Librería Jackson para el manejo de ficheros

[Configuración de jackson 1](#_Toc147216942)

[Lectura y escritura de json 1](#_Toc147216943)

[lectura de json. Deserialización 2](#_Toc147216944)

[escritura de json: serialización 2](#_Toc147216945)

[realizando operaciones. 3](#_Toc147216946)

[agregar un nuevo producto 3](#_Toc147216947)

[leer (read) 3](#_Toc147216948)

[actualizar el precio de un producto existente. 3](#_Toc147216949)

[eliminar un producto existente. 4](#_Toc147216950)

[convertir datos json a xml 5](#_Toc147216951)

[la clase jsonnode. 6](#_Toc147216952)

# Configuración de Jackson

Dependencias que necesitas agregar a tu archivo de configuración de Maven (pom.xml) para utilizar Jackson y su soporte para XML:

<dependencies>

<!-- Jackson Core -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.13.0</version>

</dependency>

<!-- Jackson Databind -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.13.0</version>

</dependency>

<!-- Jackson Dataformat XML -->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

<version>2.13.0</version>

</dependency>

<!-- ... Otras dependencias de tu proyecto ... -->

</dependencies>

# Lectura y Escritura de JSON

Jackson ofrece una potente API para leer y escribir JSON. Supongamos que tenemos un archivo datos.json similar al ejemplo anterior:

{

"cliente": {

"nombre": "María",

"edad": 30,

"direccion": {

"calle": "Calle Ejemplo",

"ciudad": "Ciudad Ejemplo"

}

},

"productos": [{

"nombre": "Producto 1",

"precio": 4.99

}, {

"nombre": "Nuevo Producto",

"precio": 39.99

}, {

"nombre": "Nuevo Producto",

"precio": 39.99

}, {

"nombre": "Nuevo Producto",

"precio": 39.99

}, {

"nombre": "Otro Producto",

"precio": 39.99

}

]

}

## Lectura de JSON. Deserialización

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;

import java.io.IOException;

import modelo.Datos;

import modelo.Producto;

public class LecturaJSON\_Deserializar {

public static void main(String[] args) {

try {

ObjectMapper objectMapper = new ObjectMapper();

// Leer datos existentes del archivo JSON

Datos datos = objectMapper.readValue(new File("./ficheros/datos.json"), Datos.class);

// Accediendo a los datos convertidos

System.out.println("Nombre del cliente: " + datos.getCliente().getNombre());

System.out.println("Edad del cliente: " + datos.getCliente().getEdad());

System.out.println("Dirección:");

System.out.println("Calle: " + datos.getCliente().getDireccion().getCalle());

System.out.println("Ciudad: " + datos.getCliente().getDireccion().getCiudad());

System.out.println("Productos:");

for (Producto producto : datos.getProductos()) {

System.out.println("Nombre: " + producto.getNombre());

System.out.println("Precio: " + producto.getPrecio());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Escritura de JSON: Serialización

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.SerializationFeature;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import modelo.\*;

public class Escritura\_JSON\_Serializar {

public static void main(String[] args) {

try {

// Crear un objeto Datos y establecer valores en él

Datos nuevosDatos = new Datos();

Cliente nuevoCliente = new Cliente();

nuevoCliente.setNombre("Nuevo Cliente");

nuevoCliente.setEdad(25);

Direccion nuevaDireccion = new Direccion();

nuevaDireccion.setCalle("Nueva Calle");

nuevaDireccion.setCiudad("Nueva Ciudad");

nuevoCliente.setDireccion(nuevaDireccion);

nuevosDatos.setCliente(nuevoCliente);

List<Producto> nuevosProductos = new ArrayList<Producto>();

nuevosProductos.add(new Producto("Nuevo Producto 1", 49.99));

nuevosProductos.add(new Producto("Nuevo Producto 2", 59.99));

nuevosDatos.setProductos(nuevosProductos);

// Escribir el objeto Datos en un archivo JSON

ObjectMapper objectMapper = new ObjectMapper();

objectMapper.enable(SerializationFeature.INDENT\_OUTPUT);

objectMapper.writeValue(new File("nuevos\_datos.json"), nuevosDatos);

System.out.println("Nuevo archivo JSON creado con éxito.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

