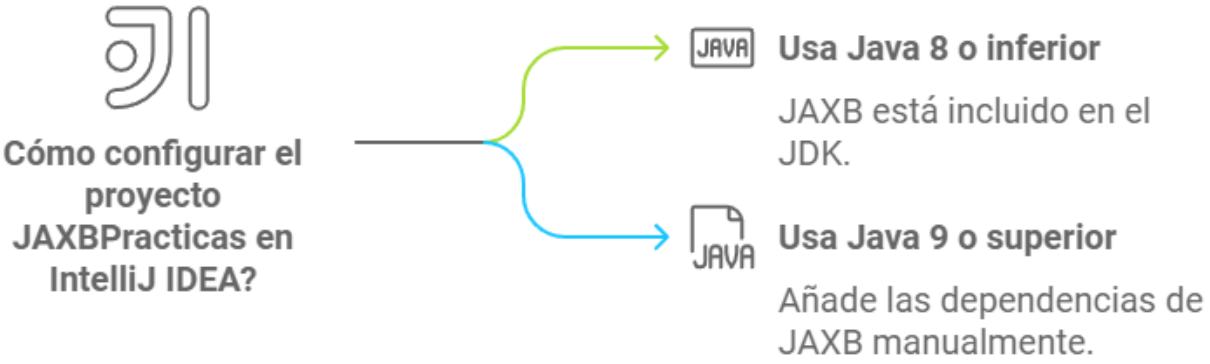
Tarea: Investigación y práctica con JAXB

Objetivo: Profundizar en el uso de JAXB (Java Architecture for XML Binding) para serialización y deserialización de objetos Java a XML y viceversa, utilizando IntelliJ IDEA.



Parte 1: Configuración del proyecto

1. Abre IntelliJ IDEA y crea un nuevo proyecto Java llamado "JAXBPracticas".
2. Configura el JDK (asegúrate de usar Java 8 o superior).
3. Añade las dependencias necesarias para JAXB (si estás usando Java 9 o superior).

Parte 2: Investigación de mejores prácticas

Investiga y resume las mejores prácticas al utilizar JAXB, incluyendo:

- Uso adecuado de anotaciones (@XmlRootElement, @XmlElement, etc.)
- Manejo de colecciones y tipos complejos
- Estrategias para manejar herencia en clases serializadas

Parte 3: Ejemplos prácticos

Desarrolla los siguientes ejemplos en tu proyecto de IntelliJ:

1. Crea una clase simple "Estudiante" con anotaciones JAXB:

```
@XmlRootElement
public class Estudiante {
    @XmlElement
    private String nombre;
    @XmlElement
```

```
    private int edad;

    // Constructores, getters y setters
}
```

2. Implementa métodos para serializar y deserializar la clase Estudiante:

```
public class JAXBUtil {
    public static void marshal(Estudiante estudiante, File file) throws
JAXBException {
        JAXBContext context = JAXBContext.newInstance(Estudiante.class);
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(estudiante, file);
    }

    public static Estudiante unmarshal(File file) throws JAXBException {
        JAXBContext context = JAXBContext.newInstance(Estudiante.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        return (Estudiante) unmarshaller.unmarshal(file);
    }
}
```

3. Crea una clase con una estructura más compleja (por ejemplo, una clase "Curso" que contenga una lista de Estudiantes) y aplica las anotaciones JAXB correspondientes.

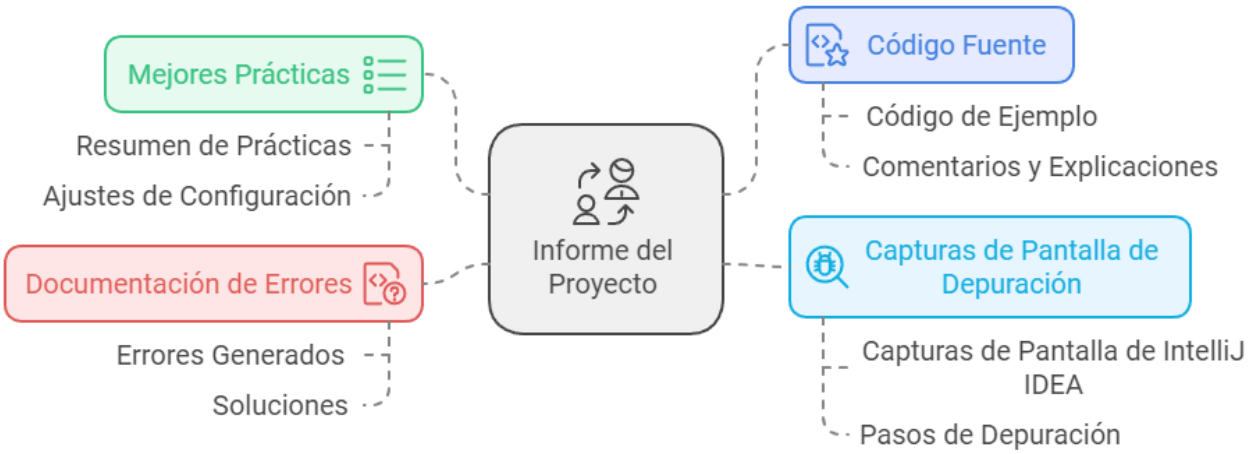
Parte 4: Depuración

1. Utiliza el depurador de IntelliJ IDEA para analizar el proceso de marshalling/unmarshalling:
    - Coloca puntos de interrupción en los métodos marshal y unmarshal.
    - Ejecuta el código en modo de depuración.
    - Examina los valores de los objetos en cada paso del proceso.
2. Genera intencionalmente algunos errores comunes:
    - Omite una anotación necesaria.
    - Utiliza tipos de datos incompatibles.
3. Para cada error, documenta:
    - El mensaje de error recibido (usa la consola de IntelliJ).

