

Raquel Sánchez Guirado



Tarea 06

Lenguaje de marcas

INTRODUCCIÓN	2
CASO B	2
CASO A	10
CASO C	12
ANEXO	15
FUENTES	17

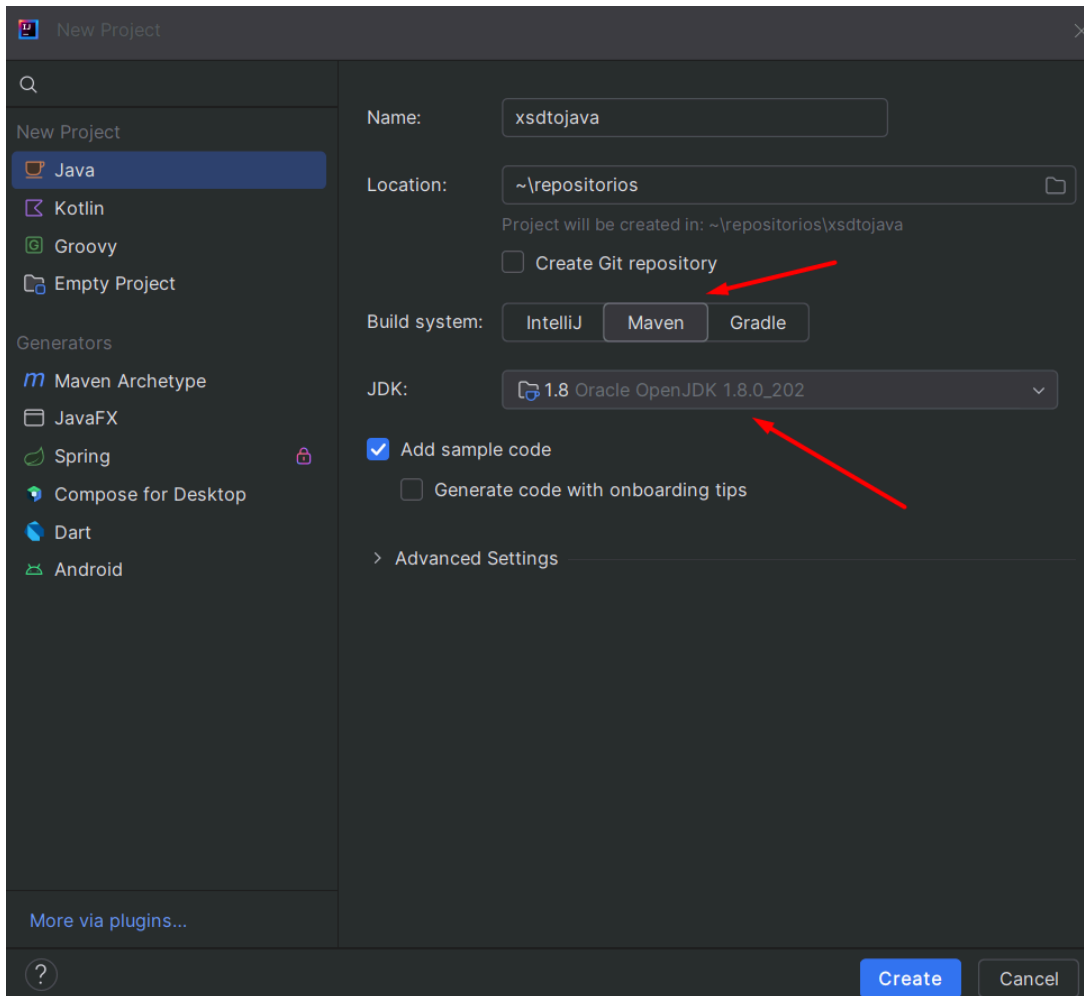
INTRODUCCIÓN

En este documento se mostrarán tres casos distintos de generación de clases java a partir de esquemas XSD. El orden de los casos es B - A - C para empezar investigando cómo utilizar el gestor de dependencias con el plugin de JAXB utilizando el xsd de la práctica anterior, que era relativamente complejo en mi caso. La información relativa a cada caso y los posibles errores encontrados se detallarán a continuación.

