>
           </xsd:restriction>
         </xsd:simpleType>
       </xsd:element>
       <xsd:element name="nombrecliente" type="xsd:string"/>
       <xsd:element name="fecha" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
```

- Para indicar que del tipo ventas puede haber varias ocurrencias (un artículo puede tener varias ventas) utilizamos los atributos ***minOccurs*** y ***maxOccurs,*** mínimo número de filas y sin límite en el máximo. Cada detalle de venta se va a llamar ***venta***. Lo escribimos así:

```
<xsd:element name="venta" minOccurs="1" maxOccurs="unbounded">
```

- También para cada elemento ***unidades*** se ha añadido una restricción, será un número positivo y el valor máximo va a ser 100. Lo escribimos así:

```
<xsd:element name="unidades">
        <xsd:simpleType>
           <xsd:restriction base="xsd:positiveInteger">
             <xsd:maxExclusive value="100"/>
           </xsd:restriction>
        </xsd:simpleType>
</xsd:element>
```

¡INTERESANTE!

Para saber más sobre restricciones en esquemas XSD podemos consultar las siguientes URLs:

http://www.w3schools.com/xml/el_restriction.asp
http://www.w3schools.com/xml/schema_facets.asp

Una vez que tenemos el esquema xsd, un ejemplo de un documento XML, con este esquema podría ser el siguiente: observa la raíz del documento <*ventasarticulos*>, los datos del artículo <*articulo*>, las ventas del artículo <*ventas*> y el detalle de cada venta <*venta*>:

```
<?xml version="1.0" encoding="UTF-8"?>
<ventasarticulos xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='ventas_articulo.xsd'>

 <articulo>
      <codigo>ART-112</codigo>
      <denominacion>Pala Padel NOX</denominacion>
      <stock>20</stock>
      <precio>70</precio>
 </articulo>
 <ventas>
      <venta>
            <numventa>10</numventa>
            <unidades>2</unidades>
            <nombrecliente>Alicia Ramos</nombrecliente>
            <fecha>10-10-2015</fecha>
      </venta>
      <venta>
            <numventa>11</numventa>
            <unidades>2</unidades>
            <nombrecliente>Pedro García</nombrecliente>
             <fecha>15-10-2015</fecha>
      </venta>
      <venta>
            <numventa>12</numventa>
            <unidades>6</unidades>
            <nombrecliente>Alberto Gil</nombrecliente>
            <fecha>20-10-2015</fecha>
      </venta>
 </ventas>
</ventasarticulos>
```

TAMBIÉN VALIDA EL XML ESTE XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

 <xsd:element name="ventasarticulos" type="VentasType"/>

 <xsd:complexType name="VentasType">
  <xsd:sequence>
   <xsd:element name="articulo" type="DatosArtic"/>
   <xsd:element name="ventas" type="Ventas"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="DatosArtic">
  <xsd:sequence>
   <xsd:element name="codigo" type="xsd:string"/>
   <xsd:element name="denominacion" type="xsd:string"/>
   <xsd:element name="stock" type="xsd:integer"/>
   <xsd:element name="precio" type="xsd:decimal"/>
  </xsd:sequence>
 </xsd:complexType>

<xsd:complexType name="Ventas">
      <xsd:sequence>
            <xsd:element name="venta" type="Venta"
                         minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Venta">
      <xsd:sequence>
        <xsd:element name="numventa" type="xsd:integer"/>
      <xsd:element name="unidades">
        <xsd:simpleType>
           <xsd:restriction base="xsd:positiveInteger">
             <xsd:maxExclusive value="100"/>
           </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="nombrecliente" type="xsd:string"/>
      <xsd:element name="fecha" type="xsd:string"/>
    </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

1.9.3 Creación de una aplicación JAXB con eclipse.

Lo siguiente que vamos a hacer es crear las clases a partir del xsd (***ventas_articulo.xsd)***. Y luego trabajaremos con este documento XML para hacer operaciones, como visualizar los datos del XML, añadir, modificar o borrar ventas. El fichero xml con datos de un artículo y sus ventas se llamará ***ventasarticulos.xml.*** Pasos:

- Si se crea un proyecto MAVEN no es necesario añadir los jar, se añaden las siguientes dependencias.

Además de las dependencias JAXB, se necesita el plugin jaxb2-maven-plugin y sus dependencias: https://search.maven.org/artifact/org.codehaus.mojo/jaxb2-maven-plugin

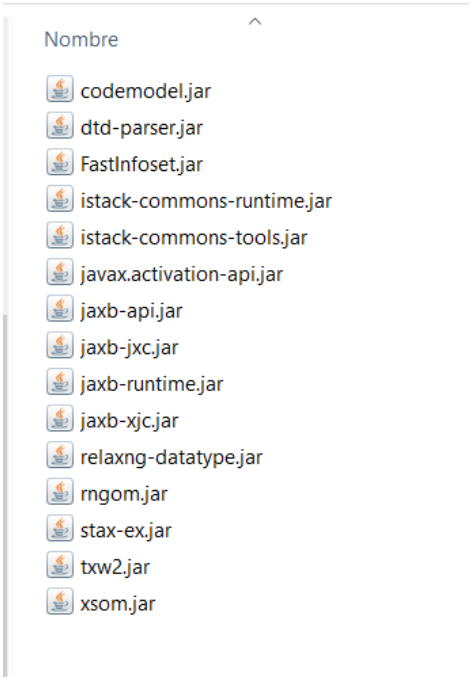
```
<dependencies>

