5️⃣

GSON —> Creación y lectura de objetos JSON

1. De Java a JSON: `toJson()`

Gson puede generar un archivo JSON a partir de objetos Java mediante el método `toJson()` de la clase `Gson`.

- **Ejemplo básico:**

```
Poeta poeta = new Poeta();
poeta.setNome("Sylvia Plath");
poeta.setIdade(30);
```

```
Gson gson = new Gson();
String json = gson.toJson(poeta);
```

- **Salida con formato estandar:**

```
{"nome":"Sylvia Plath","idade":30}
```

- **Impresión con formato "bonito"**: `setPrettyPrinting()`

> 💡 Gson genera por defecto una salida JSON compacta, pero puedes hacerla más legible con `setPrettyPrinting()` de `GsonBuilder`:

```
Gson gson = new GsonBuilder()
                    .setPrettyPrinting()
                    .create();
String json = gson.toJson(poeta);
```

- **Salida con formato** `setPrettyPrinting()` :

```
{
  "nome": "Sylvia Plath",
  "idade": 30
}
```

---

2. De JSON a Java: `fromJson()`

Gson también puede convertir JSON en objetos Java usando el método `fromJson()`.

- **Ejemplo básico:**

```
String textoJson = "{\"nome\":\"Sylvia Plath\", \"idade\": 30}"; //Cadena Jsona
Gson gson = new Gson();
Poeta poeta = gson.fromJson(textoJson, Poeta.class);// Debemos indicar el ti
```

> El **primer parámetro** de `fromJson()` es la fuente JSON (`String`, `Reader`, `JsonReader` o `JsonElement`).

> El **segundo parámetro** del método `fromJson()` es la clase de Java para analizar el JSON en una instancia.

> **!** En el ejemplo anterior, la fuente JSON es una cadena, pero existen varias versiones de este método (sobrecargado).

Sobrecarga de `fromJson()`

Existen varias versiones sobrecargadas de `fromJson()` que permiten analizar JSON desde diferentes fuentes como `String`, `Reader`, o `JsonReader`, `JsonElement` (Estas dos últimas procedentes del API de Gson).

```
public <T> T fromJson (String json, Class<T> classOfT)
   throws JsonSyntaxException;

public <T> T fromJson (String json, Type typeOfT)
   throws JsonSyntaxException;

public <T> T fromJson (String json, TypeToken<T> typeOfT)
   throws JsonSyntaxException;

public <T> T fromJson (Reader json, Class<T> classOfT)
   throws JsonSyntaxException, JsonIOException

public <T> T fromJson (Reader json, Type typeOfT)
   throws JsonIOException, JsonSyntaxException;

public <T> T fromJson (Reader json, TypeToken<T> typeOfT)
   throws JsonIOException, JsonSyntaxException;

public <T> T fromJson (JsonReader reader, Type typeOfT)
```

```
    throws JsonIOException, JsonSyntaxException;

public <T> T fromJson (JsonReader reader, TypeToken<T> typeOfT)
    throws JsonIOException, JsonSyntaxException;

public <T> T fromJson (JsonElement json, Class<T> classOfT)
    throws JsonSyntaxException;

public <T> T fromJson (JsonElement json, Type typeOfT)
    throws JsonSyntaxException;

public <T> T fromJson (JsonElement json, TypeToken<T> typeOfT)
    throws JsonSyntaxException;
```

---

3. Exclusión de Atributos en la Serialización

Puedes **excluir ciertos atributos** de tus objetos Java durante la serialización con Gson.

3.1. Atributos `transient`

> 🗣️ Los atributos marcados como `transient` **no se incluirán en la serialización ni deserialización.**

```
public class Poeta {
    public transient String nome;  // No se serializa
    public int idade;
}
```

3.2. Anotación `@Expose`

- Para mayor control, puedes usar la anotación `@Expose` y configurar `GsonBuilder` para que **excluya los campos sin esta anotación**.

> ! La anotación `expose` solo funcionara si se crea un objeto de tipo `GsonBuilder`

- La anotación `@Expose` **puede tener dos parámetros:** `serialize` **y** `deserialize`, ambos son **booleanos** que pueden tener los valores `true` o `false`:
  - El parámetro `serialize` de la anotación `@Expose` indica si el atributo anotado con la `@Expose` debe incluirse cuando el objeto se serializa.
  - El parámetro `deserialize` anota si ese atributo debe leerse cuando el objeto se deserializa.

```
public class Estudiante {
  @Expose private String nome; // Se incluirá en la serialización y deserializac
  @Expose(serialize = false) private String apelidos;//no serializa
  @Expose (serialize = false, deserialize = false) private String email; // ni seri
  private String password; // ... ? NO LO SERIALIZA NI DESERIALIZA (sin anota
}
```

▼ Formatos anotación @Expose:

- `@Expose(serialize = true);`
- `@Expose(serialize = false);`
- `@Expose(deserialize = true);`
- `@Expose(deserialize = false);`
- `@Expose(serialize = true, deserialize = false);`
- `@Expose(serialize = false, deserialize = true);`

```
Gson gson = new GsonBuilder().excludeFieldsWithoutExposeAnnotation().crea
```

3.3. Exclusión con `ExclusionStrategy`

Si necesitas más control, puedes implementar la interfaz `ExclusionStrategy` para excluir campos de forma personalizada. Esta estrategia te **permite especificar qué campos o clases deben ser excluidos.**

! `ExclusionStrategy` es una interfaz, por lo que hay que **crear una clase que implemente la interfaz** `ExclusionStrategy`.

```
ExclusionStrategy exclusion = new ExclusionStrategy() {
public boolean shouldSkipField(FieldAttributes field) {
return field.getName().equals("password");
}
public boolean shouldSkipClass(Class<?> clazz) {
return false;
}
};
Gson gson = new GsonBuilder().setExclusionStrategies(exclusion).create();
```

3.4. Serialización de Campos Nulos

Por defecto, Gson excluye campos null. Para incluirlos:

```
Gson gson = new GsonBuilder().serializeNulls().create();
```

- Ejemplo:

```
{"nome": null, "idade": 30}
```

Resumen de Métodos Clave de Gson

- `toJson(Object src)`: Convierte un objeto Java a JSON.
- `fromJson(String json, Class<T> classOfT)`: Convierte JSON a un objeto Java.
- `GsonBuilder.setPrettyPrinting()`: Formato de salida legible.
- `GsonBuilder.excludeFieldsWithoutExposeAnnotation()`: Excluye atributos no marcados con `@Expose`.
- `GsonBuilder.serializeNulls()`: Incluye campos con valor `null` en la salida JSON.

💡 Con estos conceptos y ejemplos, puedes trabajar con JSON de manera eficiente en tus aplicaciones Java usando la biblioteca Gson.