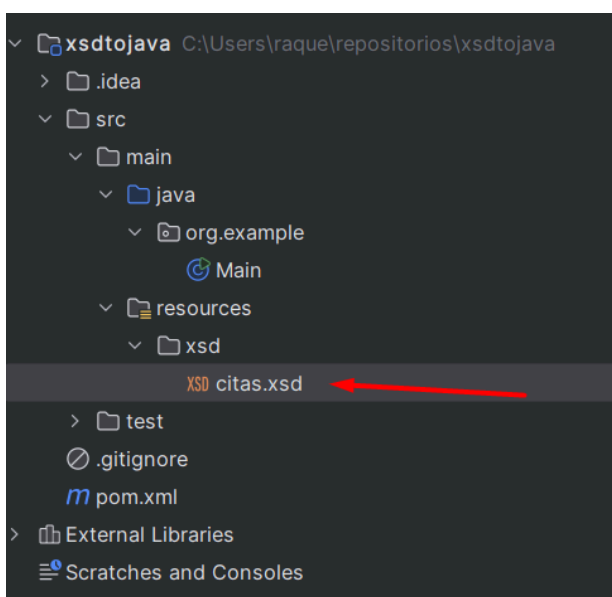
CASO B

En este segundo apartado, utilizaremos un xsd complejo con múltiples tipos y restricciones, utilizando el esquema de Maven pom.xml. Es importante tener instalado Java 8 ya que esta versión de java trae JAXB incluido por defecto, para evitar problemas con las versiones de java y las dependencias, crearemos un nuevo proyecto en IntelliJ cuyo build system sea Maven y el JDK 8:


```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.or
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>org.example</groupId>
8      <artifactId>xsdtojava</artifactId>
9      <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>8</maven.compiler.source>
13         <maven.compiler.target>8</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <build>
18         <plugins>
19             <plugin>
20                 <groupId>org.codehaus.mojo</groupId>
21                 <artifactId>jaxb2-maven-plugin</artifactId>
22                 <version>2.5.0</version>
23             </plugin>
24         </plugins>
25     </build>
26 </project>
```



Después, nos traemos el archivo XSD, en mi caso creo una carpeta dentro de resources que se llama xsd y muevo manualmente el archivo:



Es necesario añadir en pom.xml la dependencia de JAXB-2 que no es más que un plugin que se encarga de crear los objetos a partir de la base del xsd. Podemos encontrar esta dependencia en: <https://mvnrepository.com/artifact/org.codehaus.mojo/jaxb2-maven-plugin>


JAXB 2 Maven Plugin
 Mojo's JAXB-2 Maven plugin is used to create an object graph from XSDs based on the JAXB 2.x implementation and to generate XSDs from JAXB annotated Java classes.

License	Apache 2.0
Categories	Jenkins Plugins
Tags	plugin mojo binding build build-system jaxb maven xml jenkins codehaus
HomePage	https://github.com/mojohaus/jaxb2-maven-plugin/
Ranking	#130007 in MvnRepository (See Top Artifacts) #340 in Jenkins Plugins
Used By	3 artifacts

Central (17) | Redhat EA (2) | ICM (3)

Version	Vulnerabilities	Repository	Usages	Date
3.3.x 3.3.0		Central	0	Mar 31, 2025
3.2.x 3.2.0		Central	0	Apr 15, 2024
3.1.x 3.1.0		Central	0	Apr 21, 2022
2.5.x 2.5.0 ←		Central	0	Jul 12, 2019
2.4.x 2.4		Central	0	May 16, 2018
2.3.x 2.3.1		Central	2	Mar 08, 2017
2.3		Central	0	Sep 17, 2016
2.2.x 2.2		Central	1	Aug 10, 2015
2.1.x 2.1		Central	0	May 02, 2015

Copiamos el texto:

Maven | Gradle | SBT | Mill | Ivy | Grape | Leiningen | Buildr

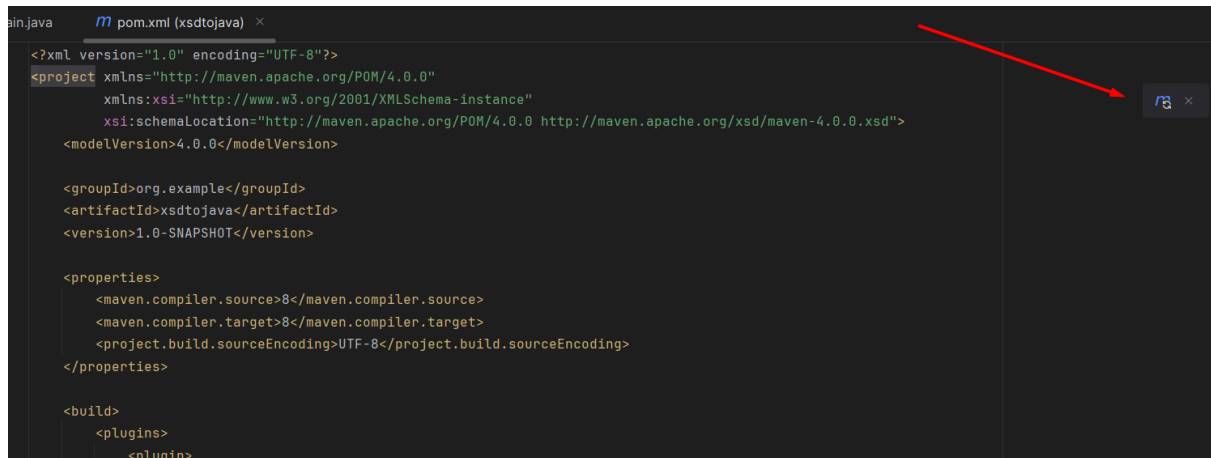
Scope: Compile

```

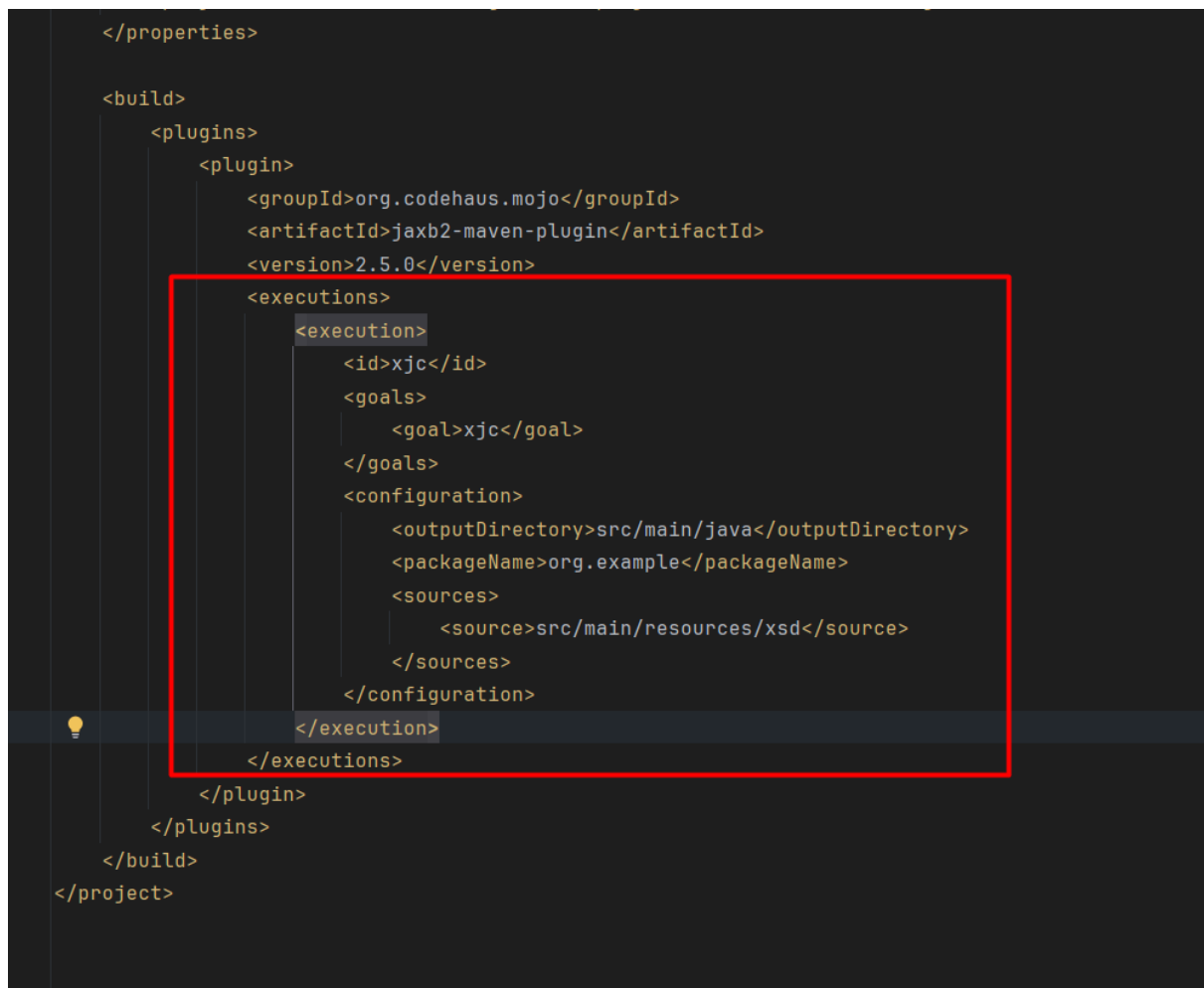
<!-- https://mvnrepository.com/artifact/org.codehaus.mojo/jaxb2-maven-plugin -->
<dependency>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>2.5.0</version>
</dependency>
  
```

En el pom.xml necesitamos abrir una etiqueta build y dentro otra que engloba todos los plugins, ahí dentro copiaremos la información obviando la etiqueta dependency (ver anexo 1b).

La primera vez que añadimos este código se marcarán rojo porque no está instalado, justo a la derecha IntelliJ nos muestra un botón para cargar maven que resolverá la instalación:

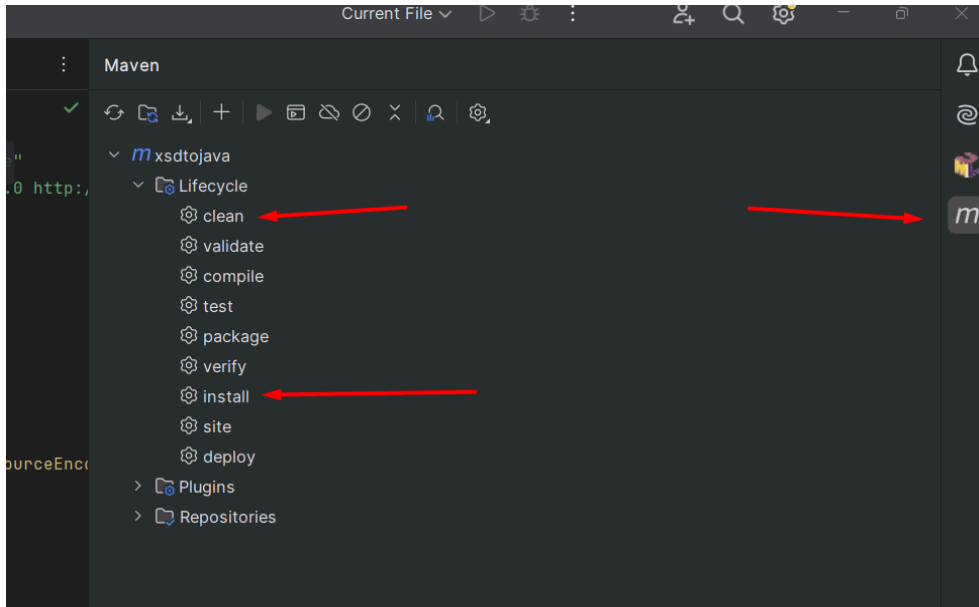


El siguiente paso es añadir la configuración necesaria para la ejecución que incluye varias etiquetas justo debajo de la versión (ver anexo 2b):