	<!-- Dependencia JAXB para Java 11 o superior -->
	<dependency>
		<groupId>javax.xml.bind</groupId>
		<artifactId>jaxb-api</artifactId>
		<version>2.3.1</version>
	</dependency>
	<dependency>
		<groupId>com.sun.xml.bind</groupId>
		<artifactId>jaxb-core</artifactId>
		<version>2.3.0.1</version>
	</dependency>
	<dependency>
		<groupId>com.sun.xml.bind</groupId>
		<artifactId>jaxb-impl</artifactId>
		<version>2.3.3</version>
	</dependency>

	<dependency>
		<groupId>org.codehaus.mojo</groupId>
		<artifactId>jaxb2-maven-plugin</artifactId>
		<version>2.5.0</version>
	</dependency>
</dependencies>

<build>
	<plugins>
		<plugin>
			<groupId>org.codehaus.mojo</groupId>
			<artifactId>jaxb2-maven-plugin</artifactId>
			<version>2.5.0</version>
		</plugin>
	</plugins>
</build>
```

- Si creamos un proyecto eclipse normal. Para esta prueba nos aseguramos de guardar el XSD dentro de la carpeta ***src*** del proyecto. Y el XML dentro de la carpeta raíz del proyecto.
- Añadimos los siguientes jar:



Para las últimas versiones de JAVA, a partir de JAVA 8, utilizaremos otros jar, los descargamos de: https://javaee.github.io/jaxb-v2/

