**Librería Gson. Json.**  (Webgrafía: Generados de ChatGPT)

Contenido

[Introducción. 1](#_Toc147217073)

[Algunos Métodos . 2](#_Toc147217074)

[EJEMPLO1. Serialización y deserealización 2](#_Toc147217075)

[EJEMPLO2. 5](#_Toc147217076)

[Realizando operaciones. 6](#_Toc147217077)

[EJEMPLO. Un **archivo JSON de tres clientes con dos productos cada uno** 10](#_Toc147217078)

[EJEMPLO Gson para trabajar con JSON en formato JSON Tree. 12](#_Toc147217079)

[Ejemplo Gson para conversion de un Map a JSON 13](#_Toc147217080)

# Introducción.

**JSON** es un formato de intercambio de datos ligero y legible por humanos. Se basa en una estructura de pares clave-valor y es ampliamente utilizado en aplicaciones web y servicios web debido a su simplicidad y compatibilidad con múltiples lenguajes de programación.

La estructura básica de un objeto JSON se compone de pares clave-valor encerrados entre llaves {}. Cada par clave-valor está separado por una coma. Las claves son cadenas de texto y los valores pueden ser cadenas, números, booleanos, objetos JSON anidados o matrices.

**Ejemplo de JSON básico:**

**{**

**"nombre": "Juan",**

**"edad": 25,**

**"ciudad": "Ejemploville"**

**}**

**Gson** es una biblioteca Java desarrollada por Google que se utiliza para la serialización y deserialización de datos en formato JSON. Permite convertir objetos Java en formato JSON y viceversa de manera sencilla. Gson es ampliamente utilizado en aplicaciones Java para interactuar con servicios web que envían o reciben datos en formato JSON.

Principales características y usos de Gson:

1. **Serialización y Deserialización**: Gson proporciona métodos para convertir objetos Java en representaciones JSON y viceversa. Esto es útil cuando deseas enviar datos desde una aplicación Java a un servicio web o cuando recibes datos JSON y deseas convertirlos en objetos Java.
2. **Facilidad de Uso**: Gson es fácil de utilizar. Puedes serializar un objeto Java en JSON con una sola línea de código y deserializar JSON en un objeto Java de manera similar. Esto simplifica la comunicación entre aplicaciones Java y servicios web que utilizan JSON como formato de intercambio de datos.
3. **Personalización**: Gson permite personalizar la serialización y deserialización de objetos Java a JSON y viceversa. Puedes utilizar anotaciones en tus clases Java para especificar cómo se deben serializar los campos y cómo se deben mapear los datos JSON a objetos Java.
4. **Soporte para Tipos Complejos**: Gson es capaz de manejar tipos de datos complejos, como listas, mapas y objetos anidados. Puedes serializar y deserializar fácilmente objetos que contienen otros objetos o estructuras de datos complejas.
5. **Rendimiento**: Gson es conocido por ser eficiente en términos de rendimiento. La biblioteca está optimizada para minimizar la sobrecarga durante la serialización y deserialización de datos.
6. **Integración**: Puedes integrar Gson fácilmente en tus proyectos Java utilizando Maven o Gradle, ya que Gson se encuentra disponible en los repositorios de Maven Central.

# Algunos Métodos .

Los métodos más comunes proporcionados por la biblioteca Gson para la serialización y deserialización de datos JSON en Java:

