

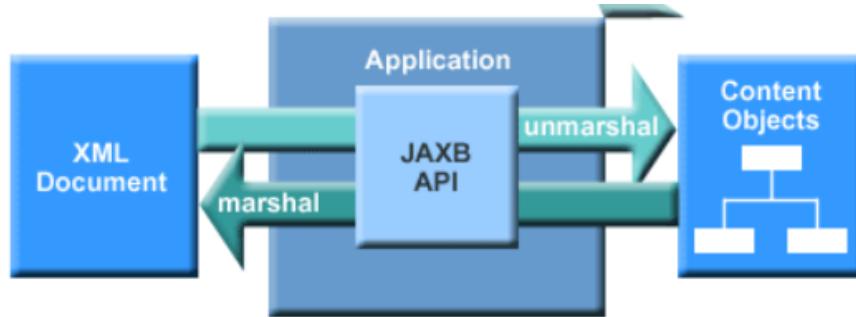
# UNIDAD 1 PARTE 2: JAXB XML

## Índice

<b>1.</b>	<b>JAXB</b> .....	<b>2</b>
<b>2.</b>	<b>Evolución, versiones y cambios importantes de JAXB</b> .....	<b>2</b>
<b>3.</b>	<b>Dependencias necesarias</b> .....	<b>3</b>
<b>4.</b>	<b>Anotaciones JAXB</b> .....	<b>4</b>
<b>4.1</b>	Las clases a mapear : .....	4
<b>4.2</b>	Cómo decide JAXB qué convertir a XML.....	4
<b>4.3</b>	Anotaciones JAXB básicas.....	6
<b>4.4</b>	Trabajar con annotation .....	7
<b>5.</b>	<b>JAXBContext JAXBContext, Marshaller y Unmarshaller</b> .....	<b>20</b>
<b>5.1</b>	<b>Convetir objetos Java en Documento XML</b> .....	<b>21</b>
<b>5.2</b>	<b>Mapeo de XML a objetos Java</b> .....	<b>23</b>

## 1. JAXB

JAXB (Java Architecture for XML Binding) es la API estándar desarrollada por Sun Java para mapear un objeto Java a un documento XML ("[marshal](#)") y el proceso contrario, es decir, a partir de un esquema XML crear su conjunto de objeto Java asociado ([unmarshal](#)).



El mapeo entre las clases java y el documento XML se describe a través de las [anotaciones](#).

De esta forma, el programador **trabaja directamente con clases y atributos**, sin necesidad de recorrer nodos ni emplear estructuras complejas como ocurre con DOM , SAX o STAX.

JAXB es especialmente útil cuando necesitamos intercambiar información con otros sistemas, almacenar configuraciones o generar documentos estructurados de forma automática.

## 2. Evolución, versiones y cambios importantes de JAXB

### Origen y contexto

- **JAXB (Java Architecture for XML Binding)** nació como parte de **Java EE** para simplificar el trabajo con XML.
- Su objetivo: **mapear clases Java ↔ documentos XML** de forma automática, evitando el uso manual de parsers como DOM o SAX.
- Se integró en el **JDK desde Java 6 hasta Java 8**, lo que lo convirtió en estándar de facto para persistencia en XML.

### Cambios importantes por etapas

A lo largo de su evolución, JAXB ha sufrido cambios significativos. Los más importantes son:

#### 1. JAXB incluido en el JDK (Java 6–8)

- Formaba parte del propio JDK.
- No era necesario añadir librerías externas.
- Usaba el paquete **javax.xml.bind**.
- Versiones habituales: 2.1, 2.2.

#### 2. Deprecación en Java 9 y 10

- JAXB pasa a estar marcado como *deprecated*.
- Se anuncia su futura eliminación.

#### 3. Eliminación en Java 11+

- JAXB desaparece completamente del JDK.
- El paquete **javax.xml.bind** ya no existe.
- Es obligatorio añadir dependencias externas (JARs o Maven).

#### 4. Nacimiento de Jakarta JAXB

- La API se traslada desde Java EE a Eclipse Foundation.
- El paquete cambia a jakarta.xml.bind.
- Aparecen las versiones modernas: JAXB 3.x y **JAXB 4.x**.
- Requiere **Java SE 11 o superior**
- Eliminación definitiva de clases y métodos obsoletos.
- Mejoras de rendimiento y simplificación en JAXBContext.
- Se consolida como **estándar actual para acceso a datos XML**.

#### 5. Lo que NO cambia

- Marshalling y unmarshalling funcionan igual.
- Las anotaciones (@XmlRootElement, @XmlElement, etc.) se usan del mismo modo.
- La lógica de trabajo con JAXB es prácticamente idéntica.

## 3. Dependencias necesarias

Dado que JAXB fue **eliminado del JDK** a partir de Java 11, es **obligatorio** que incorporemos manualmente las librerías necesarias a nuestro proyecto. Podemos hacerlo mediante Maven o añadiendo los JAR en el IDE.

#### Añadir manualmente los Jar a tu proyecto

- Descarga **ambos JAR**: jakarta.xml.bind-api-4.0.5.jar (API Jabx) y jaxb-runtime-4.0.5.jar (implementación principal).
- Para que la referencia de GlassFish JAXB funcione sin que de una excepción de NoClassDefFoundError debes incluir también las dependencias auxiliares que el runtime usa:  
**jaxb-core-4.x.jar** (core; contiene ClassFactory), **jakarta.activation-api-2.x.jar** (activation / DataHandler), **istack-commons-runtime-4.x.jar** (contiene com.sun.istack.Pool)
- Añádelos a tu proyecto

IDE	Dónde añadir JAR
NetBeans	Propiedades → Libraries → Add JAR/Folder
IntelliJ	Project Structure → Modules → Dependencies → Add JAR

#### Dependencias Maven recomendadas

**Maven** es una herramienta de automatización y gestión de proyectos que simplifica la inclusión de librerías externas. En lugar de descargar manualmente los archivos JAR, solo es necesario declarar las dependencias en el archivo de configuración del proyecto: el pom.xml.

```
<dependencies>
  <dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.2</version>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>4.0.6</version>
  </dependency>
  <dependency>
    <groupId>com.sun.activation</groupId>
    <artifactId>jakarta.activation</artifactId>
    <version>2.1.3</version>
  </dependency>
</dependencies>
```

Maven descargará jaxb-core, istack-commons-runtime automáticamente.

## 4. Anotaciones JAXB

Las anotaciones JAXB controlan como se realiza el mapeo entre los elementos del documento XML y los objetos Java.

- Las anotaciones están dentro del paquete **javax.xml.bind.annotation** (JAXB 2.x) o **jakarta.xml.bind.annotation** (JAXB 3.x/4.x).
- Empiezan por **@XMLXXXXXX**.
- Se aplican sobre clases, atributos o métodos.
- **Se incluyen dentro de las clases Java** que se quieren mapear a documentos XML o viceversa.  
Como mínimo la clase que actúe como raíz del documento XML deberá anotarse con **@XmlRootElement** (Es una anotación a nivel de clase).

### 4.1 Las clases a mapear :

- Deben cumplir **las características de un JavaBean**. Esto garantiza que JAXB pueda acceder, leer y modificar los valores de los atributos.  
Los JavaBeans son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.
- Las convenciones requeridas son:
  - **Debe tener un constructor sin argumentos.** (Puede haber varios constructores, pero uno siempre sin argumentos)
  - Sus **propiedades deben ser accesibles** mediante métodos **get y set** que siguen una convención de nomenclatura estándar.