- La causa raíz del problema.
- Los pasos para solucionarlo.

Parte 5: Uso de herramientas de IntelliJ

1. Utiliza el generador de código de IntelliJ para crear getters y setters automáticamente.
2. Usa la función "Refactor" para renombrar variables y métodos de manera segura.
3. Emplea el formateador de código de IntelliJ para mantener un estilo consistente.



Entrega

Prepara un informe que incluya:

- Resumen de las mejores prácticas y configuraciones investigadas.
- Código fuente de los ejemplos desarrollados, con comentarios explicativos.
- Capturas de pantalla del proceso de depuración en IntelliJ IDEA.
- Documentación de los errores generados y sus soluciones.

Serialización de Objetos

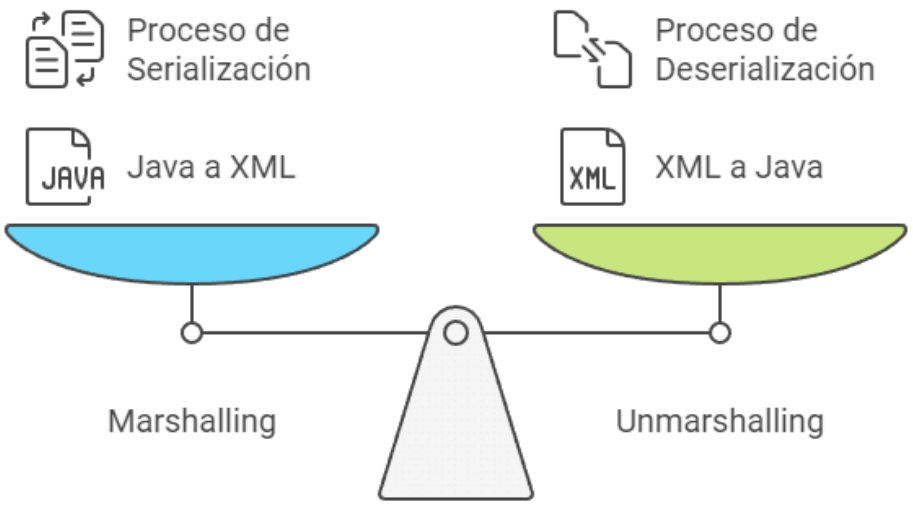
Serializar es el proceso por el cual un objeto en memoria pasa a transformarse en una estructura que pueda ser almacenada en un fichero (persistencia). Al proceso contrario le llamaremos *deserializar*.

Hay que tener en cuenta que durante el proceso de serialización, cada objeto se serializa a un fichero, por lo que si queremos almacenar todos los objetos de una aplicación, la idea más conveniente es tener todos éstos en alguna estructura compleja (y por comodidad dinámica) de forma que sea esta estructura la que serialicemos o deserialicemos para guardar o cargar los objetos de una aplicación.

JAXB (Java Architecture for XML Binding) es una tecnología que permite mapear clases Java a representaciones XML y viceversa.

Conceptos fundamentales de JAXB

1. **Marshalling**: Es el proceso de convertir objetos Java en XML.
2. **Unmarshalling**: Es el proceso inverso, convirtiendo XML en objetos Java.
3. **Anotaciones JAXB**: Son marcadores que se añaden a las clases Java para definir cómo deben ser serializadas a XML.



Comprendiendo los procesos fundamentales de JAXB.

Principales anotaciones JAXB

- `@XmlRootElement`: Define la clase como el elemento raíz del XML.
- `@XmlElement`: Especifica que un campo debe ser tratado como un elemento XML.
- `@XmlAttribute`: Indica que un campo debe ser tratado como un atributo XML.
- `@XmlTransient`: Marca un campo para que sea ignorado durante la serialización/deserialización.

Proceso básico de uso de JAXB

1. Crear las clases Java: Definir las clases que representarán la estructura del XML.
2. Anotar las clases: Usar las anotaciones JAXB para especificar cómo se mapearán los campos a XML.
3. Crear un JAXBContext: Es el punto de entrada para las operaciones JAXB.
4. Marshalling (Java a XML):
    - Crear un `Marshaller` del `JAXBContext`.
    - Usar el método `marshal()` para convertir el objeto Java a XML.
5. Unmarshalling (XML a Java):
    - Crear un `Unmarshaller` del `JAXBContext`.
    - Usar el método `unmarshal()` para convertir XML a objetos Java.

Ejemplo básico uso de JAXB

```
@XmlRootElement
public class Estudiante {
    @XmlElement
    private String nombre;
    @XmlElement
    private int edad;

    // Constructores, getters y setters
}

// Marshalling
JAXBContext context = JAXBContext.newInstance(Estudiante.class);
Marshaller marshaller = context.createMarshaller();
marshaller.marshal(estudiante, new File("estudiante.xml"));

// Unmarshalling
Unmarshaller unmarshaller = context.createUnmarshaller();
Estudiante estudianteRecuperado = (Estudiante) unmarshaller.unmarshal(new
File("estudiante.xml"));
```

Consideraciones importantes

1. Control de versiones: Usar `serialVersionUID` para manejar cambios en las clases.
2. Manejo de colecciones: JAXB puede manejar colecciones como List o Set, pero puede requerir configuración adicional.
3. Namespaces XML: JAXB permite definir y manejar namespaces XML.
4. Validación: Se puede integrar la validación de esquemas XML en el proceso de unmarshalling.
5. Personalización: Es posible personalizar el proceso de marshalling/unmarshalling para casos especiales.