1. **Serialización a JSON**:
   * **toJson(Object obj)**: Convierte un objeto Java en una cadena JSON.
   * **toJson(Object obj, Writer writer)**: Serializa un objeto Java en formato JSON y lo escribe en un flujo de caracteres proporcionado (por ejemplo, un archivo o un StringWriter).
2. **Deserialización desde JSON**:
   * **fromJson(String json, Class<T> classOfT)**: Convierte una cadena JSON en un objeto Java del tipo especificado.
   * **fromJson(JsonElement json, Class<T> classOfT)**: Deserializa un elemento JSON en un objeto Java del tipo especificado.
3. **Personalización de la Serialización/Deserialización**:
   * Puedes usar anotaciones como **@SerializedName** para personalizar los nombres de los campos JSON y las clases adaptadoras para realizar conversiones personalizadas.
4. **Otros:**
   * **Trabajo con JSON en Formato JSON Tree**:
     + **JsonParser.parseString(String json)**: Analiza una cadena JSON y crea un objeto **JsonElement** que representa la estructura JSON. Luego, puedes navegar y manipular la estructura JSON utilizando métodos de **JsonElement**.
   * **Conversión entre Mapas y JSON**:
     + **toJsonTree(Map<?, ?> src)**: Convierte un mapa en un objeto **JsonElement** que representa un objeto JSON.
     + **fromJson(JsonElement json, Type typeOfT)**: Deserializa un elemento JSON en un tipo de datos complejo, como una lista o un mapa, utilizando el tipo **Type**.
   * **Configuración Avanzada**:
     + **GsonBuilder**: Te permite configurar Gson con opciones avanzadas, como el formato de fecha/hora, el manejo de valores nulos, la inclusión/exclusión de campos, entre otros.
   * **Streaming de JSON**:
     + **JsonWriter** y **JsonReader**: Gson proporciona clases para trabajar con JSON en modo de streaming, lo que es útil cuando necesitas procesar JSON de manera eficiente sin cargar todo el documento en la memoria.
5. **Conversiones Especiales**:
   * Gson admite la conversión de tipos especiales, como Enums y números en punto flotante de precisión simple (float).
6. **Manejo de Excepciones**:
   * Gson arroja excepciones como **JsonSyntaxException** y **JsonIOException** para manejar errores durante la serialización y deserialización.

Estos son algunos de los métodos y capacidades más comunes de la biblioteca Gson en Java. La elección de los métodos dependerá de tus necesidades específicas de serialización y deserialización de datos JSON en tu aplicación.

# EJEMPLO1. Serialización y deserealización

Dependencias Maven que puedes agregar a tu archivo pom.xml:

<dependencies>

<!-- Dependencia de Gson -->

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.8</version> <!-- La versión puede variar -->

</dependency>

</dependencies>

**Clase Persona**

import com.google.gson.annotations.SerializedName;

public class Persona {

@SerializedName("nombrePersona")

private String nombre;

@SerializedName("edad")

private int edad;

public Persona(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

}

}

**Serialización a JSON**

import com.google.gson.Gson;

public class SerializacionEjemplo {

public static void main(String[] args) {

// Crear un objeto Persona

Persona persona = new Persona("Juan", 30);

// Crear una instancia de Gson

Gson gson = new Gson();

// Serializar el objeto Persona a JSON

**String json = gson.toJson(persona);**

// Imprimir el JSON resultante

System.out.println(json);

}

}

En este ejemplo, hemos creado una instancia de la clase Gson y luego utilizamos el método toJson para serializar un objeto Persona en una cadena JSON. El resultado será algo como:

{"nombre":"Juan","edad":30}

**Deserialización desde JSON**

import com.google.gson.Gson;

public class DeserializacionEjemplo {

public static void main(String[] args) {

// JSON de ejemplo

String json = "{\"nombre\":\"Maria\",\"edad\":25}";

// Crear una instancia de Gson

Gson gson = new Gson();

// Deserializar el JSON a un objeto Persona

**Persona persona = gson.fromJson(json, Persona.class);**

// Acceder a los datos deserializados

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

System.out.println("Edad: " + persona.getEdad());

}

}

En este caso, hemos utilizado el método fromJson para deserializar una cadena JSON en un objeto Persona. Luego, accedemos a los datos del objeto deserializado.