Convención	Descripción	Ejemplo
Constructor vacío	La clase debe tener un constructor sin parámetros	<code>public Alumno() {}</code>
Métodos get/set	Acceso a propiedades mediante métodos públicos	<code>getNombre() / setNombre(String nombre)</code>
Nomenclatura estándar	El nombre del método refleja el atributo	<code>getEdad() ↳ atributo edad</code>
Encapsulación	Atributos privados, acceso controlado	<code>private String nombre;</code>

### 4.2 Cómo decide JAXB qué convertir a XML

Cuando usas JAXB para convertir un objeto Java en XML, el sistema necesita saber **qué atributos y qué métodos** de la clase deben aparecer en el XML. Esto se controla de dos formas:

- **Por defecto**, siguiendo las reglas estándar de JAXB.
- **Opcionalmente**, modificando ese comportamiento con la anotación **@XmlAccessorType**.

#### ¿Qué se convierte a XML por defecto?

- Si no se configura nada, **JAXB transforma automáticamente**:
  - Todos los **atributos públicos** (que no sean static ni transient).
  - Todos los **métodos getXXX() / setXXX()** públicos.
- El **nombre de la etiqueta XML coincide con el nombre del atributo o propiedad**.

Problema Común: Si una clase tiene un campo public Y su par de getter/setter, JAXB podría intentar mapear ambos, lo que resultaría en elementos duplicados o XML no válido.

Ejemplo:

```
import jakarta.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Alumno {
    // Atributo privado
    private String nombre;

    // Constructor sin argumentos (requisito JavaBean)
    public Alumno() {}

    // Getter y Setter públicos
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

Alumno a = new Alumno();
a.setNombre("Manuel");

Resultado XML (por defecto)
<alumno>
    <nombre>Manuel</nombre>
</alumno>
```

Explicación:

- ✓ **@XmlRootElement**: indica que la clase será el **elemento raíz** del XML.
- ✓ **Nombre por defecto**: JAXB usa el **nombre de la clase en minúsculas** → Alumno → <alumno>.
- ✓ **Atributos privados + getters/setters públicos**: JAXB sigue la convención JavaBean, así que aunque el atributo sea privado, si tiene getNombre() y setNombre(), se convierte en un elemento XML <nombre>.
- ✓ **Valor del objeto**: el contenido del XML refleja el valor asignado en el objeto Java (Manuel).

### Controlando el Mapeo con **@XmlAccessorType**

Para evitar el comportamiento por defecto (y sus duplicidades) y tener un control más preciso, usas la anotación **@XmlAccessorType** a nivel de clase.

Esta define la **estrategia de acceso** a las propiedades de la clase para la serialización.

#### ■ **@XmlAccessType.FIELD** (Acceso a Campo)

- **Mapeo**: JAXB mapeará **todos los atributos de la clase** (campos), independientemente de su visibilidad (private, protected, etc.), siempre y cuando no sean static o transient.
- **Ignora**: Ignora completamente los **getters y setters**.
- **Recomendación**: Es la opción **más limpia y común**. Permite encapsular los datos como private y usar getters y setters solo para la lógica interna de Java, sin que afecten al XML.

#### ■ **@XmlAccessType.PROPERTY** (Acceso a Propiedad)

- **Mapeo**: JAXB mapeará **solo las propiedades públicas** (los pares getter/setter).
- **Ignora**: Ignora los campos directos de la clase.
- **Uso**: Útil si quieras que el XML se genere a partir de un valor que se **calcula** en el método *getter* en lugar de un campo almacenado.

■ **@XmlAccessType.NONE** (Sin Acceso por Defecto)

- Mapeo:** No mapea absolutamente nada de forma automática.
- Uso:** Solo se serializarán los campos o métodos que estén explícitamente anotados con `@XmlElement` o `@XmlAttribute`. Ofrece el máximo nivel de control manual.

■ **XmlAccessType.PUBLIC\_MEMBER** (comportamiento por defecto).

Tipo	Qué se mapea	Uso recomendado
FIELD	Todos los atributos (no estáticos ni transient). Ignora getters/setters.	Opción más limpia y común.
PROPERTY	Solo propiedades públicas (get/set). Ignora campos directos.	Útil si el valor se calcula en el getter.
NONE	No mapea nada automáticamente. Solo lo anotado explícitamente.	Máximo control manual.
PUBLIC_MEMBER	Comportamiento por defecto: atributos y propiedades públicas.	Caso estándar.

## 4.3 Anotaciones JAXB básicas

- @XmlRootElement**: Se define el elemento raíz de un documento XML. Obligatorio
- @XmlRootElement(name = "xxx")**: Define que el elemento raíz del objeto Java se llamará xxx.
- @XmlType**: Podemos utilizarlo para ordenar los elementos en el XML.
- @XmlTransient**: el atributo del objeto no se escribe en el XML.
- @XmlAttribute**: el campo del objeto se mapea como atributo.
- @XmlElement (name = "abc")**: Se creará el elemento con el nombre "abc".

Ejemplo de crear una clase para mapearla a XML con los valores por defecto:

ListaPuntos.java

```
//Obligatorio indicar el elemento raiz
@XmlRootElement()
public class ListaPuntos {
    private ArrayList <Puntos> lista;

    public ListaPuntos() {
        lista = new ArrayList<>();
    }

    public ArrayList<Puntos> getList() {      return lista;      }
    public void setList(ArrayList<Puntos> lista) {      this.lista = lista;      }
}
```

Puntos.java

```
package ejJAXBserialobjetosxml;
public class Puntos {
    private int x;
    private int y;
    //Muy importante: LA CLASE DEBE TENER UN CONSTRUCTOR POR DEFECTO, SINO DA ERROR
    public Puntos() {}
    public Puntos(int x,int y){
        this.x=x;
        this.y=y;
    }
    public int getX() {      return x;      }
    public void setX(int x) {      this.x = x;      }
    public int getY() {      return y;      }
    public void setY(int y) {      this.y = y;      }
}

<!-- Elemento raíz: la clase ListaPuntos está anotada con @XmlRootElement --&gt;
&lt;listapuntos&gt;
    &lt;!-- Cada objeto Puntos dentro del ArrayList se convierte en un elemento &lt;lista&gt; --&gt;</pre>

```

```

<lista>
    <!-- Atributo x del objeto Puntos -->
    <x>5</x>
    <!-- Atributo y del objeto Puntos -->
    <y>8</y>
</lista>
<lista>
    <x>1</x>
    <y>9</y>
</lista>
<lista>
    <x>3</x>
    <y>7</y>
</lista>
<lista>
    <x>9</x>
    <y>5</y>
</lista>
</listapuntos>

```

## 5. Reglas de Conversión de Tipos

La **conversión de tipos (binding)** es el proceso que **convierte automáticamente valores Java a su representación XML ( Schema XSD)** al hacer **marshalling**, y la operación inversa al hacer **unmarshalling**. Aquí tienes unas apuntes completos, claros y listos para clase.

### 5.1 Principios generales

- JAXB mapea clases Java anotadas a elementos y tipos XML, y campos a elementos o atributos según las anotaciones (@XmlRootElement, @XmlType, @XmlElement, @XmlAttribute, @XmlValue).
- Marshalling: Java → XML. Unmarshalling: XML → Java.
- Acceso: controla si JAXB usa campos o propiedades con @XmlAccessorType(XmlAccessType.FIELD | PROPERTY | PUBLIC\_MEMBER).
- Nombres: usa explícitamente @XmlElement(name="...") y @XmlAttribute(name="...") para evitar ambigüedades.
- Orden de elementos: controla el orden con @XmlType(propOrder = { ... }).