La última versión es jaxb-ri-2.3.1.zip

Al descomprimir los jar están contenidos en la carpeta ***mod***.

- Generamos los tipos, es decir las clases, a partir del fichero XSD, ***ventas_articulo.xsd***: botón derecho sobre el **fichero *XSD* ->*Generate* ->*JAXB Classes.*** En la siguiente ventana se selecciona el proyecto, y se añade el nombre del paquete donde queremos que se guarden los tipos generados, y dejamos el resto de opciones. Ver figura:

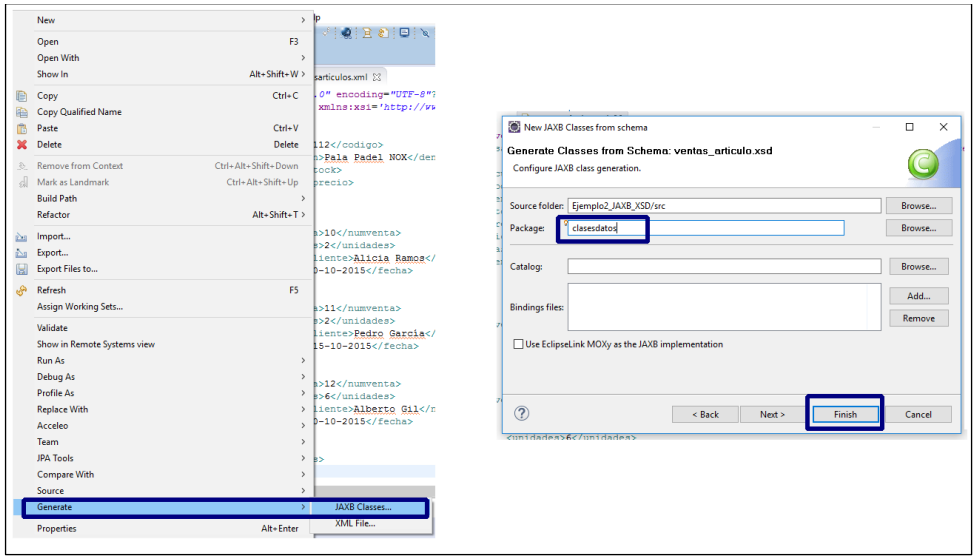
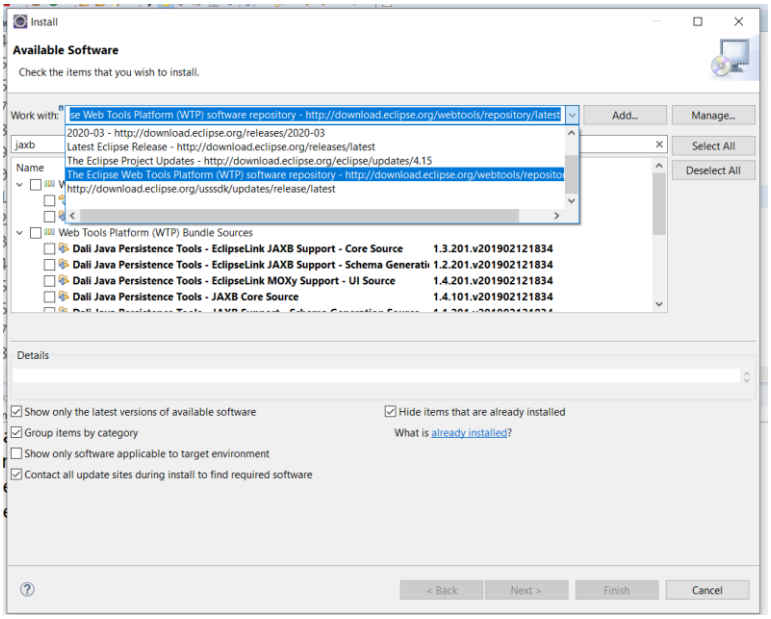
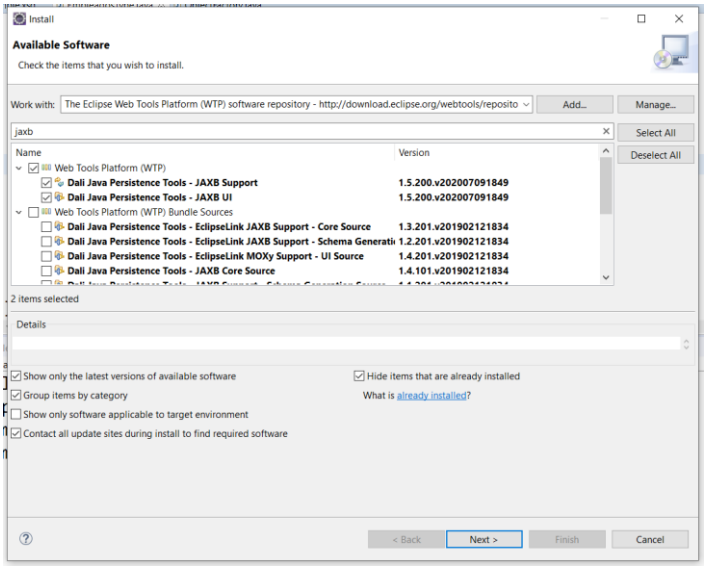


Figura 1.4. Creación de las clases JAXB.

Si no aparece se puede instalar desde Help/Install new software





- Una vez generados los tipos observa que las clases que se han creado se corresponden con los *types* del fichero ***ventas_articulo.xsd***. Ver figura:

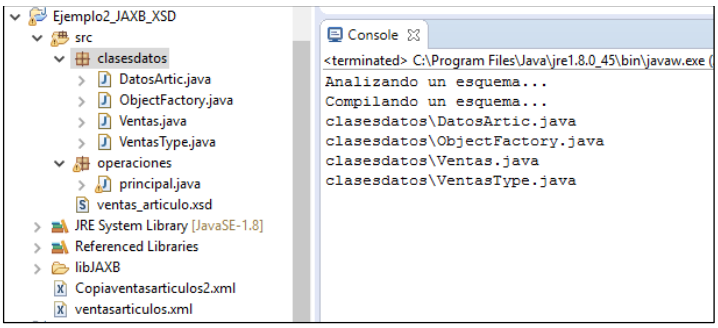


Figura 1.5. Creación de las clases JAXB.

Las clases creadas son *VentasType*, *DatosArtic*, y *Ventas* (el nombre de clase va en mayúscula).