# 

# EJEMPLO2.

**Archivo JSON**

Supongamos que tenemos un archivo datos.json con la siguiente estructura:

{

"cliente": {

"nombre": "María",

"edad": 30,

"dirección": {

"calle": "Calle Ejemplo",

"ciudad": "Ciudad Ejemplo"

}

},

"productos": [

{

"nombre": "Producto 1",

"precio": 19.99

},

{

"nombre": "Producto 2",

"precio": 29.99

}

]

}

**Clases Datos.**

Para que el proceso de conversión funcione, necesitamos crear clases en Java que reflejen la estructura del JSON.

public class Datos {

private Cliente cliente;

private Producto[] productos;

// Getters y setters

}

public class Cliente {

private String nombre;

private int edad;

private Direccion direccion;

// Getters y setters

}

public class Direccion {

private String calle;

private String ciudad;

// Getters y setters

}

public class Producto {

private String nombre;

private double precio;

// Getters y setters }

**Lectura JSON. Deserialización desde JSON**

Ahora, utilizaremos Gson para leer este archivo JSON y convertirlo en objetos Java.

import com.google.gson.Gson;

import java.io.FileReader;

import java.io.IOException;

public class Main {

public static void main(String[] args) {

try (FileReader reader = new FileReader("datos.json")) {

Gson gson = new Gson();

Datos datos = gson.fromJson(reader, 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 JSON. Serialización.**

import com.google.gson.Gson; import java.io.\*; import java.util.\*;

public class Main {

public static void main(String[] args) {

try {

Gson gson = new Gson();

// Leer datos existentes del archivo JSON

Datos datos = gson.fromJson(new FileReader("datos.json"), Datos.class);

// Crear un nuevo producto

Producto nuevoProducto = new Producto("Nuevo Producto", 39.99);

List<Producto> productos = new ArrayList<>(Arrays.asList(datos.getProductos()));

productos.add(nuevoProducto);

datos.setProductos(productos.toArray(new Producto[0]));

// Escribir los datos actualizados en el archivo JSON

try (FileWriter writer = new FileWriter("datos.json")) {

gson.toJson(datos, writer);

}

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

} catch (IOException e) {

e.printStackTrace();

}

} }

# Realizando operaciones.

1. Crear (Create)

import com.google.gson.Gson;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Main {

public static void main(String[] args) {

try {

Gson gson = new Gson();

// Leer datos existentes del archivo JSON

Datos datos = gson.fromJson(new FileReader("datos.json"), Datos.class);

// Crear un nuevo producto

Producto nuevoProducto = new Producto("Nuevo Producto", 39.99);

// Agregar el nuevo producto a la lista de productos

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

productos.add(nuevoProducto);

datos.setProductos(productos.toArray(new Producto[0]));

// Escribir los datos actualizados en el archivo JSON

try (FileWriter writer = new FileWriter("datos.json")) {

gson.toJson(datos, writer);

}

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

} catch (IOException e) {

e.printStackTrace();

}

}

}

2. Leer (Read)

Utiliza el código de lectura de Gson que ya has visto en lecciones anteriores.

3. Actualizar (Update)

import com.google.gson.Gson;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

try {

Gson gson = new Gson();

// Leer datos existentes del archivo JSON

Datos datos = gson.fromJson(new FileReader("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

try (FileWriter writer = new FileWriter("datos.json")) {

gson.toJson(datos, writer);

}

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

} else {

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

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

3. Borrar (Delete)

import com.google.gson.Gson;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Main {

public static void main(String[] args) {

try {

Gson gson = new Gson();

// Leer datos existentes del archivo JSON

Datos datos = gson.fromJson(new FileReader("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<>(Arrays.asList(datos.getProductos()));

productos.remove(productoEliminar);

datos.setProductos(productos.toArray(new Producto[0]));

// Escribir los datos actualizados en el archivo JSON

try (FileWriter writer = new FileWriter("datos.json")) {

gson.toJson(datos, writer);

}

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

} else {

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

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**4. Actualizar un archivo JSON**

import com.google.gson.Gson; import java.io.\*; import java.util.Arrays;

public class Main {

public static void main(String[] args) {

try {

Gson gson = new Gson();

// Leer datos existentes del archivo JSON

Datos datos = gson.fromJson(new FileReader("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) {

productoActualizar.setPrecio(24.99); // Actualizar el precio del producto

// Escribir los datos actualizados en el archivo JSON

try (FileWriter writer = new FileWriter("datos.json")) {

gson.toJson(datos, writer);

}

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

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

}

} catch (IOException e) { e.printStackTrace(); }

} }

ANEXOS. Otros ejemplos.