### 5.2 Conversión de tipos

Cuando JAXB realiza el **Marshalling (Java → XML)**, o **UnMarshalling (XML → JAVA)**, convierte los tipos de datos de Java a la representación XML o viceversa de la siguiente manera

#### Tipos Básicos (Primitivos y Envoltorios):

JAXB realiza la **conversión automática** (sin necesidad de anotaciones adicionales) para la mayoría de los tipos comunes de Java a sus equivalentes en XML Schema Definition (XSD).

Los tipos simples de Java (como String, int, double, boolean, etc.) se transforman en el **contenido de texto** de un elemento XML simple.

Tipo Java	Tipo XSD	XML	Como elemento XML	Como atributo XML
<b>String</b>	xsd:string		<nombre>Ana</nombre>	nombre="Ana"
<b>int</b>	xsd:int		<edad>30</edad>	edad="30"
<b>Integer</b>	xsd:int		<edad>30</edad>	edad="30"
<b>long</b>	xsd:long		<id>123456</id>	id="123456"

<b>short</b>	xsd:short	<nivel>2</nivel>	nivel="2"
<b>byte</b>	xsd:byte	<codigo>12</codigo>	codigo="12"
<b>float</b>	xsd:float	<altura>1.75</altura>	altura="1.75"
<b>double</b>	xsd:double	<precio>19.95</precio>	precio="19.95"
<b>boolean</b>	xsd:boolean	<activo>true</activo>	activo="true"
<b>Boolean</b>	xsd:boolean	<activo>false</activo>	activo="false"
<b>BigInteger</b>	xsd:integer	<granNumero>1234567890</granNumero>	granNumero="1234567890"
<b>BigDecimal</b>	xsd:decimal	<saldo>1000.50</saldo>	saldo="1000.50"
<b>java.util.Date</b> <b>Calendar</b>	/xsd:dateTime	<fecha>2025-11-17T00:00:00</fecha> FormatoISO-8601 (yyyy-MM-dd'T'HH:mm:ss)	fecha="2025-11-17T00:00:00"
<b>XMLGregorianCalendar</b>	xsd:dateTime	<fecha>2025-11-17T00:00:00</fecha>	fecha="2025-11-17T00:00:00"

### Objetos (Clases Anotadas)

- Una clase Java que contiene otra clase Java (anotada con JAXB) se convierte en un **elemento XML compuesto o anidado**.
  - Ejemplo:* Una clase Pedido que contiene un objeto Cliente → El XML del cliente será un sub-elemento dentro del XML del pedido.

```
// Clase Pedido
public class Pedido {
    // El campo 'cliente' es un objeto
    private Cliente cliente;
    //...
}
XML
<pedido>
    <cliente>...</cliente>
</pedido>
```

### Colecciones (Arrays, Listas, etc.):

- Las colecciones se convierten en un **conjunto de elementos XML repetidos**.

JAXB soporta directamente las colecciones más comunes, pero su flexibilidad disminuye cuando se trabaja con estructuras que no son ordenadas o que requieren pares clave-valor (Map).

JAXB puede serializar de forma automática cualquier colección que implemente la interfaz **java.util.Collection** (List o arrays nativos de Java).

Colección Java	Soporte JAXB	Cómo se maneja / Notas	Ejemplo XML
<b>List</b> (ArrayList, LinkedList)	<input checked="" type="checkbox"/> Sí	Se serializa directamente. Mantiene el orden de los elementos.	<productos> <producto>Teclado</producto> <producto>Ratón</producto> </productos>
<b>Arrays</b> (String[], int[], Producto[])	<input checked="" type="checkbox"/> Sí	Igual que List. Mantiene el orden.	<colores> <color>Rojo</color> <color>Verde</color> </colores>

Colección Java	Soporte JAXB	Cómo se maneja / Notas	Ejemplo XML
Vector	<input checked="" type="checkbox"/> Sí	Igual que List. Mantiene el orden.	<pre>&lt;items&gt;   &lt;item&gt;1&lt;/item&gt;   &lt;item&gt;2&lt;/item&gt; &lt;/items&gt;</pre>
Set (HashSet, TreeSet)	<input checked="" type="checkbox"/> Sí	Se serializa correctamente con etiquetas repetidas. Al deserializar, JAXB reconstruye el Set. <b>El orden no se conserva</b> , ya que un Set no garantiza orden. Si el orden es importante, usar List.	<pre>&lt;usuarios&gt;   &lt;usuario&gt;Ana&lt;/usuario&gt;   &lt;usuario&gt;Paco&lt;/usuario&gt; &lt;/usuarios&gt;</pre>
Map (HashMap, TreeMap)	<input type="checkbox"/> No	No soportado directamente. Usar XmlAdapter para serializar como lista de pares clave-valor.	<pre>&lt;mapa&gt;   &lt;entry key="clave1"&gt;     valor1&lt;/entry&gt;   &lt;entry key="clave2"&gt;     valor2&lt;/entry&gt; &lt;/mapa&gt;</pre>

#### ■ Set (java.util.Set) en JAXB

- **Serialización (marshalling):**
  - JAXB puede serializar un Set (por ejemplo, HashSet o TreeSet) porque implementa Collection.
  - Cada elemento del Set se convierte en una **etiqueta XML repetida** dentro de un contenedor.
- **Deserialización (unmarshalling):**
  - Al leer el XML, JAXB reconstruye un Set automáticamente.
- **Limitación clave:**
  - **El orden de los elementos no se conserva**, ya que un Set no garantiza orden.
  - Si necesitas mantener el orden en el XML, es mejor usar un List.

```
Set<String> usuarios = new HashSet<>();
usuarios.add("Ana");
usuarios.add("Paco");
xml
<usuarios>
  <usuario>Ana</usuario>
  <usuario>Paco</usuario>
</usuarios>
```

- Al deserializar, JAXB devuelve un Set<String> con los mismos elementos.
- El orden en que aparecen <usuario> en XML **no garantiza** el orden en el Set.

#### ■ Map (java.util.Map) en JAXB

- **Serialización (marshalling):**
  - JAXB **no soporta directamente Map** (por ejemplo, HashMap o TreeMap).
  - **Para serializarlo, se utiliza un XmlAdapter**, que convierte el Map en una **lista de entradas clave-valor** que JAXB sí puede manejar.
  - Cada entrada se convierte en un contenedor <entry> con subetiquetas <key> y <value>.
- **Deserialización (unmarshalling):**
  - Al leer el XML, JAXB utiliza el XmlAdapter para reconstruir el Map automáticamente.
- **Limitación clave:**
  - Si el Map original es un HashMap, **el orden de las entradas no está garantizado** en el XML ni al reconstruirlo.
  - Si se necesita orden, se puede usar LinkedHashMap y adaptar el XmlAdapter para mantenerlo.

```

Map<String, String> parametros = new HashMap<>();
parametros.put("usuario", "Ana");
parametros.put("contraseña", "1234");

public class MapAdapter extends XmlAdapter<MapAdapter.AdaptedMap, Map<String, String>> {
    public static class AdaptedMap {
        public List<Entry> entry = new ArrayList<>();
    }

    public static class Entry {
        public String key;
        public String value;
    }

    @Override
    public Map<String, String> unmarshal(AdaptedMap v) {
        Map<String, String> map = new HashMap<>();
        for (Entry e : v.entry) {
            map.put(e.key, e.value);
        }
        return map;
    }

    @Override
    public AdaptedMap marshal(Map<String, String> v) {
        AdaptedMap adaptedMap = new AdaptedMap();
        for (Map.Entry<String, String> e : v.entrySet()) {
            Entry entry = new Entry();
            entry.key = e.getKey();
            entry.value = e.getValue();
            adaptedMap.entry.add(entry);
        }
        return adaptedMap;
    }
}

Clase que usa el map
@XmlRootElement
public class Configuracion {

    @XmlJavaTypeAdapter(MapAdapter.class)
    private Map<String, String> parametros;

    // getters y setters
}

<configuracion>
    <parametros>
        <entry>
            <key>usuario</key>
            <value>Ana</value>
        </entry>
        <entry>
            <key>contraseña</key>
            <value>1234</value>
        </entry>
    </parametros>
</configuracion>

```

## Conversión de tipos de fecha en JAXB

JAXB no soporta directamente `java.time.LocalDate` ni otros tipos de `java.time` en la versión estándar. Solo soporta los tipos antiguos como `java.util.Date` o `java.util.Calendar`.