```
<xsd:element name="ventasarticulos" type="VentasType"/>
<xsd:complexType name="VentasType">
  <xsd:sequence>
   <xsd:element name="articulo" type="DatosArtic"/>
   <xsd:element name="ventas" type="Ventas"/>
  </xsd:sequence>
 </xsd:complexType>
```

- Se crea además la clase ***ObjectFactory***, que nos va a permitir crear un *ObjectFactory* que se va a utilizar para crear nuevas instancias de las clases java del esquema XSD. En nuestro ejercicio creará instancias de las clases del paquete *clasesdatos*.

  Esta clase crea un objeto ***QName***. QName representa un nombre completo tal como se define en las especificaciones XML, está formado por el nombre del namespace (espacio de nombres URI, en el ejemplo va a espacios), y el nombre que hemos puesto al elemento principal en el XSD, se le llama prefijo local, en el ejercicio es ***ventasarticulos***. QName es inmutable, una vez creado no puede ser cambiado. La clase creada es la siguiente:

```
package clasesdatos;

import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlElementDecl;
import javax.xml.bind.annotation.XmlRegistry;
import javax.xml.namespace.QName;

@XmlRegistry
public class ObjectFactory {

 private final static QName _Ventasarticulos_QNAME
                   = new QName("", "ventasarticulos");

 public ObjectFactory() {   }

 public Ventas createVentas() {
        return new Ventas();   }

 public VentasType createVentasType() {
        return new VentasType();  }

 public DatosArtic createDatosArtic() {
        return new DatosArtic();   }

 public Ventas.Venta createVentasVenta() {
        return new Ventas.Venta();   }

 @XmlElementDecl(namespace = "", name = "ventasarticulos")
 public JAXBElement<VentasType>
    createVentasarticulos(VentasType value) {
       return new JAXBElement<VentasType>
         (_Ventasarticulos_QNAME, VentasType.class, null, value);
    }
}
```

  - **@XmlRegistry**, se utiliza para marcar una clase que tiene anotaciones *@XmlElementDecl*. Para implementar JAXB, hay que asegurar que se incluye en la lista de clases y se utilizan para arrancar el ***JAXBContext*** .
  - **@XmlElementDecl.** Si el valor del campo / propiedad va a ser un ***JAXBElement*** entonces se necesita añadir esta anotación. Un ***JAXBElement*** obtiene la siguiente información:

- Nombre del elemento, esto es necesario si se ha mapeado una estructura donde hay varios elementos del mismo tipo. En el ejercicio es *VentasType*.
- JAXBElement puede ser usado para representar un elemento con xsi : nil = "true".

- En nuestro programa java para crear el contexto JAXB, podemos utilizar el nombre del paquete que contiene a todas las clases, o el nombre de clase ObjectFactory:

```
JAXBContext contexto = JAXBContext.newInstance("clasesdatos");
```

O también:

```
JAXBContext contexto =
    JAXBContext.newInstance("clasesdatos.ObjectFactory.class");
```

Vamos ahora a crear una clase principal y con un método para visualizar el contenido del fichero xml ***ventasarticulos.xml*** utilizando este contexto, el método es el siguiente:

```
public static void visualizarxml() {
  System.out.println("------------------------------ ");
  System.out.println("-------VISUALIZAR XML--------- ");
  System.out.println("------------------------------ ");
  try {
      //Creamos el contexto
      JAXBContext jaxbContext =
          JAXBContext.newInstance(ObjectFactory.class);

      Unmarshaller u = jaxbContext.createUnmarshaller();

      JAXBElement jaxbElement = (JAXBElement) u.unmarshal
             (new FileInputStream("./ventasarticulos.xml"));

      Marshaller m = jaxbContext.createMarshaller();

      m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

      // Visualiza por consola
      m.marshal(jaxbElement, System.out);

      //Cargamos ahora el documento en los tipos
      VentasType miventa = (VentasType) jaxbElement.getValue();

      //Obtenemos una instancia para obtener todas las ventas
      Ventas vent = miventa.getVentas();

       // Guardamos las ventas en la lista
      List listaVentas = new ArrayList();
      listaVentas = vent.getVenta();

