1 Acceso a Datos: Examen Tema 1

Hay que resolver la tarea como proyecto Maven con Java.

Se trata de hacer un programa que convierta de JSON a XML y de XML a JSON.

Ejemplo de XML:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<proyecto>
    <nombre>Proyecto Sistema de Gestión</nombre>
    <descripcion>Desarrollo de un sistema de gestión de tareas</
       descripcion>
    <tareas>
        <titulo>Desarrollo de API</titulo>
        <descripcion>Desarrollo de una API RESTful</descripcion>
        <fechaInicio>2024-01-15</fechaInicio>
        <fechaFin>2024-02-15</fechaFin>
        <responsable>
            <nombre>Carlos López</nombre>
            <puesto>Desarrollador</puesto>
            <correoElectronico>carlos@ejemplo.com</correoElectronico>
            <experiencia>5</experiencia>
        </responsable>
    </tareas>
    <tareas>
        <titulo>Análisis de Requisitos</titulo>
        <descripcion>Revisión de los requisitos con el cliente</
           descripcion>
        <fechaInicio>2024-03-01</fechaInicio>
        <fechaFin>2024-04-01</fechaFin>
        <responsable>
            <nombre>Ana Martínez</nombre>
            <puesto>Analista</puesto>
            <correoElectronico>ana@ejemplo.com</correoElectronico>
            <experiencia>3</experiencia>
        </responsable>
    </tareas>
</proyecto>
```

Ejemplo de JSON:

```json
{
   "proyecto" : {
      "nombre" : "Proyecto Sistema de Gestión",
      "descripcion" : "Desarrollo de un sistema de gestión de tareas",
      "tareas" : [ {
         "titulo" : "Desarrollo de API",
         "descripcion" : "Desarrollo de una API RESTful",
         "fechaInicio" : "2024-01-15",
         "fechaFin" : "2024-02-15",
```

```
            "responsable" : {
               "nombre" : "Carlos López",
               "puesto" : "Desarrollador",
               "correoElectronico" : "carlos@nube.com",
               "experiencia" : 5
            }
         }, {
            "titulo" : "Análisis de Requisitos",
            "descripcion" : "Revisión de los requisitos con el cliente",
            "fechaInicio" : "2024-03-01",
            "fechaFin" : "2024-04-01",
            "responsable" : {
               "nombre" : "Ana Martínez",
               "puesto" : "Analista",
               "correoElectronico" : "ana@sol.es",
               "experiencia" : 3
            }
         } ]
      }
}
```

1.1 Indicaciones para empezar la tarea

Creamos el proyecto Maven:

- **paquete**: “com.iesvdc.acceso”
- **artefacto**: “examen”
- la clase principal sigue siendo la contenida en **App.java**

Corregimos la versión de Java a java 17 mínimo y le añadimos las dependencias:

1. javax activation
2. jaxb2 codehaus
3. eclipse moxy
4. jakarta json

De lo que llevamos hasta ahora hay que documentar:

- Título del examen
- Nombre y apellidos, fecha, módulo y tema en una tabla
- Qué tipo de proyecto necesitamos y cómo crearlo
- Qué dependencias hay que añadir al POM (poner el bloque de XML de cada una)

Documentamos en el archivo Readme.md que tiene que estar sí o sí en la raíz del proyecto. Si no está en esa carpeta no se valora ese criterio.

1.2 Adaptador para LocalDate

Para trabajar con fechas vamos a usar **sí o sí** `LocalDate`.

En el paquete “modelos” deberás añadir una clase **LocalDateAdapter** con el siguiente código:

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

import jakarta.xml.bind.annotation.adapters.XmlAdapter;

public class LocalDateAdapter extends XmlAdapter<String, LocalDate> {

    private static final DateTimeFormatter dateFormatter =
       DateTimeFormatter.ISO_LOCAL_DATE;

    @Override
    public LocalDate unmarshal(String date) {
        return LocalDate.parse(date, dateFormatter);
    }

    @Override
    public String marshal(LocalDate localDate) {
        return localDate.format(dateFormatter);
    }
}
```

Para que este adaptador funcione deberás añadir a tu código estas dos líneas. Suponemos que como eres un buen programador ya sabes encima de qué atributo hay que ponerlo…

```
@XmlElement
@XmlJavaTypeAdapter(LocalDateAdapter.class)
```

1.3 Ejecución parametrizada

Ahora vamos a ver cómo ejecutar con parámetros Maven.

Agrega el siguiente código al pom.xml para configurar el plugin exec-maven-plugin. En este ejemplo, vamos a pasar un parámetro llamado ***message*** a la clase .

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <mainClass>com.iesvdc.acceso.App</mainClass>
```

```
                    <arguments>
                        <argument>${message}</argument>
                    </arguments>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Este parámetro será el archivo al que hay que hacer el *unmarshalling* y posterior *marshalling*, luego nuestro programa Java deberá aceptar parámetros y además:

- Si el archivo tiene extensión **.json** automágicamente el programa lo carga en RAM y luego lo convierte a **.xml**.
- Si el archivo tiene extensión **.xml** se hace la operación inversa, se carga en RAM también pero esta vez se convierte a **.json**.
- Si se pasan dos parámatros o más o ninguno, deberá mostrar un mensaje de cómo usar el programa.

Se deberá hacer gestión de excepciones.

1.4 Libre Configuración

Al comenzar el proyecto hacermos un `git init` en la carpeta donde está el proyecto. Tiene que estar en la misma carpeta que el pom.xml o no se valora.

Hay que crear un .gitignore con lo visto en clase.

Comenzamos con un `git add .` y un `git commit -m "primer commit"`.

Al finalizar el proyecto añadimos todos los cambios y hacemos un commit con el mensaje “examen terminado”.

Si no sigues exactamente estas instrucciones, no se valora esta parte.

1.5 Qué hay que entregar

Un archivo ZIP con la carpeta del proyecto. Si se entrega otra cosa que no sea un ZIP no se corrige la tarea. **No quiero enlaces a Github o gitlab**.