Para usar `LocalDate`, `LocalTime` o `LocalDateTime`, hay que utilizar un `XmlAdapter` que haga la conversión entre el tipo `java.time` y un tipo soportado por JAXB (`String`, `Date` o `Calendar`).

Tipo Java	Soporte directo JAXB	Necesita adaptador	Ejemplo XML
<code>java.util.Date</code>	<input checked="" type="checkbox"/> Sí	No	<fecha>2025-11-17T14:30:00</fecha>
<code>java.util.Calendar</code>	<input checked="" type="checkbox"/> Sí	No	<fecha>2025-11-17T14:30:00</fecha>
<code>java.time.LocalDate</code>	<input type="checkbox"/> No	Sí, usar XmlAdapter	<fecha>2025-11-17</fecha>

Tipo Java	Soporte directo JAXB	Necesita adaptador	Ejemplo XML
java.time.LocalTime	✗ No	Sí, usar XmlAdapter	<hora>14:30:00</hora>
java.time.LocalDateTime	✗ No	Sí, usar XmlAdapter	<fechaHora>2025-11-17T14:30:00</fechaHora>

### Control de Formato con @XmlSchemaType

Aunque estos tipos se mapean por defecto a xsd:dateTime, es muy común querer serializar solo la fecha (xsd:date) o solo la hora (xsd:time).

Para ello, utilizamos la anotación **@XmlSchemaType** sobre el campo:

```

@XmlRootElement
public class Empleado {

    // Se fuerza a serializar solo la fecha (yyyy-MM-dd)
    @XmlSchemaType(name = "date")
    private Date fechaNacimiento;

    // Se fuerza a serializar solo la hora (HH:mm:ss)
    @XmlSchemaType(name = "time")
    private Date horaEntrada;

    // getter y setter
}

```

XML resultante:

```

<empleado>
    <fechaNacimiento>1985-07-23</fechaNacimiento>
    <horaEntrada>09:00:00</horaEntrada>
</empleado>

```

### @XmlSchemaTypes

- Es un **contenedor** que permite aplicar **varios @XmlSchemaType** en el mismo ámbito (paquete, clase,) para distintos tipos.

```

@XmlSchemaTypes({
    @XmlSchemaType(name="date", type=java.util.Date.class),
    @XmlSchemaType(name="time", type=java.util.Calendar.class)
})
public class Evento {
    private Date fecha;
    private Calendar hora;
}

```

fecha se serializa como xsd:date y hora como xsd:time.

### Resumen de niveles de aplicación

Nivel	Cómo se aplica	Ejemplo
Propiedad / campo	Directamente sobre el atributo o getter/setter	<code>@XmlSchemaType(name="date") private Date fecha;</code>
Clase	Sobre la clase, se aplica a campos de cierto tipo	<code>@XmlSchemaTypes({     @XmlSchemaType(name="date",     type=Date.class) })</code>
Paquete	En <code>package-info.java</code> , aplica a todo el paquete	Ver ejemplo de paquete con <code>@XmlSchemaTypes</code>

## Nivel de paquete

- En el nivel de paquete se puede usar el archivo `package-info.java` con la anotación `@XmlSchemaType` para indicar **tipos predeterminados para ciertas clases Java** dentro del paquete.

## Ejemplo:

```
@jakarta.xml.bind.annotation.XmlSchema(
    namespace = "http://www.ejemplo.com/empleado",
    elementFormDefault = jakarta.xml.bind.annotation.XmlNsForm.QUALIFIED
)
@jakarta.xml.bind.annotation.XmlSchemaTypes({
    @XmlSchemaType(name="date", type=java.util.Date.class),
    @XmlSchemaType(name="dateTime", type=java.util.Calendar.class)
})
package com.ejemplo.empleado;
```

- Con esto, **todos los Date del paquete** se serializan por defecto como xsd:date y los Calendar como xsd:dateTime.

## Conversion de tipos con LocalDate, LocalTime o LocalDateTime

**JAXB no soporta directamente java.time.LocalDate ni otros tipos de java.time** en la versión estándar. Solo soporta los tipos antiguos como `java.util.Date` o `java.util.Calendar`.

Para usar `LocalDate`, `LocalTime` o `LocalDateTime`, **hay que utilizar un XmlAdapter** que haga la conversión entre el tipo `java.time` y un tipo soportado por JAXB (`String`, `Date` o `Calendar`).

```
public class LocalDateAdapter extends XmlAdapter<String, LocalDate> {
    private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

    @Override
    public LocalDate unmarshal(String v) {
        return LocalDate.parse(v, formatter);
    }

    @Override
    public String marshal(LocalDate v) {
        return v.format(formatter);
    }
}

En la clase
@XmlRootElement
public class Evento {
    @XmlJavaTypeAdapter(LocalDateAdapter.class)
    private LocalDate fecha;
}

XML resultante
<evento>
    <fecha>2025-11-17</fecha>
</evento>
```

- Al deserializar, JAXB usa el adaptador para devolver un `LocalDate`.
- Al serializar, convierte el `LocalDate` a `String` en formato `yyyy-MM-dd`.

## Conversion de fechas que no están en formato ISO 8601,

- JAXB espera que las fechas estén en formato ISO 8601**, por ejemplo: `yyyy-MM-dd` o `yyyy-MM-dd'T'HH:mm:ss`.
- Si el XML tiene 10/03/25** (formato dd/MM/yy), **JAXB no puede convertirlo automáticamente a Date**.
- Intentar hacer `unmarshal` directamente **provocará una excepción** (`UnmarshalException` o `ParseException`) porque no reconoce el formato.

### Solución: usar un XmlAdapter

Para poder leer y escribir fechas en formato personalizado (dd/MM/yy), hay que usar un adaptador:

```

public class DateAdapter extends XmlAdapter<String, Date> {
    private final SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yy");
    @Override
    public Date unmarshal(String v) throws Exception {
        return dateFormat.parse(v); // Convierte String a Date
    }
    @Override
    public String marshal(Date v) throws Exception {
        return dateFormat.format(v); // Convierte Date a String
    }
}
Uso en la clase Java
@XmlRootElement
public class Evento {
    @XmlJavaTypeAdapter(DateAdapter.class)
    private Date fecha;
    // getters y setters
}

Comportamiento al serializar y deserializar
Serialización (marshalling): convierte el Date a String con formato dd/MM/yy.
Deserialización (unmarshalling): convierte el String del XML 10/03/25 a un objeto Date
en Java.

XML resultante:
<evento>
    <fecha>10/03/25</fecha>
</evento>

```

## 6. Trabajar con annotation

### Anotaciones asociadas a una clase.

- Se especifican delante de la clase.