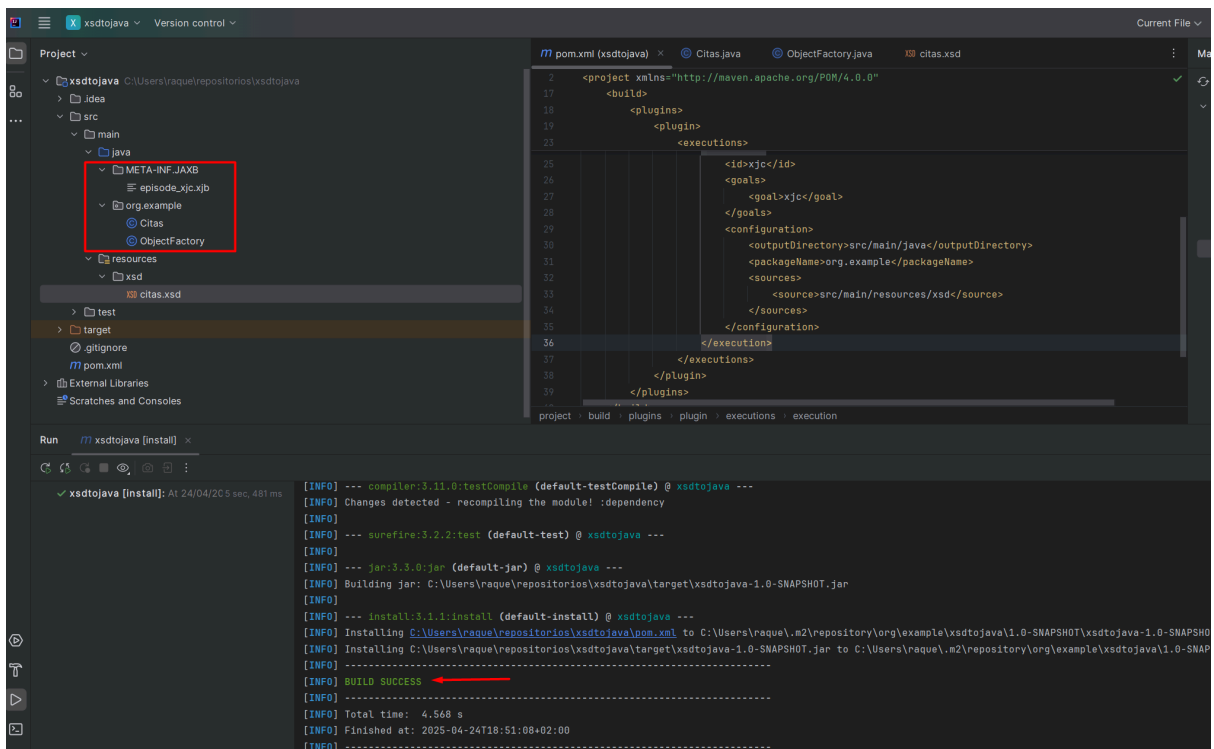


- outputDirectory: directorio donde se generarán los nuevos archivos
- packageName: nombre del paquete
- sources / source: directorio donde está el archivo xsd

Ahora debemos ejecutar dos funciones de Maven que son clean e install:



Una vez termine se mostrará en la consola un BUILD SUCCESS y comprobamos que los archivos se han creado:



*****En el proyecto el xsd a ejecutar para este caso se llama citas.xsd***

He utilizado el xsd de la tarea anterior, cuya etiqueta principal es citas, definido como un complexType. JAXB lo que hace es generar una sola clase que contiene clases internas o atributos, pero no divide estos archivos en varias clases.

Esto se podría solucionar si en cada complexType se agrega un nombre, de forma que JAXB las pueda detectar como una clase.