      System.out.println("---------------------------- ");
      System.out.println("---VISUALIZAR LOS OBJETOS--- ");
      System.out.println("---------------------------- ");
      // Cargamos los datos del artículo
      DatosArtic miartic = (DatosArtic) miventa.getArticulo();
      System.out.println("Nombre art: " +
               miartic.getDenominacion());
      System.out.println("Coodigo art: " + miartic.getCodigo());
      System.out.println("Ventas  del artículo , hay: " +
          listaVentas.size());
      //Visualizamos las ventas del artículo
      for (int i = 0; i < listaVentas.size(); i++) {
            Ventas.Venta ve = (Ventas.Venta) listaVentas.get(i);
            System.out.println("Número de venta: " +
```

```
               ve.getNumventa() + ". Nombre cliente: " +
               ve.getNombrecliente() + ", unidades: " +
               ve.getUnidades() + ", fecha: " + ve.getFecha());
            }
   } catch (JAXBException je) {
                  System.out.println(je.getCause());
   } catch (IOException ioe) {
                  System.out.println(ioe.getMessage());}
}
```

El siguiente método recibe datos de una venta y lo añade al documento XML. Se comprobará antes de insertar que el número de venta no exista:

```
private static void insertarventa
   (int numeventa, String nomcli, int uni, String fecha) {
  System.out.println("---------------------------- ");
  System.out.println("-------AÑADIR VENTA--------- ");
  System.out.println("---------------------------- ");
  try {
     JAXBContext jaxbContext =
             JAXBContext.newInstance(ObjectFactory.class);
           Unmarshaller u = jaxbContext.createUnmarshaller();
           JAXBElement jaxbElement = (JAXBElement)
     u.unmarshal(new FileInputStream("./ventasarticulos.xml"));

           VentasType miventa = (VentasType) jaxbElement.getValue();

           Ventas vent = miventa.getVentas();

           List listaVentas = new ArrayList();
           listaVentas = vent.getVenta();

           // comprobar si existe el número de venta,
     // reccorriendo el arraylist
           int existe = 0;
           for (int i = 0; i < listaVentas.size(); i++) {
              Ventas.Venta ve = (Ventas.Venta) listaVentas.get(i);
              if (ve.getNumventa().intValue() == numeventa) {
                  existe = 1; break;}
           }
           if (existe == 0) {
             // Crear el objeto Ventas.Venta
             Ventas.Venta ve = new Ventas.Venta();
             ve.setNombrecliente(nomcli);
             ve.setFecha(fecha); ve.setUnidades(uni);
             BigInteger nume = BigInteger.valueOf(numeventa);
             ve.setNumventa(nume);
       // Se añade la venta a la lista
       listaVentas.add(ve);
       //Se crea el Marshaller, volcar la lista al fichero XML
             Marshaller m = jaxbContext.createMarshaller();
             m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
             Boolean.TRUE);
       m.marshal(jaxbElement, new
              FileOutputStream("./ventasarticulos.xml"));
       System.out.println("Venta añadida: " + numeventa);
         } else
             System.out.println("En número de venta ya existe: " +
           numeventa);
  } catch (JAXBException je) {
            System.out.println(je.getCause());
```

```
  } catch (IOException ioe) {
            System.out.println(ioe.getMessage());}

}
```

ACTIVIDAD 1.8

Añade al proyecto anterior 3 métodos:

Método que reciba un número de venta y la borre, si existe, del documento XML. El método devolverá true si se ha borrado la venta y false si no se ha borrado o ha ocurrido algún error.

Método para modificar el stock del artículo. El método recibe una cantidad numérica y se debe sumar al stock del artículo. El método devuelve true si la operación se realiza correctamente y false si ocurre algún error.

Método para cambiar los datos de una venta, este método recibe el número de venta a modificar, las unidades, y la fecha. Se desean modificar esos datos del número de venta. El método devuelve true si la operación se realiza correctamente y false si ocurre algún error.

Ejemplo5:

En este ejemplo partimos del mismo XSD ***ventas_articulo.xsd***, y lo que vamos a hacer es crear un documento ***nuevo nuevo_ventasarticulo.xml,*** con datos de un artículo y dos ventas, que asignaremos manualmente. Los datos serán los siguientes:

Datos para el artículo:

Codigo	Denominación	Stock	Precio
Arti 1	Bicicleta Plegable	10	200

Datos para las ventas:

Num venta	Unidades	Nombre de cliente	Fecha
1	2	Alicia Ramos	10-02-2016
2	1	Dori Martín	21-02-2016

El código java es el siguiente:

```
public static void crearnuevoventasxml() {

  // Creo el objeto DatosArtic y asigno valores
  DatosArtic articulo = new DatosArtic();
  articulo.setCodigo("Arti 1");
  articulo.setDenominacion("Bicicleta Plegable");
  BigInteger stv = BigInteger.valueOf(10);
  BigDecimal pvv = BigDecimal.valueOf(200);
  articulo.setPrecio(pvv);
  articulo.setStock(stv);

  // Creamos el objeto Ventas
  Ventas ventas = new Ventas();
  // Creo la primera venta y la añado a ventas
  Ventas.Venta ven = new Ventas.Venta();
  ven.setNombrecliente("Alicia Ramos");
  ven.setNumventa(BigInteger.valueOf(1));
  ven.setUnidades(2);
  ven.setFecha("10-02-2016");
```

```
  ventas.getVenta().add(ven);

  // Creo la segunda venta y la añado a ventas
  ven = new Ventas.Venta();
  ven.setNombrecliente("Dori Martín");
  ven.setNumventa(BigInteger.valueOf(2));
  ven.setUnidades(1);
  ven.setFecha("21-02-2016");
  ventas.getVenta().add(ven);

  // Creo un VentasType y asigno los datos del artículo y sus ventas
  VentasType miventaarticulo = new VentasType();
  miventaarticulo.setArticulo(articulo);
  miventaarticulo.setVentas(ventas);

  // Creo el ObjectFactory
  ObjectFactory miarticulo = new ObjectFactory();
  // Creo El JAXBElement lo obtenemos del ObjectFactory y del VentasType
  JAXBElement<VentasType> element =
        miarticulo.createVentasarticulos(miventaarticulo);
  // Creo el contexto y el Marshaller
  JAXBContext jaxbContext;
  try {
      jaxbContext = JAXBContext.newInstance(ObjectFactory.class);
      Marshaller m = jaxbContext.createMarshaller();
      m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
            Boolean.TRUE);
      String fiche = "./nuevo_ventasarticulos.xml";
      m.marshal(element, new FileOutputStream(fiche));
      System.out.println("Venta creada. ");

      // Visualizamos por por consola
       m.marshal(element, System.out);

      } catch (JAXBException e) {e.printStackTrace();
      } catch (FileNotFoundException e){ e.printStackTrace();}
}
```

ACTIVIDAD 1.9

Crea un esquema XSD, con nombre ***centrosprofes.xsd*** para representar los datos de un centro educativo y sus profesores.

Y luego crea una aplicación JAXB para crear un fichero XML llamado ***micentro.xml,*** con los datos de un centro con tres profesores, y uno de ellos debe ser el director del centro.

La estructura del XML debe quedar de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<datoscentro xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:noNamespaceSchemaLocation='centrosprofes.xsd'>
  <centro>
      <codigocentro>11</codigocentro>
      <nombrecentro>111</nombrecentro>
      <direccion>111</direccion>
      <director>
            <codigoprofesor>11</codigoprofesor>
            <nombreprofe>11</nombreprofe >
      </director>
  </centro>
  <profesores>
      <profe>
```

```
            <codigoprofesor>12</codigoprofesor>
             <nombreprofe>11</nombreprofe >
      </profe>
       <profe>
            <codigoprofesor>13</codigoprofesor>
          <nombreprofe>11</nombreprofe >
      </profe>
  </profesores>
</datoscentro>
```

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">

<xsd:element name="datoscentro" type="DatosCentroType" />

<xsd:complexType name="DatosCentroType">
  <xsd:sequence>
     <xsd:element name="centro" type="CentroType" />
     <xsd:element name="profesores" type="ProfesoresType" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CentroType">
  <xsd:sequence>
       <xsd:element name="codigocentro" type="xsd:integer" />
       <xsd:element name="nombrecentro" type="xsd:string" />
       <xsd:element name="direccion" type="xsd:string" />
       <xsd:element name="director" type="ProfeType" />
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ProfesoresType">
   <xsd:sequence>
     <xsd:element name="profe" minOccurs="1" maxOccurs="unbounded" type="ProfeType" />
   </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="ProfeType">
        <xsd:sequence>
          <xsd:element name="codigoprofesor" type="xsd:integer" />
          <xsd:element name="nombreprofe" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```