Anotación	Descripción
<code>@XmlType</code>	<p>Controla el <b>orden que van a tener las propiedades en el mapeo a los elementos XML</b>. Por defecto, aparecen en el mismo orden de definición de las propiedades</p> <p>Sintaxis:</p> <pre> @XmlType(propOrder={"prop1", "prop2", "prop3"}) @XmlRootElement @XmlType(propOrder={"id", "nombre", "apellidos"}) public class Cliente {     private String nombre;     private String apellidos;     private int id; }</pre> <pre> &lt;cliente&gt;     &lt;id&gt;123&lt;/id&gt;     &lt;nombre&gt;Ana&lt;/nombre&gt;     &lt;apellidos&gt;Pérez&lt;/apellidos&gt; &lt;/cliente&gt;</pre>
<code>@XmlRootElement</code>	<p>Define el <b>elemento raíz del documento</b>.</p> <p>Sintaxis:</p> <pre> @XmlRootElement(name = valor1, namespace = valor2) name → nombre del elemento raíz. Por defecto, nombre de la clase en minúsculas namespace → espacio de nombres del documento XML. Por defecto, ninguno.</pre> <pre> @XmlRootElement(name="Alumno", namespace="http://ejemplo.com/alumnos") public class Alumno {     private int id;     private String nombre; }  &lt;Alumno xmlns="http://ejemplo.com/alumnos"&gt;     &lt;id&gt;1&lt;/id&gt;     &lt;nombre&gt;Manuel&lt;/nombre&gt; &lt;/Alumno&gt;</pre>

<b>@XmlAccessorType</b>	Configura qué tipo de elementos se serializarán en XML. También se puede utilizar a nivel de paquete <ul style="list-style-type: none"> <li>• <code>@XmlAccessType.FIELD ()</code></li> <li>• <code>@XmlAccessType.NONE ()</code></li> <li>• <code>@XmlAccessType.PROPERTY ()</code></li> <li>• <code>@XmlAccessType.PUBLIC_MEMBER ()</code></li> </ul>
<b>@XmlSeeAlso</b>	Indica que, al procesar una clase, también debe conocer y considerar otras clases relacionadas (normalmente subclases) para el mapeo XML. Sirve para garantizar que JAXBContext incluya tipos que no aparecen explícitamente referenciados en la clase raíz.

**¿Por qué y cuándo usarla?**

Tienes jerarquías de herencia y quieres mapear la clase base pero JAXB debe conocer también las subclases. Java no puede descubrir automáticamente todas las subclases de una clase base en tiempo de ejecución

No puedes (o no quieres) crear el JAXBContext con todas las clases explícitas.

```
@XmlSeeAlso({Clase1.class, Clase2.class})
```

```
@XmlSeeAlso({Dog.class, Cat.class}) // JAXB ahora conoce Dog y Cat
class Animal { /* ... */ }
class Dog extends Animal { /* ... */ }
class Cat extends Animal { /* ... */ }
```

**Anotaciones utilizadas para los campos de un objeto Java**

- Se anotan antes de la propiedades o de los métodos get().

Anotación	Descripción																		
<b>@XmlElement</b>	Indica que una propiedad del objeto Java se mapeará como un <b>elemento XML</b> Sintaxis: <pre><code>@XmlElement(     name = "nombreElemento",     required = false,     namespace = "espacioNombres",     nillable = true )</code></pre> Tiene varios parámetros opcionales para declarar restricciones sobre el elemento. <pre><code>public class Alumno {     @XmlElement(name="nombreAlumno", required=true)     private String nombre; } &lt;alumno&gt;     &lt;nombreAlumno&gt;Manuel&lt;/nombreAlumno&gt; &lt;/alumno&gt;</code></pre> <table border="1"> <thead> <tr> <th>Parámetro</th><th>Descripción</th><th>Valor por defecto</th></tr> </thead> <tbody> <tr> <td><b>Valor por defecto</b></td><td>El nombre del elemento se deriva automáticamente del nombre de la propiedad Java.</td><td>Nombre de la propiedad</td></tr> <tr> <td><b>name</b></td><td>Indica el nombre del elemento en el XML.</td><td>Nombre de la propiedad</td></tr> <tr> <td><b>required</b></td><td>Define si el elemento es obligatorio (<code>true</code>) u opcional (<code>false</code>).</td><td><code>false</code> (opcional)</td></tr> <tr> <td><b>namespace</b></td><td>Especifica el espacio de nombres del elemento en el XML.</td><td>Ninguno</td></tr> <tr> <td><b>nillable</b></td><td>Permite que el elemento se represente como <code>xsi:nil</code> si el valor es nulo.</td><td><code>false</code></td></tr> </tbody> </table>	Parámetro	Descripción	Valor por defecto	<b>Valor por defecto</b>	El nombre del elemento se deriva automáticamente del nombre de la propiedad Java.	Nombre de la propiedad	<b>name</b>	Indica el nombre del elemento en el XML.	Nombre de la propiedad	<b>required</b>	Define si el elemento es obligatorio ( <code>true</code> ) u opcional ( <code>false</code> ).	<code>false</code> (opcional)	<b>namespace</b>	Especifica el espacio de nombres del elemento en el XML.	Ninguno	<b>nillable</b>	Permite que el elemento se represente como <code>xsi:nil</code> si el valor es nulo.	<code>false</code>
Parámetro	Descripción	Valor por defecto																	
<b>Valor por defecto</b>	El nombre del elemento se deriva automáticamente del nombre de la propiedad Java.	Nombre de la propiedad																	
<b>name</b>	Indica el nombre del elemento en el XML.	Nombre de la propiedad																	
<b>required</b>	Define si el elemento es obligatorio ( <code>true</code> ) u opcional ( <code>false</code> ).	<code>false</code> (opcional)																	
<b>namespace</b>	Especifica el espacio de nombres del elemento en el XML.	Ninguno																	
<b>nillable</b>	Permite que el elemento se represente como <code>xsi:nil</code> si el valor es nulo.	<code>false</code>																	
<b>@XmlElements</b>	Actúa como un contenedor de múltiples anotaciones @XmlElement Sintaxis <pre><code>@XmlElements({     @XmlElement(name="elemento1", type=Clase1.class),     @XmlElement(name="elemento2", type=Clase2.class) }) // Clase raíz // Clase raíz que agrupa empleados</code></pre>																		