# EJEMPLO. Un **archivo JSON de tres clientes con dos productos cada uno**

{ "clientes":

[ { "nombre": "María", "edad": 30, "dirección": { "calle": "Calle Ejemplo", "ciudad": "Ciudad Ejemplo" }, "productos": [ { "nombre": "Producto 1", "precio": 19.99 }, { "nombre": "Producto 2", "precio": 29.99 } ] },

{ "nombre": "Cliente 1", "edad": 25, "dirección": { "calle": "Calle 1", "ciudad": "Ciudad 1" }, "productos": [ { "nombre": "Producto 3", "precio": 9.99 }, { "nombre": "Producto 4", "precio": 39.99 } ] },

{ "nombre": "Cliente 2", "edad": 35, "dirección": { "calle": "Calle 2", "ciudad": "Ciudad 2" }, "productos": [ { "nombre": "Producto 5", "precio": 14.99 }, { "nombre": "Producto 6", "precio": 24.99 } ] }

]

}

Principio del formulario

**Clases para representar los datos**

import java.util.List;

public class Cliente {

private String nombre;

private int edad;

private Direccion direccion;

private List<Producto> productos;

// Constructor

public Cliente(String nombre, int edad, Direccion direccion, List<Producto> productos) {

this.nombre = nombre;

this.edad = edad;

this.direccion = direccion;

this.productos = productos;

}

// Getters y Setters ….

}

public class Direccion {

private String calle;

private String ciudad;

// Constructor

public Direccion(String calle, String ciudad) {

this.calle = calle;

this.ciudad = ciudad;

}

// Getters y Setters …

}

public class Producto {

private String nombre;

private double precio;

// Constructor

public Producto(String nombre, double precio) {

this.nombre = nombre;

this.precio = precio;

}

// Getters y Setters ….

}

**Muestra el nombre, ciudad un cuantos productos tiene cada cliente.**   
NOTA. En este caso no tenemos ninguna clase que representa la lista de los clientes. Por esta razón se crea un tipo de dato para representar la lista de los clientes

**Rediseñando el código anterior, pero utilizando la clase llamada Datos, que representa la lista de los clientes**

import com.google.gson.Gson;

import java.util.List;

public class Main {

public static void main(String[] args) {

// JSON que contiene la lista de clientes con productos

String json = "..." // Coloca aquí el JSON completo

// Crear una instancia de Gson

Gson gson = new Gson();

// Deserializar el JSON en un objeto Datos que contiene la lista de clientes

Datos datos = gson.fromJson(json, Datos.class);

// Obtener la lista de clientes desde el objeto Datos

List<Cliente> listaClientes = datos.getClientes();

// Iterar a través de la lista de clientes y mostrar nombre, ciudad y cantidad de productos

for (Cliente cliente : listaClientes) {

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

System.out.println("Ciudad del cliente: " + cliente.getDireccion().getCiudad());

System.out.println("Cantidad de productos del cliente: " + cliente.getProductos().size());

System.out.println();

}

}

}

**Clase Datos**

import java.util.List;

public class Datos {

private List<Cliente> clientes;

public List<Cliente> getClientes() {

return clientes;

}

public void setClientes(List<Cliente> clientes) {

this.clientes = clientes;

}

}

# EJEMPLO Gson para trabajar con JSON en formato JSON Tree.

Sea el siguiente JSON

{

"nombre": "Juan",

"edad": 30,

"direcciones": [

{

"calle": "Calle Principal",

"ciudad": "Ciudad A"

},

{

"calle": "Avenida Secundaria",

"ciudad": "Ciudad B"

}

]

}

**Analizar y manipular este Json**

import com.google.gson.JsonArray;

import com.google.gson.JsonElement;

import com.google.gson.JsonObject;

import com.google.gson.JsonParser;

public class JsonElementEjemplo {

public static void main(String[] args) {

// JSON de ejemplo

String json = "{\n" +

" \"nombre\": \"Juan\",\n" +

" \"edad\": 30,\n" +

" \"direcciones\": [\n" +

" {\n" +

" \"calle\": \"Calle Principal\",\n" +

" \"ciudad\": \"Ciudad A\"\n" +

" },\n" +

" {\n" +

" \"calle\": \"Avenida Secundaria\",\n" +

" \"ciudad\": \"Ciudad B\"\n" +

" }\n" +

" ]\n" +

"}";

