4️⃣

GSON —> Creación de instancias Gson

1. Introducción a la Clase Gson

- **Gson** es una clase principal para trabajar con la conversión entre objetos Java y JSON.
- Permite la **serialización** (convertir objetos Java a JSON) y la **deserialización** (convertir JSON a objetos Java).

---

2. Creación de una Instancia de Gson

Existen dos formas principales para crear una instancia de la clase `Gson`:

2.1. Creación con `new Gson()`

- **Sintaxis simple**:

```
Gson gson = new Gson();
```

💡 Se usa cuando se desea trabajar con la **configuración predeterminada** de Gson.

2.2. Creación con `GsonBuilder.create()`

- **GsonBuilder** permite personalizar la configuración antes de crear el objeto `Gson`.
- Ejemplo de código:

```
Gson gson = new GsonBuilder()
    .registerTypeAdapter(Id.class, new IdTypeAdapter())
    .serializeNulls()
    .setDateFormat(DateFormat.LONG)
    .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
    .setPrettyPrinting()
    .setVersion(1.0)
    .create();
```

- El método `create()` genera el objeto `Gson` después de configurar diversas opciones.

💡 Se usa cuando se desea trabajar con la **configuración personalizada** de Gson.

2.3. Configuración Predeterminada y Personalización

- **Configuración predeterminada** de Gson:
  - Serialización y deserialización para clases comunes como `java.util.Date`, `java.net.URL`, entre otras.
  - Por defecto, omite los **campos nulos** en la serialización, pero los **conserva en los arrays**.
  - El formato de fecha es el predeterminado de `java.text.DateFormat`.

- Usa la convención de **nombres de campos de Java** en la salida JSON (ej. `versionNumber`).
- El formato JSON generado es compacto, sin espacios en blanco innecesarios.

- **Opciones de configuración personalizadas**:
  - **Campos nulos**: `GsonBuilder.serializeNulls()`
  - **Formato de fecha**: `GsonBuilder.setDateFormat(int)` o `GsonBuilder.setDateFormat(String)`
  - **Política de nombres de campos**: `GsonBuilder.setFieldNamingPolicy(FieldNamingPolicy)`
  - **Formato de presentacion**: `GsonBuilder.setPrettyPrinting()`
  - **Manejo de versiones**: `GsonBuilder.setVersion(double)`
  - **Anotaciones** `@Expose` : `GsonBuilder.excludeFieldsWithoutExposeAnnotation()`

---

3. Conversión entre Primitivas JSON y sus Equivalentes Java: `fromJson()` y `toJson()`

- **Método** `fromJson()` : Convierte una cadena JSON a un objeto Java.
- **Método** `toJson()` : Convierte un objeto Java a JSON.

Ejemplo de Código:

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import static java.lang.System.*;

public class GsonDemo {
    public static void main(String[] args) {
        Gson gson = new Gson();

        // Deserialización de una cadena JSON
        String nome = gson.fromJson("\"Sylvia Plath\"", String.class);
        out.println(nome);  // Salida: Sylvia Plath
```

```
        // Serialización de un entero
        gson.toJson(256, out);  // Salida: 256
        out.println();

        // Serialización de una cadena de texto
        gson.toJson("<html>", out);  // Salida: "<html>"
        out.println();

        // Gson personalizado (deshabilitando el escapado de HTML)
        gson = new GsonBuilder().disableHtmlEscaping().create();
        gson.toJson("<html>", out);  // Salida: <html> (sin escape)
        out.println();
    }
}
```

Explicación:

1. **Deserialización**:
   - `fromJson("\"Sylvia Plath\"", String.class)` convierte el texto JSON `"Sylvia Plath"` en un objeto Java de tipo `String`.
2. **Serialización**:
   - `toJson(256, out)` convierte el entero `256` en un valor JSON y lo imprime.
   - `toJson("<html>", out)` convierte la cadena `<html>` en un formato JSON. Por defecto, los caracteres especiales como `<` y `>` son escapados.
3. **Gson Personalizado**:
   - `disableHtmlEscaping()` evita que Gson escape los caracteres especiales en las cadenas. En el ejemplo, el texto `<html>` se imprime tal cual, sin ser escapado a `<html>`.

Ejercicio Propuesto:

- Crear un proyecto y compilar el código anterior para observar cómo la conversión entre primitivas JSON y objetos Java se realiza con los métodos `fromJson()` y `toJson()`.

Resumen de Métodos Clave:

- `fromJson()`: Convierte JSON a objeto Java.
  - **Sintaxis**: `fromJson(String json, Class<T> classOfT)`
- `toJson()`: Convierte objeto Java a JSON.
  - **Sintaxis**: `toJson(Object src, Appendable writer)`

Con estos métodos puedes convertir fácilmente entre JSON y objetos Java utilizando la biblioteca Gson.