	<pre> @XmlRootElement(name="empresa") public class Empresa {     @XmlElement(name="empleado")     //cada elemento de la lista se serializa como un &lt;empleado&gt;.     private List&lt;Empleado&gt; empleados;      public class Empleado {         private String nombre;         @XmlElements({             @XmlElement(name="direccion", type=Direccion.class),             @XmlElement(name="numerotelefono", type=Telefono.class)         })         private Contacto contactoInfo;     }     // Clase base abstracta     public abstract class Contacto { }     // Subclase Direccion     class Direccion extends Contacto {         private String calle;         private String ciudad;     }     // Subclase Telefono     class Telefono extends Contacto {         private String tipo;         private String numero;  &lt;empresa&gt;     &lt;empleado&gt;         &lt;nombre&gt;Ana&lt;/nombre&gt;         &lt;direccion&gt;             &lt;calle&gt;Gran Vía&lt;/calle&gt;             &lt;ciudad&gt;Madrid&lt;/ciudad&gt;         &lt;/direccion&gt;     &lt;/empleado&gt;      &lt;empleado&gt;         &lt;nombre&gt;Juan&lt;/nombre&gt;         &lt;numerotelefono&gt;             &lt;tipo&gt;Casa&lt;/tipo&gt;             &lt;numero&gt;988787878&lt;/numero&gt;         &lt;/numerotelefono&gt;     &lt;/empleado&gt; &lt;/empresa&gt; </pre>
<b>@XmlElementWrapper</b>	<p>Agrupa elementos para organizar una colección de datos</p> <p>Sintaxis:</p> <pre> @XmlElementWrapper(name="nombreContenedor") @XmlElement(name="nombreElemento") private List&lt;Tipo&gt; lista; </pre> <pre> @XmlRootElement public class Cliente {     @XmlElementWrapper(name="emails")     @XmlElement(name="email")     private List&lt;String&gt; emails; }</pre> <pre> &lt;cliente&gt;     &lt;emails&gt;         &lt;email&gt;janed@example.com&lt;/email&gt;         &lt;email&gt;joseg@example.org&lt;/email&gt;     &lt;/emails&gt; &lt;/cliente&gt;</pre> <pre> SIN @XmlElementWrapper @XmlRootElement public class Cliente {     @XmlElement(name="email")     private List&lt;String&gt; emails; }</pre> <pre> &lt;cliente&gt;     &lt;email&gt;janed@example.com&lt;/email&gt;     &lt;email&gt;joseg@example.org&lt;/email&gt; &lt;/cliente&gt;</pre>
<b>@XmlAttribute</b>	<p>Mapea una propiedad de un objeto como <b>un atributo de un elemento</b> en el XML resultante. Sintaxis</p> <pre> @XmlAttribute(     name = "nombreAtributo", required = false,     namespace = "espacioNombres" ) </pre>

	<p>private Tipo propiedad;</p> <table border="1"> <thead> <tr> <th>Parámetro</th><th>Descripción</th><th>Valor por defecto</th></tr> </thead> <tbody> <tr> <td><code>name</code></td><td>Indica el nombre del atributo en el XML.</td><td>Nombre de la propiedad</td></tr> <tr> <td><code>required</code></td><td>Define si el atributo es obligatorio (<code>true</code>) u opcional (<code>false</code>).</td><td><code>false</code> (opcional)</td></tr> <tr> <td><code>namespace</code></td><td>Especifica el espacio de nombres del atributo en el XML.</td><td>Ninguno</td></tr> </tbody> </table> <pre>@XmlRootElement public class Telefono {     @XmlAttribute(name="tipo")     private String tipoTelefono;     @XmlElement     private String numero; &lt;Telefono tipo="Casa"&gt;     &lt;numero&gt;988787878&lt;/numero&gt; &lt;/Telefono&gt;</pre>	Parámetro	Descripción	Valor por defecto	<code>name</code>	Indica el nombre del atributo en el XML.	Nombre de la propiedad	<code>required</code>	Define si el atributo es obligatorio ( <code>true</code> ) u opcional ( <code>false</code> ).	<code>false</code> (opcional)	<code>namespace</code>	Especifica el espacio de nombres del atributo en el XML.	Ninguno
Parámetro	Descripción	Valor por defecto											
<code>name</code>	Indica el nombre del atributo en el XML.	Nombre de la propiedad											
<code>required</code>	Define si el atributo es obligatorio ( <code>true</code> ) u opcional ( <code>false</code> ).	<code>false</code> (opcional)											
<code>namespace</code>	Especifica el espacio de nombres del atributo en el XML.	Ninguno											
<code>@XmlTransient</code>	<p>El campo del objeto no se mapea a un elemento en la representación XML.  Sintaxis:  <code>@XmlTransient</code>  private Tipo propiedad;</p> <p>Se utiliza cuando una propiedad es solo interna de la clase y no debe aparecer en el XML. Muy útil para datos sensibles (ej. contraseñas) o para campos auxiliares que no forman parte del modelo XML.</p> <pre>@XmlRootElement public class Usuario {     @XmlElement     private String nombre;     @XmlTransient     private String claveSecreta; // este campo no aparecerá en el XML }  &lt;Usuario&gt;     &lt;nombre&gt;Pepa&lt;/nombre&gt; &lt;/Usuario&gt;</pre>												
<code>@XmlValue</code>	<p>Asigna el valor de un objeto directamente al contenido de un elemento XML, mientras que otras propiedades pueden mapearse como atributos.  Se utiliza cuando quieres que el texto interno del elemento represente el valor principal del objeto.  Sintaxis:  <code>@XmlValue</code>  private Tipo propiedad;</p> <pre>RootElement(name="Telefono") public class Telefono {     // atributo XML     @XmlAttribute(name="tipo")     private String tipo;     // valor principal del elemento XML     @XmlValue     private String numero;  &lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;Telefono tipo="Casa"&gt;988787878&lt;/Telefono&gt;</pre> <p>-Sin <code>@XMLValue</code></p> <pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;Telefono tipo="Casa"&gt;     &lt;numero&gt;988787878&lt;/numero&gt; &lt;/Telefono&gt;</pre>												
<code>@XmlID</code>	Marca un atributo como identificador único dentro del documento XML. Es similar a una clave primaria: cada objeto anotado con <code>@XmlID</code> debe tener un valor único												
<code>@XmlIDREF</code>	<code>@XmlAttribute</code> <code>@XmlID</code>												

	<pre> private String id; &lt;empleado id="1" nombre="Carlos"/&gt;  <b>@XmlIDREF:</b> Permite que un campo haga referencia a otro elemento identificado con @XmlID. Es decir, en lugar de duplicar el objeto, se guarda el ID del objeto referenciado.  @XmlAccessorType(XmlAccessType.FIELD) public class Empleado {     @XmlAttribute     @XmlID     private String id;     @XmlAttribute     private String nombre;     @XmlIDREF     private Empleado director;     public Employee()         { lista = new ArrayList&lt;Empleado&gt;();      }      }  &lt;compañia&gt;     &lt;empleado id="1" nombre="Carlos"/&gt;     &lt;empleado id="2" nombre="John Smith"&gt;         &lt;director&gt;1&lt;/director&gt;     &lt;/empleado&gt;     &lt;empleado id="3" nombre="Anne Jones"&gt;         &lt;director&gt;1&lt;/director&gt;     &lt;/empleado&gt; &lt;/compañia&gt; </pre>
<b>@XmlList</b>	<p>Especifica cómo se mapea un atributo de tipo List o Collection. En lugar de generar múltiples subelementos, <b>une todos los valores en un único campo de texto</b>, separados por espacios.</p> <p>Sintaxis:</p> <pre> @XmlList private List&lt;Tipo&gt; lista; </pre> <pre> @XmlElement(name="cliente") @XmlAccessorType(XmlAccessType.FIELD) public class Cliente {     // La lista se mapea como un único bloque de texto     @XmlList     private List&lt;String&gt; emails; } &lt;cliente&gt;     &lt;emails&gt;janed@example.com joseg@example.org&lt;/emails&gt; &lt;/cliente&gt; </pre>

### Anotaciones asociadas Java Package.

Para especificar una anotación de nivel de paquete, se debe crear una clase especial llamada **pachage-info.java** y especificar aquí el **@XMLXXX**.

