# XML

JAXB: Correspondencia entre Objetos y XML

Aplicaciones Distribuidas

Curso 2025/26

DIS
Departamento de
Informática y Sistemas

UNIVERSIDAD DE
MURCIA

# Introducción JAXB

- **JAXB** (*Java Architecture for XML Binding*)
- **Objetivo**:
  - Evitar manejar un árbol genérico de objetos (DOM).
  - Establecer la correspondencia entre documentos XML y tipos de datos Java.
- Resultado:
  - Los documentos XML pueden ser procesados como objetos de dominio en Java (ej., Calificacion, Diligencia, ...)
- ¿Qué ofrece?
  - **Anotaciones** para establecer el mapeo entre los objetos y los documentos XML

# Configuración en Maven

- En el fichero `pom.xml` de **Maven**:

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>

<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.1</version>
</dependency>
```

# Componentes de la arquitectura

- **API JAXB** (paquete `javax.xml.bind`):
  - Ofrece la funcionalidad para la conversión entre documentos XML y objetos Java.

- **Terminología** utilizada en JAXB:
  - **Árbol de contenido**: objetos de dominio Java que corresponden con un documento XML.
  - **Desempaquetado** (*unmarshal*): proceso de conversión de un documento XML en un árbol de contenido.
  - **Empaquetado** (*marshal*): proceso de conversión de un árbol de contenido en un documento XML.

# Contexto JAXB

- La clase **`JAXBContext`** representa el contexto de la tarea de conversión entre XML y Java.
- Se configura con la clase que representa el **elemento raíz** del documento.

```
// 1. Construir el contexto JAXB

JAXBContext contexto = JAXBContext.newInstance(Acta.class);
```

# Almacenar en un documento XML

- **Empaquetado** (*marshal*): proceso de conversión de objetos de dominio Java (árbol de contenido) en un documento XML.
- La clase **`Marshaller`** ofrece la funcionalidad para realizar esta tarea:
  - El contexto JAXB actúa como factoría de estos objetos.
- El método `marshal` realiza la conversión

```
// Empaquetado en un documento XML (marshalling)

Marshaller marshaller = contexto.createMarshaller();


marshaller.marshal(acta, new File(salida));
```

# Almacenar en un documento XML

- Sobre el proceso de almacenamiento:
  - Se obtiene un documento XML no formateado: todo el contenido en una sola línea.
- Podemos **configurar el proceso** para corregir los problemas anteriores a través del objeto *marshaller* :

```
marshaller.setProperty("jaxb.formatted.output", true);
```

- Otras propiedades configurables: codificación documento.

# Cargar un documento XML

- **Desempaquetado** (*unmarshal*): proceso de conversión de un documento XML en objetos de dominio Java (árbol de contenido).
- La clase `Unmarshaller` ofrece la funcionalidad para realizar esta tarea:
  - El contexto JAXB actúa como factoría de estos objetos.
- El método `unmarshal` realiza la conversión

```
// Obtener el árbol de contenido de un documento XML

Unmarshaller unmarshaller = contexto.createUnmarshaller();

Acta acta = (Acta) unmarshaller.unmarshal(new File(documento));
```

# Modelado con JAXB

- Utilizamos clases Java (POJO) para implementar el modelo del documento XML.
- Una de las clases representa el **elemento raíz**.
- Esta clase debe tener la anotación `@XmlRootElement`
- En la declaración de las propiedades, los **tipos de datos permitidos** son:
  - Tipos primitivos y clases envoltorio (Integer, Double, etc.)
  - Clases POJO.
  - Clases que tengan correspondencia con JAXB (ej. Date, pero no LocalDate).
  - Colecciones de lo anterior.

# JAXB – Anotaciones

- Ejemplo de clase raíz del documento:

```
@XmlRootElement
public class Persona {

   // Por defecto, son elementos
   private String nombre;
   private String apellidos;

   public String getNombre() { return nombre; }
   public void setNombre(String nombre) { this.nombre = nombre; }
   public String getApellidos() { return apellidos; }
   public void setApellidos(String apellidos) {
      this.apellidos = apellidos; }
}
```

# JAXB – Anotaciones

- Por defecto, JAXB utiliza la convención *get/set* para descubrir las **propiedades** (elementos).
- Si queremos que las identifique a partir de los atributos se utiliza `@XmlAccessorType(XmlAccessType.FIELD)`
- Ejemplo:

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Persona {

    private String nombre;
    private String apellidos;

    // ...
}
```

# JAXB – Anotaciones

- Si la clase tuviera relaciones de **clientela** con otras clases, éstas solo necesitan la anotación `@XmlAccessorType` si no declaran todas sus propiedades con *get/set*.
- Ejemplo:

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Persona {

    private String nombre;
    private String apellidos;
    private Direccion direccion;
    // ...
}

public class Direccion {

    private String calle;
   // ...
}
```

# JAXB – Anotaciones

- Con la anotación `@XmlType` podemos definir el orden en el que los campos se escriben en el fichero XML.
- `@XmlElement`: permite definir el nombre del elemento XML.
- `@XmlAttribute`: define que el campo se mapea a un atributo en lugar de a un elemento en el XML.
- `@XmlTransient`: para anotar campos que no se quieren incluir en el XML.

```
@XmlRootElement
@XmlType(propOrder={"apellidos","nombre","direccion"})
public class Persona {
   private String nombre;
   private String apellidos;
   private Direccion direccion;
   // ...
 @XmlElement(name="familyName")
 public void setNombre(String nombre) { this.nombre = nombre; }
}
```

# JAXB – Ejemplo

- Generación de un documento XML a partir de una clase Java anotada:

```
// Construir el contexto JAXB para las clases anotadas

JAXBContext contexto =
                    JAXBContext.newInstance(Persona.class);

Persona persona = new Persona();
persona.setNombre("Juan");
persona.setApellidos("González");

// Empaquetado en un documento XML (marshalling)

Marshaller marshaller = contexto.createMarshaller();

marshaller.setProperty("jaxb.formatted.output", true);

marshaller.marshal(persona, new File(salida));
```

# JAXB – Ejemplo

- Documento XML obtenido:

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

<persona>
    <nombre>Juan</nombre>
    <apellidos>González</apellidos>
</persona>
```

# JAXB – Adaptadores

- Cuando se utilizan tipos de datos de una librería que no tienen una representación directa en JAXB es necesario hacer uso de un adaptador.
- Un adaptador se encarga del proceso de serialización en ambos sentidos de un dato a XML.
- Ejemplo: el tipo de datos `java.time.LocalDate`

```
public class LocalDateAdapter extends XmlAdapter<String, LocalDate>
{
    public LocalDate unmarshal(String v) throws Exception {
        return LocalDate.parse(v);
    }

    public String marshal(LocalDate v) throws Exception {
        return v.toString();
    }
}
```

# JAXB – Adaptadores

- En el nombre de la propiedad, ya sea en el atributo o método de consulta, establecemos el uso del adaptador:

```
@XmlJavaTypeAdapter(value = LocalDateAdapter.class)
public LocalDate getFecha() {
    return fecha;
}
```

- Ejemplo:
  - El atributo `fecha` es de tipo `LocalDate`.
  - El adaptador se encarga de convertir el objeto en cadena y viceversa.

# Referencias

- Introducción (Oracle):
  - http://docs.oracle.com/javase/tutorial/jaxb/intro/

- Tutorial y documentación técnica:
  - https://github.com/eclipse-ee4j/jaxb-ri