// Analizar la cadena JSON en un objeto JsonElement

JsonElement elementoJson = JsonParser.parseString(json);

// Verificar si es un objeto JSON

if (elementoJson.isJsonObject()) {

// Convertir a JsonObject

JsonObject jsonObject = elementoJson.getAsJsonObject();

// Obtener valores individuales

String nombre = jsonObject.get("nombre").getAsString();

int edad = jsonObject.get("edad").getAsInt();

System.out.println("Nombre: " + nombre);

System.out.println("Edad: " + edad);

// Obtener un objeto JSON anidado

JsonArray direcciones = jsonObject.getAsJsonArray("direcciones");

// Iterar a través de las direcciones

for (JsonElement direccion : direcciones) {

JsonObject direccionObj = direccion.getAsJsonObject();

String calle = direccionObj.get("calle").getAsString();

String ciudad = direccionObj.get("ciudad").getAsString();

System.out.println("Dirección: " + calle + ", " + ciudad);

}

}

}

}

En este ejemplo, hemos utilizado **JsonParser.parseString(json)** para analizar la cadena JSON en un objeto **JsonElement**. Luego, verificamos si es un objeto JSON y lo convertimos en un **JsonObject**. A partir de ahí, accedemos a los valores individuales y al objeto JSON anidado "direcciones" para obtener y mostrar la información.

Este enfoque es útil cuando necesitas trabajar con JSON en un formato estructurado sin tener que crear clases Java correspondientes. Puedes navegar por la estructura JSON utilizando métodos como **getAsJsonObject()**, **getAsJsonArray()**, **getAsString()**, y otros métodos proporcionados por **JsonElement**.

# Ejemplo Gson para conversion de un Map a JSON

import com.google.gson.Gson;

import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;

import java.util.HashMap;

import java.util.Map;

public class MapToJsonAndJsonToMapExample {

public static void main(String[] args) {

**// Crear un mapa**

Map<String, Object> mapa = new HashMap<>();

mapa.put("nombre", "Juan");

mapa.put("edad", 30);

mapa.put("ciudad", "Ciudad A");

**// Crear una instancia de Gson**

Gson gson = new Gson();

**// Convertir el mapa a una cadena JSON**

String json = gson.toJson(mapa);

// Imprimir la cadena JSON resultante

System.out.println("Mapa convertido a JSON:");

System.out.println(json);

// Deserializar la cadena JSON en un mapa

Type tipoMapa = new TypeToken<Map<String, Object>>(){}.getType();

Map<String, Object> mapaDeserializado = gson.fromJson(json, tipoMapa);

// Acceder a los valores en el mapa deserializado

String nombre = (String) mapaDeserializado.get("nombre");

int edad = (int) mapaDeserializado.get("edad");

String ciudad = (String) mapaDeserializado.get("ciudad");

// Imprimir los valores del mapa deserializado

System.out.println("\nMapa deserializado desde JSON:");

System.out.println("Nombre: " + nombre);

System.out.println("Edad: " + edad);

System.out.println("Ciudad: " + ciudad);

}

}

En este ejemplo, primero creamos un mapa (HashMap) llamado mapa y lo llenamos con algunos datos. Luego, utilizamos Gson para convertir este mapa en una cadena JSON utilizando el método toJson. A continuación, imprimimos la cadena JSON resultante.

Luego, utilizamos el método fromJson de Gson para deserializar la cadena JSON nuevamente en un mapa. Para hacer esto, necesitamos especificar el tipo de dato adecuado utilizando TypeToken de Gson. Finalmente, accedemos a los valores dentro del mapa deserializado y los imprimimos.

Este ejemplo ilustra cómo puedes convertir fácilmente un mapa en formato JSON y viceversa utilizando Gson, lo que puede ser útil en situaciones donde necesitas trabajar con datos JSON de manera dinámica.