Anotación	Descripción
<b>@XmlSchema</b>	<p>Sirve para configurar un espacio de nombres (namespace) y <b>cómo se mapean los elementos y atributos de todas las clases</b> de un paquete.</p> <p>Sintaxis</p> <pre> @XmlSchema (     xmlns = {},     namespace = "",     elementFormDefault = XmlNsForm.UNSET,     attributeFormDefault = XmlNsForm.UNSET ) </pre> <ul style="list-style-type: none"> <li>▪ xmlns = {} → Lista de prefijos y namespaces adicionales. Por defecto está vacío. Ejemplo: podrías definir xmlns = { @XmlNs(prefix="xsi", namespaceURI="http://www.w3.org/2001/XMLSchema-instance") }.</li> <li>▪ namespace = "" → Namespace por defecto para el paquete. Si está vacío (""), significa que no se aplica ningún namespace. Ejemplo: namespace =</li> </ul>

	<p>"http://www.example.org/mipaquete".</p> <ul style="list-style-type: none"> <li>elementFormDefault = XmlNsForm.UNSET → Indica si los elementos deben estar calificados con el namespace.           <ul style="list-style-type: none"> <li>UNSET → sin especificar (depende del contexto).</li> <li>QUALIFIED → todos los elementos llevan el namespace.</li> <li>UNQUALIFIED → los elementos no llevan namespace.</li> </ul> </li> <li>attributeFormDefault = XmlNsForm.UNSET → Igual que lo anterior, pero para atributos.</li> <li>QUALIFIED → atributos con namespace.</li> <li>UNQUALIFIED → atributos sin namespace.</li> </ul> <pre> @jakarta.xml.bind.annotation.XmlSchema(     namespace = "http://www.example.org/mipaquete",     elementFormDefault = jakarta.xml.bind.annotation.XmlNsForm.QUALIFIED,     attributeFormDefault = jakarta.xml.bind.annotation.XmlNsForm.UNQUALIFIED ) package ejemplo; &lt;cliente xmlns="http://www.example.org/mipaquete" id="123"&gt;     &lt;nombre&gt;Pepa&lt;/nombre&gt; &lt;/cliente&gt;</pre>
<b>@XmlAccessorType</b>	<p>Controla el orden que van a tener los atributos en el mapeo a los elementos XML a nivel de paquete, es decir, está disponible para todas las clases.</p> <p>Sintaxis:</p> <pre>@XmlAccessorType (     value = AccessorOrder.UNDEFINED AccessorOrder.ALPHABETICAL )</pre> <ul style="list-style-type: none"> <li>AccessorOrder.UNDEFINED → el orden de los atributos/elementos no está definido, JAXB los genera según la implementación.</li> <li>AccessorOrder.ALPHABETICAL → los atributos/elementos se ordenan alfabéticamente por nombre.</li> </ul>

### Ejemplo:ListaPuntos.java

- XmlElementWrapper(name="Lista") → agrupa la lista de puntos dentro de una etiqueta <Lista>.
- XmlElement(name="Punto") → cada objeto de la lista se serializa como <Punto>.

```

@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "listaPuntos")
public class ListaPuntos {
    @XmlElementWrapper(name="Lista")
    @XmlElement(name = "Punto")
    private ArrayList <Puntos> lista;
    public ListaPuntos() {
        lista=new ArrayList <>();
        public ArrayList<Puntos> getLista() {      return lista;      }
        public void setLista(ArrayList<Puntos> lista) {      this.lista = lista;      }
    }
}
```

### Puntos.java

- XmlAccessorType(XmlAccessType.NONE) → obliga a que solo los métodos anotados se incluyan en el XML.
- XmlType(propOrder={"y","x"}) → define el orden de serialización: primero y, luego x.
- Los getters están anotados con @XmlElement(name="EjeX") y @XmlElement(name="EjeY").

```

package ejJAXBserialobjetosaxml;
import javax.xml.bind.annotation.*;
@XmlAccessorType(XmlAccessType.NONE)
@XmlType(propOrder = {"y","x"})
public class Puntos {
    private int x;
    private int y;
    //Muy importante: LA CLASE DEBE TENER UN CONSTRUCTOR POR DEFECTO, SINO DA ERROR
    public Puntos(){}
    public Puntos(int x,int y){
        this.x=x;
        this.y=y;
        @XmlElement(name="EjeX")
        public int getX() { return x; }
        public void setX(int x) { this.x = x; }
        @XmlElement(name="EjeY")
        public int getY() { return y; }
        public void setY(int y) { this.y = y; }
    }
}

```

En el package-info.java

- Define un namespace por defecto con `@XmlSchema`.
- `elementFormDefault = QUALIFIED` → todos los elementos se califican con el namespace.
- Se añade un prefijo com para el namespace `http://es.indra.transporte.common`.

```

@javax.xml.bind.annotation.XmlSchema (
    xmlns = { @javax.xml.bind.annotation.XmlNs(prefix = "com",
                                                 namespaceURI="http://es.indra.transporte.common"),
               },
    elementFormDefault = javax.xml.bind.annotation.XmlNsForm.QUALIFIED
)
package ejJAXBserialobjetosaxml;

```

### Configuración del Marshaller

```

public class Main {
    public static void main(String[] args) throws Exception {
        // Crear objeto de prueba
        ListaPuntos lista = new ListaPuntos();
        lista.getLista().add(new Puntos(5,8));
        lista.getLista().add(new Puntos(1,9));

        // Crear contexto JAXB
        JAXBContext context = JAXBContext.newInstance(ListaPuntos.class);

        // Crear marshaller
        Marshaller marshaller = context.createMarshaller();

        // Propiedades comunes
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
        // Con esquema XSD
        marshaller.setProperty(Marshaller.JAXB_SCHEMA_LOCATION, "XMLPuntos.xsd");

        marshaller.marshal(lista, new File("listaPuntos.xml"));
    }
}

```

Resultado:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<listaPuntos xmlns:com="http://es.indra.transporte.common"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="XMLPuntos.xsd">
    <Lista>
        <Punto>
            <EjeY>8</EjeY>
            <EjeX>5</EjeX>
        </Punto>
        <Punto>
            <EjeY>9</EjeY>
            <EjeX>1</EjeX>
        </Punto>
        <Punto>
            <EjeY>7</EjeY>
            <EjeX>3</EjeX>
        </Punto>
        <Punto>
            <EjeY>5</EjeY>
            <EjeX>9</EjeX>
        </Punto>
    </Lista>
</listaPuntos>
```

## 7. JAXBContext, Marshaller y Unmarshaller

### JAXBContext,

Esta clase gestiona el mapeo mapeo entre objetos Java y documentos XML. Proporciona las clases **Marshaller** y **Unmarshaller**

- La clase **Unmarshaller** transforma un documento XML a objetos Java (opcionalmente puede validar el documento mediante su esquema XSD).
- La clase **Marshaller** transforma objetos Java en un documento XML.

#### Paquetes a importar

- **JAXB 2.x (hasta JDK 8, Java EE)**

Se usaban los paquetes **javax**:

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
```

- **JAXB 3.x / 4.x (Jakarta EE, JDK 11 en adelante)**

Ahora se usan los paquetes **jakarta**:

```
import jakarta.xml.bind.JAXBContext;
import jakarta.xml.bind.JAXBException;
import jakarta.xml.bind.Marshaller;
import jakarta.xml.bind.Unmarshaller;
```

#### Instanciación de JAXBContext

Para trabajar con JAXB necesitas crear un **contexto** que conozca la clase raíz anotada con `@XmlRootElement`.