# Realizando operaciones.

Agregar un nuevo producto

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import modelo.Datos;

import modelo.Producto;

public class AñadirProducto {

public static void main(String[] args) {

try {

ObjectMapper objectMapper = new ObjectMapper();

// Leer datos existentes del archivo JSON

Datos datos = objectMapper.readValue(new File("datos.json"), Datos.class);

// Crear un nuevo producto

Producto nuevoProducto = new Producto("Producto\_2", 39.99);

// Agregar el nuevo producto a la lista de productos

List<Producto> productos = new ArrayList<>();

productos.add(nuevoProducto);

datos.setProductos(productos);

// Escribir los datos actualizados en el archivo JSON

objectMapper.writeValue(new File("datos.json"), datos);

System.out.println("Nuevo producto agregado con éxito.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Leer (Read)

Ya visto antes

## Actualizar el precio de un producto existente.

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.File;

import java.io.IOException;

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

try {

ObjectMapper objectMapper = new ObjectMapper();

// Leer datos existentes del archivo JSON

Datos datos = objectMapper.readValue(new File("datos.json"), Datos.class);

// Encontrar el producto que deseas actualizar

Producto productoActualizar = null;

for (Producto producto : datos.getProductos()) {

if (producto.getNombre().equals("Producto 1")) {

productoActualizar = producto;

break;

}

}

if (productoActualizar != null) {

// Actualizar el precio del producto

productoActualizar.setPrecio(24.99);

// Escribir los datos actualizados en el archivo JSON

objectMapper.writeValue(new File("datos.json"), datos);

System.out.println("Producto actualizado con éxito.");

} else {

System.out.println("Producto no encontrado.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Eliminar un producto existente.

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**import** modelo.Datos;  
**import** modelo.Producto;  
  
**public class** EliminarProducto {  
 **public static void** main(String[] args) {  
 **try** {  
 ObjectMapper objectMapper = **new** ObjectMapper();  
   
 *// Leer datos existentes del archivo JSON* Datos datos = objectMapper.readValue(**new** File(**"./ficheros/datos.json"**), Datos.**class**);  
   
 *// Encontrar el producto que deseas eliminar* Producto productoEliminar = **null**;  
 **for** (Producto producto : datos.getProductos()) {  
 **if** (producto.getNombre().equals(**"Producto\_2"**)) {  
 productoEliminar = producto;  
 **break**;  
 }  
 }  
   
 **if** (productoEliminar != **null**) {  
 *// Eliminar el producto de la lista de productos* List<Producto> productos = **new** ArrayList<>();  
 productos.remove(productoEliminar);  
 datos.setProductos(productos);  
   
 *// Escribir los datos actualizados en el archivo JSON* objectMapper.writeValue(**new** File(**"datos.json"**), datos);  
   
 System.*out*.println(**"Producto eliminado con éxito."**);  
 } **else** {  
 System.*out*.println(**"Producto no encontrado."**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

# Convertir datos JSON a XML

Para convertir datos JSON a XML, la librería más potente y comúnmente utilizada es Jackson. Si bien Gson es excelente para trabajar con JSON, Jackson brinda soporte sólido tanto para JSON como para XML. A continuación, te mostraré cómo realizar la conversión de JSON a XML utilizando la librería Jackson:

Agrega las dependencias necesarias para Jackson a tu proyecto. Asegúrate de agregar tanto jackson-databind como jackson-dataformat-xml.

A continuación se presenta un ejemplo de cómo realizar la conversión de JSON a XML:

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.SerializationFeature;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import java.io.File;

import java.io.IOException;

public class JsonToXmlConverter {

public static void main(String[] args) {

try {

// Leer datos JSON usando Jackson

ObjectMapper jsonMapper = new ObjectMapper();

Datos datos = jsonMapper.readValue(new File("datos.json"), Datos.class);

// Configurar el mapper XML

XmlMapper xmlMapper = new XmlMapper();

xmlMapper.enable(SerializationFeature.INDENT\_OUTPUT); // Formato legible

// Convertir datos a XML y escribir en archivo

xmlMapper.writeValue(new File("datos.xml"), datos);

System.out.println("Datos JSON convertidos a XML con éxito.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Este código utiliza ObjectMapper para leer los datos JSON y XmlMapper para convertirlos a XML. La configuración SerializationFeature.INDENT\_OUTPUT agrega sangría y formato para hacer el XML más legible.