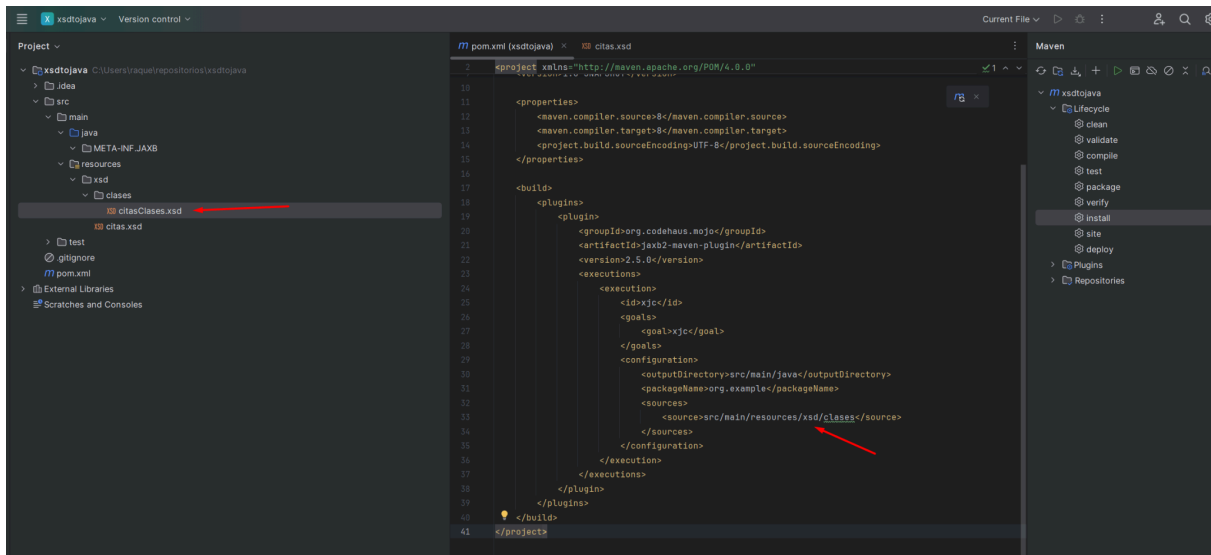
Ejemplo:

```
<xs:complexType name="CargoType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="importe" type="xs:decimal"/>
      <xs:element name="poliza" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

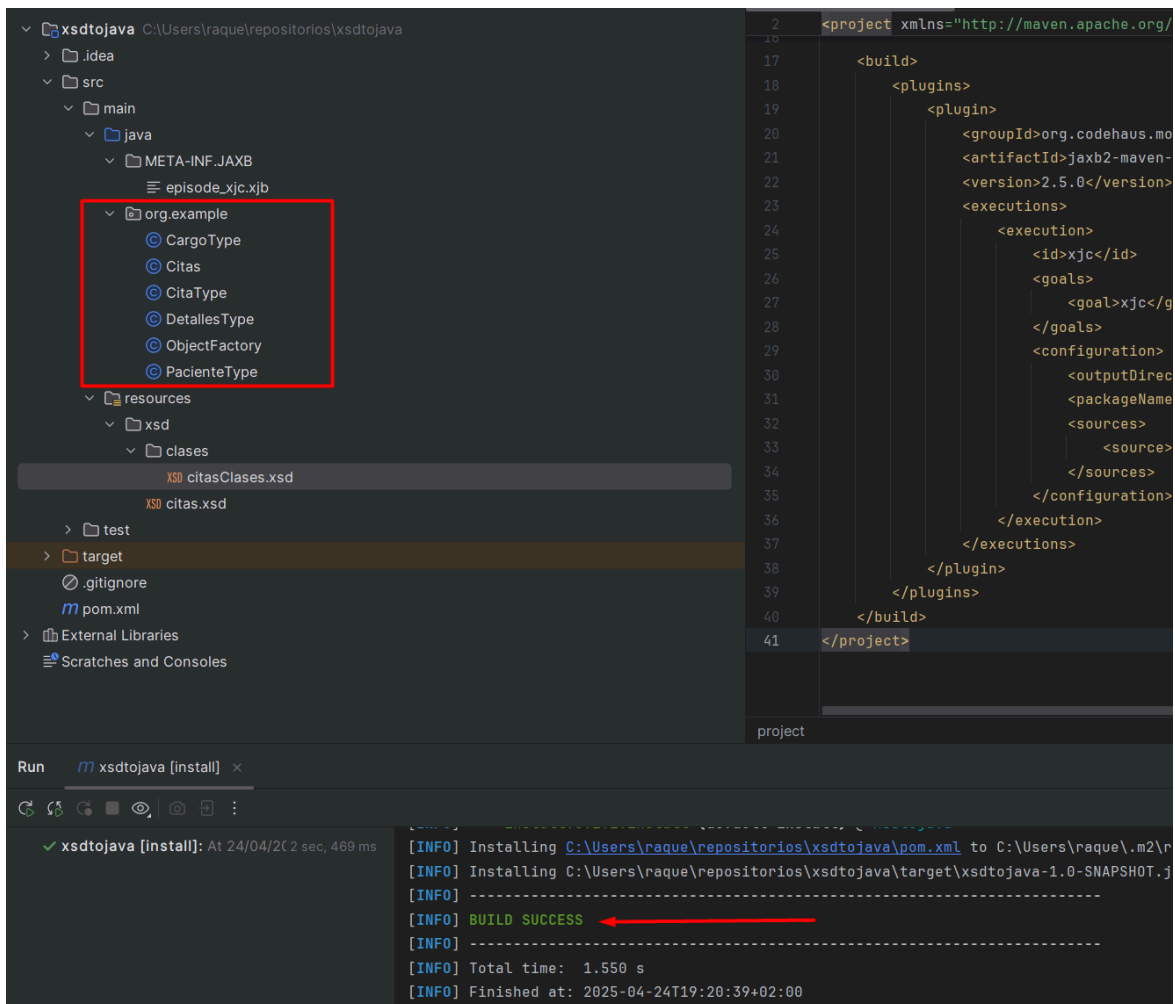
<xs:complexType name="PacienteType">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="fechaNacimiento" type="xs:date"/>
    <xs:element name="telefono">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="\d{9}"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="email" type="xs:string" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="identificador" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="DetallesType">
  <xs:sequence>
    <xs:element name="fechaCita" type="xs:date"/>
    <xs:element name="horario">
```

