Ejercicios Ficheros XML

DOM

- *Ejemplos*
  Carpeta DOM
  Partiendo de fichero *aleatorio* que habrá que crear, NombresEdades_new.dat, generar el .xml y después leerlo.

- *Planteados para resolución*
  Mostrar documento tras Validación clientes.xml
  - Con .xsd
  - Con .dtd

  Mostrar documento Personas.xml

SAX

- *Ejemplos*
  Carpeta SAX
  Leer mismo fichero .xml creado en el ejercicio anterior.

- *Planteados para resolución (mismos que DOM)*
  Mostrar documento tras Validación clientes.xml
  - Con .xsd
  - Con .dtd

  Mostrar documento Personas.xml

Conversión

- *Ejemplos*
  hojaEstilo => "librosPlantilla.xsl";
  datosLibros => "Libros.xml";
  paginaHTML => "mipagina_new.html";//página generada con el .xml usando plantilla .xsl

- *Ej. Planteados para resolución (Conversión Plantillas XSL)*
  Conversión XML usando Plantillas XSL

```
String fXML = "clientes.xml";
String fXSLHTML = "clientes_HTML.xsl";
String fXSLCSV = "clientes_CSV.xsl";
```

  Generación .html y .csv

```
String fHTML = "clientes_new.html";
String fCSV = "clientes_new.csv";
```

Serialización

- *Ejemplos*

Partiendo del fichero de objetos que habrá que crear-> *Alumnos*_new.dat

Uso XStream (***xstream-1.4.18.jar***)

*/*XStream es una interesante y simple libreria para serializar objetos a XML y viceversa.*

***Caracteristicas** de XStream*

==========

***Fácil de usar**. Cuenta una fachada de alto nivel que simplifica los casos más comunes de uso.*

***No** requiere homologaciones "**mappings**". La mayoría de los objetos se puede serilizar sin necesidad de crear archivos de homologación o "mapping".*

***Rendimiento**. Velocidad y bajo consumo de memoria son parte esencial del diseño, lo que hace que XStream sea adecuada sistemas con grandes objetos o alta demanda de envío de mensajes.*

***XML Limpio**. XStream usa "reflection" y crea archivos XML facilmente entendibles por humanos y más compactos que usando la serialización nativa de Java.*

***No requiere modificar los objetos**. Se serializan los campos internos, incluyendo privados y finales. Las clases internas y no publicas son soportadas. No se requiere que las clases tengan un constructor por defecto (sin parametros).*

***Soporte completo para objetos complejos**. Las referencias duplicadas (duplicate references) encontradas en el objeto se mantienen. Soporta referencias duplicadas (circular references).*

***Integración con otras API de XML**. Implementado un interfaz, Xstream puede serializar directamente hacia/desde cualquier estructura de arbol (tree structure) no sólo XML.*

***Estrategias de conversión personalizables**. Las estrategias de conversión se pueden registrar permitiendo personalizar como los tipos son representados en XML.*

***Mensajes de error**. Cuando una excepción, por XML mal formado, es encontrada se provee un diagnóstico detallado para ayudar a encontrar y solucionar el problema.*

***Formato se salida alterno**. El diseño modular permite otros formatos de salida. Xstream tiene actualmente soporte para JSON.*

**/*

  - generar el .xml leyendo los objetos serializables (*xstream.toXML*)
  - y después leer el xml obtenido (*xstream.fromXML*)
    - //Para que no de Errores de acceso de Tipos

      xstream.allowTypes(new Class[] {Clase1.class,Clase2.class});

- *Planteados para resolución*
  - Crear con DOM estructura clientes.xml
  - Crear y leer estructura Person (usando XStream)

JAXB

/*

Java JAXB o Java XML API Binding nos permite trabajar con XML y JSON de una forma cómoda usando Java.

Vamos a ver una introducción a este estándar y sus anotaciones. Para ello nos

apoyaremos en la clase Libro y las anotaciones de JAXB:

```
@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)
```

It defines the fields or properties of your Java classes that the JAXB engine uses for including into generated XML. It has four possible values.

- `FIELD` – Every non static, non transient field in a JAXB-bound class will be automatically bound to XML, unless annotated by `XmlTransient`.
- `NONE` – None of the fields or properties is bound to XML unless they are specifically annotated with some of the JAXB annotations.
- `PROPERTY` – Every getter/setter pair in a JAXB-bound class will be automatically bound to XML, unless annotated by `XmlTransient`.
- `PUBLIC_MEMBER` – Every public getter/setter pair and every public field will be automatically bound to XML, unless annotated by `XmlTransient`.
- Default value is `PUBLIC_MEMBER`.

```
@XmlElement(name = "employee")
```

Hemos añadido dos anotaciones @XmlRootElement que especifica la clase raíz que vamos a convertir a XML . Por otro lado @XmlElement permite cambiar el nombre de los elementos cuando el fichero XML se construya.

Marshaller- Leer

```
// Creamos el contexto indicando la clase raíz

JAXBContext contexto = JAXBContext.newInstance( libro.getClass() );


//Creamos el Marshaller, convierte el java bean en una cadena XML

Marshaller marshaller = contexto.createMarshaller();


//Formateamos el xml para que quede bien

marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
```

```
        // Lo visualizamos con system out
        marshaller.marshal(libro, System.out);

        // Escribimos en el archivo
        marshaller.marshal(libro, new File("Libro.xml"));
```

Unmarshaller - Escribir

```
        JAXBContext context = JAXBContext.newInstance( Libro.class );
        Unmarshaller unmarshaller = context.createUnmarshaller();

        Libro libro = (Libro)unmarshaller.unmarshal(new File("Libro.xml") );
        System.out.println(libro.getTitulo());
*/
```

- *Ejemplos*
  - Libro – Marshalling (Marshaller para escrbir) y Unmarshalling (Unmarshaller para leer)

- *Planteados para resolución*
  - Empleados-Empleado
  - Librería-Libro