Asegúrate de que las clases Datos, Cliente, Direccion y Producto estén disponibles y tengan las anotaciones adecuadas para el mapeo.

Recuerda que, aunque esta es una forma de convertir JSON a XML, los formatos JSON y XML son diferentes en su estructura y semántica. La conversión directa puede no ser perfecta en todos los casos, especialmente cuando los formatos difieren significativamente.

# Convertir CSV a JSON

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.SerializationFeature;

import com.fasterxml.jackson.databind.node.ArrayNode;

import com.fasterxml.jackson.databind.node.ObjectNode;

import java.io.\*;

import java.nio.charset.StandardCharsets;

public class CSVtoJson {

public static void main(String[] args) {

String csvFile = "alumnos-RETOCADO.csv"; // Reemplaza con la ruta de tu archivo CSV

String line;

String[] headers = null;

ObjectMapper objectMapper = new ObjectMapper();

objectMapper.enable(SerializationFeature.INDENT\_OUTPUT); // Habilitar la sangría del JSON

ArrayNode jsonArray = objectMapper.createArrayNode();

try (BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(csvFile), StandardCharsets.UTF\_8))) {

// Leer la primera línea para obtener los encabezados

if ((line = br.readLine()) != null) {

headers = line.split(";");

}

while ((line = br.readLine()) != null) {

String[] values = line.split(";");

ObjectNode jsonObject = objectMapper.createObjectNode();

for (int i = 0; i < headers.length; i++) {

jsonObject.put(headers[i], values[i]);

}

jsonArray.add(jsonObject);

}

} catch (IOException e) {

e.printStackTrace();

}

// Convertir el JSONArray a una cadena JSON formateada con sangría y codificada en UTF-8

try {

String jsonString = objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(jsonArray);

System.out.println(jsonString);

} catch (IOException e) {

e.printStackTrace();

}

}

}

# 

# La clase JsonNode.

Es parte de la librería Jackson y se utiliza para representar nodos individuales en una estructura de datos JSON. Proporciona varios métodos para acceder, manipular y trabajar con los datos JSON de manera programática.

Algunos de los métodos más útiles de la clase JsonNode:

* get(String fieldName): Devuelve el nodo correspondiente al nombre del campo especificado en el JSON.
* get(int index): Devuelve el nodo correspondiente al índice del array especificado en el JSON.
* asText(): Devuelve el contenido del nodo como una cadena de texto.
* asInt(): Devuelve el contenido del nodo como un valor entero.
* asDouble(): Devuelve el contenido del nodo como un valor de punto flotante (doble).
* asBoolean(): Devuelve el contenido del nodo como un valor booleano.
* isObject(): Verifica si el nodo es un objeto JSON.
* isArray(): Verifica si el nodo es un array JSON.
* fieldNames(): Devuelve un iterador que recorre los nombres de campo en un objeto JSON.
* elements(): Devuelve un iterador que recorre los elementos de un array JSON.
* size(): Devuelve el número de elementos en un array JSON o el número de campos en un objeto JSON.
* getNodeType(): Devuelve el tipo de nodo, que puede ser OBJECT, ARRAY, STRING, NUMBER, BOOLEAN, NULL, entre otros.
* path(String fieldName): Devuelve el nodo correspondiente al nombre de campo especificado, similar a get(), pero retorna un nodo vacío si el campo no se encuentra.
* with(String fieldName): Devuelve un nodo mutable que representa el nodo con el nombre de campo especificado.
* isNull(): Verifica si el nodo es un valor nulo.
* get(String fieldName, JsonNode defaultValue): Devuelve el nodo correspondiente al nombre de campo especificado, pero si no se encuentra, retorna el valor predeterminado proporcionado.

Estos son solo algunos de los métodos más comunes de la clase JsonNode. Puedes consultar la documentación de Jackson para obtener una lista completa de los métodos disponibles y su descripción detallada: [Clase JsonNode - Documentación de Jackson](https://fasterxml.github.io/jackson-databind/javadoc/2.13/com/fasterxml/jackson/databind/JsonNode.html)

https://fasterxml.github.io/jackson-databind/javadoc/2.13/com/fasterxml/jackson/databind/JsonNode.html