Procedemos a probar cambiando la configuración y añadiendo este nuevo XML:



Ejecutamos de nuevo clean install:



****En el proyecto el xsd a ejecutar para este caso se llama citasClasesCasoB.xsd**

CASO A

En este caso utilizaremos un xsd más sencillo de [w3schools](http://w3schools.com) y simplemente repetiremos los pasos del caso B, con la diferencia de que esta vez añadimos el xsd en la misma carpeta de resources/xsd donde ya habíamos introducido en el caso anterior un archivo citas.xsd. Veamos qué sucede:

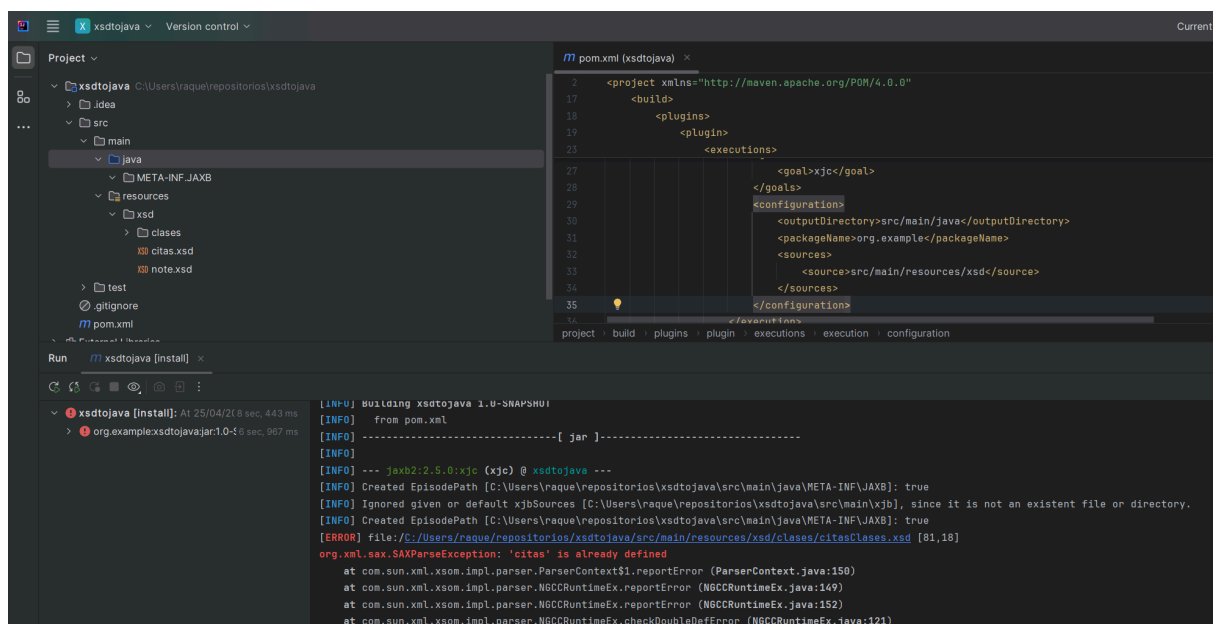
Configuramos la ruta donde están los dos archivos xsd:

```
<sources>

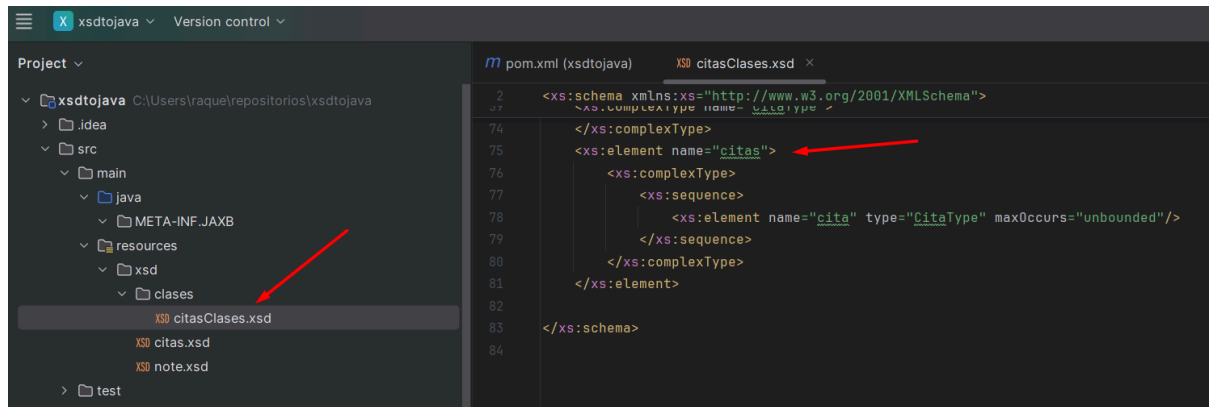
  <source>src/main/resources/xsd</source>

</sources>
```

Ejecutamos clean - install y observamos que se produce un error:



Esto sucede porque en el último paso del caso B maven ejecutó un xsd cuyo elemento raíz también es citas:



Entonces se produce un conflicto ya que citas.xsd tiene el mismo elemento raíz, por lo que JAXB es incapaz de diferenciar adecuadamente ambos esquemas.