- **Instanciar un objeto JAXBContext** y especificar la clase que contiene el elemento raíz :

Se llama al método newInstance():

```
newInstance
```

```
public static JAXBContext newInstance(Class... classesToBeBound)
                                      throws JAXBException
```

Puede lanzar la excepción JAXBException si no se puede instanciar.

Ej:

```
JAXBContext contexto = null;
try {
// Se crea el contexto indicando la clase raíz (ListaPuntos)
    contexto = JAXBContext.newInstance(ListaPuntos.class);
} catch (JAXBException ex) {
    System.out.println("Error al instanciar el objeto JAXBContext ");
}
```

## 7.1 Convetir objetos Java en Documento XML

### 1.-Creación de un objeto Marshaller.

Este objeto controla el **proceso de mapeo de objetos java a XML**.

Para la creación del objeto Marshaller se llama al método **createMarshaller()** de la clase estática JAXBContext.

```
createMarshaller
```

```
public abstract Marshaller createMarshaller()
                                              throws JAXBException
```

Lanza la excepción **JAXBException** si ocurre un error en el proceso de creación del objeto Marshaller.

Ejemplo:

```
JAXBContext contexto = null;
Marshaller m = null;
try {
    contexto = JAXBContext.newInstance(ListaPuntos.class);
    m = contexto.createMarshaller();
} catch (JAXBException ex) {
    System.out.println("error al crear el objeto Marshaller");
}
```

### 2.-Establecer las propiedades de objeto Marshaller

El objeto Marshaller controla cómo se convierte un objeto Java en un documento XML. Se pueden ajustar sus propiedades con el método **setProperty()**:

```
setProperty
```

```
void setProperty(String name,
                  Object value)
                     throws PropertyException
```

Algunas propiedades son:

Nombre Propiedad (String name)	Tipo propiedad (Object value)	Descripción
<b>jaxb.formatted.output</b>	Boolean	Esta propiedad controla si el Marshaller dará <b>formato a los datos XML resultante con saltos de línea y sangría</b> . Por defecto es falso (sin formato). <code>m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);</code> <code>&lt;alumno&gt;</code> <code>    &lt;nombre&gt;Manuel&lt;/nombre&gt;</code> <code>&lt;/alumno&gt;</code> <code>(en lugar de &lt;alumno&gt;&lt;nombre&gt;Manuel&lt;/nombre&gt;&lt;/alumno&gt; todo seguido).</code>
<b>jaxb.encoding</b>	String	La codificación de salida del documento XML en el mapeo de los objetos Java. Por defecto es "UTF-8".
<b>jaxb.schemaLocation</b>	String	Añade el atributo <b>xsi:schemaLocation</b> al XML, indicando la <b>ubicación del esquema XSD</b> que valida el documento. (El documento xml tiene su propio espacio de nombre definido en el esquema xsd) <code>m.setProperty(Marshaller.JAXB_SCHEMA_LOCATION,</code> <code>"http://ejemplo.com alumno.xsd");</code>  <code>&lt;alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code> <code>    xsi:schemaLocation="http://ejemplo.com alumno.xsd"&gt;</code> <code>    &lt;nombre&gt;Manuel&lt;/nombre&gt;</code> <code>&lt;/alumno&gt;</code>
<b>jaxb.noNamespaceSchemaLocation</b>	String	Añade el <b>atributo xsi:noNamespaceSchemaLocation</b> al XML, indicando el esquema XSD cuando el documento <b>no tiene espacio de nombres</b> . (El documento xml no tiene su propio espacio de nombre definido en el esquema xsd) <code>m.setProperty(Marshaller.JAXB_NO_NAMESPACE_SCHEMA_LOCATION,</code> <code>"alumno.xsd");</code> <code>&lt;alumno xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code> <code>    xsi:noNamespaceSchemaLocation="alumno.xsd"&gt;</code> <code>    &lt;nombre&gt;Manuel&lt;/nombre&gt;</code> <code>&lt;/alumno&gt;</code>

Ejemplo:

```
try {
    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    m.setProperty(Marshaller.JAXB_NO_NAMESPACE_SCHEMA_LOCATION,"listapuntos.xsd");
} catch (PropertyException ex) {
    System.out.println("error al establecer la propiedad");
}
```

Ejemplo de salida:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<listaPuntos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="listapuntos.xsd">
    <lista>
        .....
```

### 3.-Realizar el mapeo de objetos Java a XML

El método **marshal(Object jaxbObject, destino)** convierte un objeto Java en un documento XML. Admite los siguientes parámetros:

Modifier and Type	Method and Description
<b>static void</b>	<b>marshal (Object jaxbObject, File xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, OutputStream xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, Result xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, String xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, URI xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, URL xml)</b>
<b>static void</b>	<b>marshal (Object jaxbObject, Writer xml)</b>

Ej:

```
//introducimos puntos a la lista
ArrayList <Puntos> listapuntos = new ArrayList();
listapuntos.add(new Puntos(5, 8));
listapuntos.add(new Puntos(1, 9));
listapuntos.add(new Puntos(3, 7));
listapuntos.add(new Puntos(9, 5));

ListaPuntos miLista=new ListaPuntos();
miLista.setLista(listapuntos);

//Salida del XML por pantalla
m.marshal(miLista, System.out);

//mapear a un fichero
Writer w = new FileWriter("XMLPuntos.xml");
m.marshal(miLista, w);
```

Resultado:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<listaPuntos>      //nombre de la clase
    <lista>          //nombre del atributo (arraylist)
        <x>5</x>
        <y>8</y>
    </lista>
    <lista>
        <x>1</x>
        <y>9</y>
    </lista>
    <lista>
        <x>3</x>
        <y>7</y>
    </lista>
    <lista>
        <x>9</x>
        <y>5</y>
    </lista>
</listaPuntos>
```

## 7.2 Mapeo de XML a objetos Java

### 1.-Creación de un objeto Unmarshaller.

Tenemos que **crear un objeto Unmarshaller**.

Este objeto controla el proceso de deserialización, es decir, de mapear un documento XML a objetos Java.

### 2. Formatos admitidos por unmarshal()

El método que transforma es unmarshal, que tiene el siguiente formato:

El origen de datos puede ser un InputStream, un String, un Source (SaxSource, DomSource, StreamSource), Reader, URI e una URL.

Método	
static <T> T	unmarshal(File xml, Class<T> type)
static <T> T	unmarshal(InputStream xml, Class<T> type)
static <T> T	unmarshal(Reader xml, Class<T> type)
static <T> T	unmarshal(Source xml, Class<T> type)
static <T> T	unmarshal(String xml, Class<T> type)
static <T> T	unmarshal(URI xml, Class<T> type)
static <T> T	unmarshal(URL xml, Class<T> type)

## Ejemplo:

```

    ListaPuntos miLista;           // Objeto Java donde se cargará el XML
    JAXBContext contexto = null;   //Contexto JAXB
    Unmarshaller m = null;         // Objeto Unmarshaller
    try {
        // Crear el contexto indicando la clase raíz anotada con @XmlRootElement
        contexto = JAXBContext.newInstance(ListaPuntos.class);
        m = contexto.createUnmarshaller();
    } catch (JAXBException ex) {
        System.out.println("error al crear el objeto Marshaller");
    }

    File f = new File("XMLPuntos.xml");

    // Transformar el XML en un objeto Java
    miLista = (ListaPuntos) m.unmarshal(f);
    //recorremos el array de puntos
    for (Puntos i:miLista.getListado())
        System.out.println(i.getX()+" , "+i.getY());
    }
}

```