Para poder ejecutar varios archivos xsd a la vez es necesario que tengan namespaces distintos, aunque otra posible solución sería simplemente añadir distintas configuraciones en el pom.xml para cada archivo. En el siguiente caso utilizaremos dos xsd relacionados entre sí para ver esto mejor.

*****En el proyecto el xsd a ejecutar para este caso se llama note.xsd***

CASO C

Para este caso, utilizaremos citasClases.xsd como en el caso B y lo relacionaremos con otro xsd que será paciente.xsd. Para ello, le he pedido a chatGPT que me elabore un xsd para relacionar los pacientes con el xsd de las citas.

En el xsd de citasClases añadimos la siguiente información del esquema para relacionar correctamente los dos archivos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:pac="http://example.com/paciente"
  targetNamespace="http://example.com/citas"
  xmlns="http://example.com/citas"
  elementFormDefault="qualified">

  <xs:import namespace="http://example.com/paciente" schemaLocation="paciente.xsd"/>

  <xs:complexType name="CargoType">
    <xs:sequence>
      <xs:choice>
        <xs:element name="importe" type="xs:decimal"/>
        <xs:element name="poliza" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="DetallesType">
    <xs:sequence>
      <xs:element name="fechaCita" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Y además eliminamos de este xsd la definición de paciente para añadirla en su correspondiente xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.com/paciente"
  xmlns:pac="http://example.com/paciente"
  elementFormDefault="qualified">

  <xs:complexType name="PacienteType">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellidos" type="xs:string"/>
      <xs:element name="fechaNacimiento" type="xs:date"/>
      <xs:element name="telefono">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="\d{9}"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="email" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="identificador" type="xs:string" use="required"/>
  </xs:complexType>
</xs:schema>
```

Al añadir los archivos a IntelliJ nos marca algunos errores como que había olvidado borrar una parte del paciente:

```

1  pom.xml (xsdtojava)  XSD citasClases.xsd  XSD paciente.xsd
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
18  <xs:complexType name="DetallesType">
19  <xs:sequence>
29  <xs:element name="especialidad">
30  <xs:simpleType>
39  </xs:restriction>
40  </xs:simpleType>
41  </xs:element>
42  <xs:element name="medico" type="xs:string" minOccurs="0"/>
43  <xs:element name="duracion" type="xs:int" minOccurs="0"/>
44  </xs:sequence>
45  </xs:complexType>
46
47  <xs:complexType name="CitaType">
48  <xs:sequence>
49  <xs:element name="cargo" type="CargoType"/>
50  <xs:element name="paciente" type="PacienteType"/>
51  <xs:element name="detalles" type="DetallesType"/>
52  </xs:sequence>
53  <xs:attribute name="urgente" type="xs:boolean"/>
54  <xs:attribute name="tipo">
xs:schema  xs:complexType  xs:sequence  xs:element

```

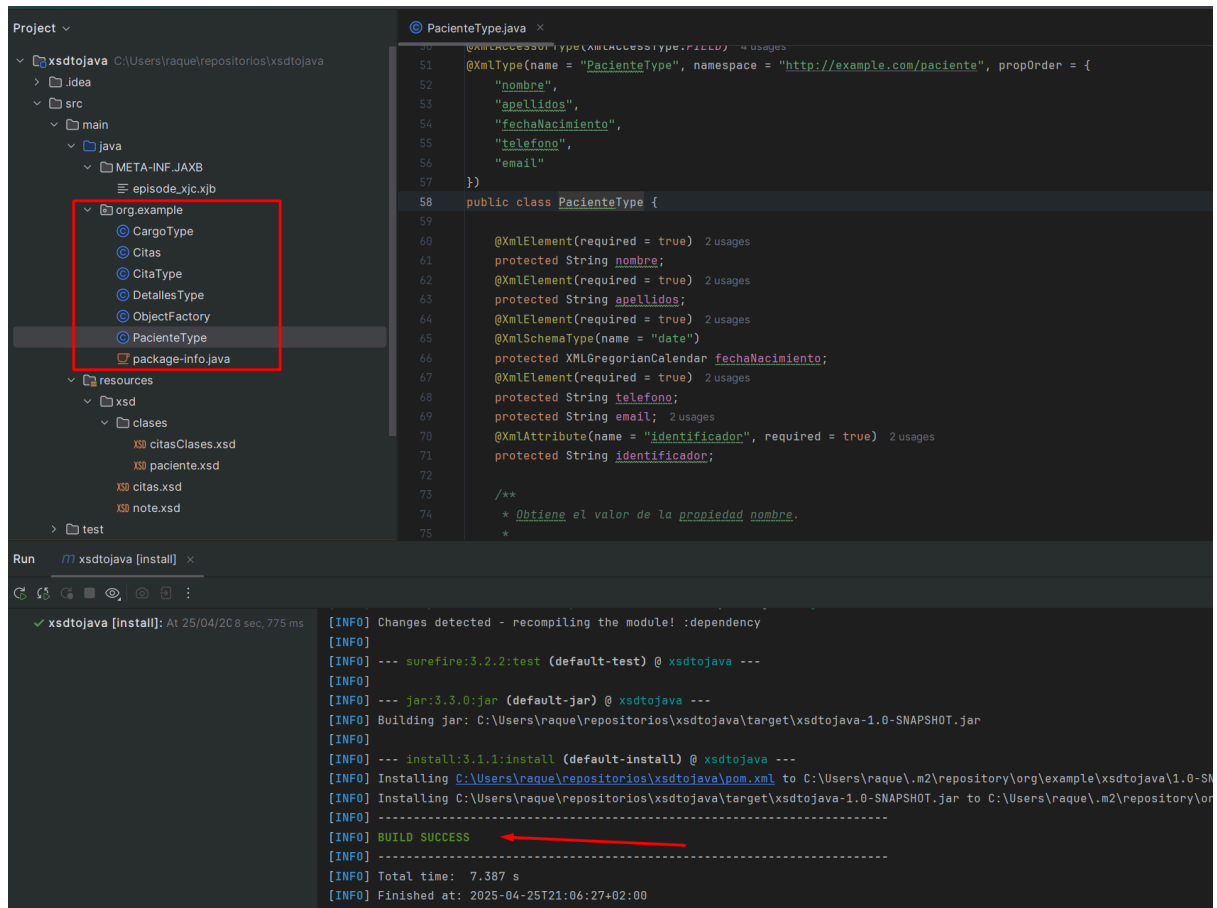
Además nos indica que no es necesaria la declaración del namespace porque no se está usando:

```

1  pom.xml (xsdtojava)  XSD citasClases.xsd  XSD paciente.xsd
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3  <!--
4  xmlns:pac="http://example.com/paciente"
5  targetNamespace="http://example.com/citas"
6  xmlns="http://example.com/citas"
7  elementFormDefault="qualified">
8
9  <xs:import namespace="http://example.com/paciente" schemaLocation="
10 <xs:complexType name="CargoType">
11 <xs:sequence>
12 <xs:choice>
13 <xs:element name="importe" type="xs:decimal"/>
14 <xs:element name="poliza" type="xs:string"/>
15 </xs:choice>
16 </xs:sequence>
17 </xs:complexType>
18 <xs:complexType name="DetallesType">
19 <xs:sequence>
20 <xs:element name="fechaCita" type="xs:date"/>
21 <xs:element name="horario">

```

Después de estos ajustes ejecutamos de nuevo clean install y observamos que se genera correctamente:



***En el proyecto los xsd a ejecutar para este caso se llaman citasClases.xsd y paciente.xsd*

ANEXO

Vídeo tutorial básico:

[Generar Clases Java desde XSD con Maven y JAXB.webm](#)

Enlace al repositorio:

<https://github.com/3592917/xsd-to-java>

1b. Contrastando con ChatGPT mis pasos, me indica que no es una dependencia como tal sino un plugin por lo que NO debe ir dentro de una etiqueta de dependencias sino dentro de build y plugins.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>org.example</groupId>
8      <artifactId>xsdtojava</artifactId>
9      <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>8</maven.compiler.source>
13         <maven.compiler.target>8</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <dependencies>
18         <dependency>
19             <groupId>org.codehaus.mojo</groupId>
20             <artifactId>jaxb2-maven-plugin</artifactId>
21             <version>3.3.0</version>
22         </dependency>
23     </dependencies>
24 </project>
  
```

2b. He intentado utilizar la última versión de jaxb2 (3.3.0) pero había que implementar un binding.xml y hay una versión anterior más sencilla que no requiere dicha implementación y solo necesita la configuración explicada anteriormente.

```
11 </properties>
13   <maven.compiler.target>8</maven.compiler.target>
14   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16
17 <build>
18   <plugins>
19     <plugin>
20       <groupId>org.codehaus.mojo</groupId>
21       <artifactId>jaxb2-maven-plugin</artifactId>
22       <version>3.3.0</version>
23       <executions>
24         <execution>
25           <id>xjc</id>
26           <goals>
27             <goal>xjc</goal>
28           </goals>
29           <configuration>
30             <outputDirectory>src/main/java</outputDirectory>
31             <packageName>org.example</packageName>
32             <sourceType>xsd</sourceType>
33             <sources>
34               <source>src/main/resources/xsd</source>
35             </sources>
36           </configuration>
37         </execution>
38       </executions>
39     </plugin>
40   </plugins>
41 </build>
42 </project>
```


FUENTES

Para la elaboración de esta tarea me he apoyado en la IA para entender mejor las diferencias que hay entre una versión y otra tanto de Maven como de Java, de forma que he podido implementar correctamente las versiones compatibles y adecuadas así como configurar el POM correctamente.

Por otro lado, también me he apoyado en la documentación oficial del plugin y de maven repository:

<https://mvnrepository.com/artifact/org.codehaus.mojo/jaxb2-maven-plugin>

https://www.mojohaus.org/jaxb2-maven-plugin/Documentation/v2.2/example_xjc_basic.html

Inicialmente, este vídeo me dio las orientaciones necesarias para saber cómo empezar:

<https://www.youtube.com/watch?v=-O9DQqPx8es>

Vídeo que explica la posible regeneración de clases:

<https://www.youtube.com/watch?v=c4qBZ0CntVE>