

API GSON

Building JSON parsers

- La API que se va a utilizar es la proporcionada por google: GSON.
- Es una biblioteca de código abierto que permite convertir un objeto Java en su representación JSON (serialización) y un elemento (array, objeto, string ...) JSON en su correspondiente objeto Java (deserialización).
- En <https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.6/> está disponible la ultima versión (2.8.6).
- Descargar:
 - gson-2.8.6.jar (que se incluirá en los proyectos Eclipse).
 - gson-2.8.6-javadoc.jar (javadoc de GSON).

■ In

<https://jar-download.com/artifacts/com.google.code.gson/gson/2.8.6/source-code>

is available the source code of the jar.

■ In

<https://github.com/google/gson/blob/master/UserGuide.md>

is available the GSON User Guide.

■ In

<https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/com/google/gson/package-summary.html>

is available the javadoc of the API GSON.

- GSON provide three API to create a JSON parser:
 - **Data Binding API**: it converts JSON to and from POJO (Plain Old Java Object) using property accessors.
 - **Tree Model API**: creates an in-memory tree of `JsonElement` that is the representation of the JSON document. It is like an XML DOM parser.
 - **Stream Model API**: It reads and writes the JSON the content of the document as discrete token with `JsonReader` and `JsonWriter`. These classes read data as `JsonTokens`. It has the lowest overhead of the three API and it is quite fast in read/write operation. It is like an XML StAX parser.

- A GSON object can be created using two ways:
 - Simple and faster (Gson):

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

- Configuring the object (GsonBuilder):

```
Gson gson = new GsonBuilder()  
    .disableHtmlEscaping()  
    .setFieldNamingPolicy  
        (FieldNamingPolicy.UPPER_CAMEL_CASE)  
    .setPrettyPrinting()  
    .serializeNulls()  
    .create();
```

```
{  
  "nombre":"Juan Garcia",  
  "lugar_nacimiento":"Madrid",  
  "exámenes":[  
    {"asignatura":"Programacion I",  
     "nota":4.5},  
    {"asignatura":"Redes II",  
     "nota":9.0}  
  ]  
}
```

JSON doc

fromJson()



```
class Alumno{  
  
  private final String nombre;  
  private final String lugar_nacimiento;  
  private final List<Examen> examenes;  
  public Alumno (String nombre, String lugar_nacimiento,  
                  List<Examen> examenes){  
    this.nombre = nombre;  
    this.lugar_nacimiento = lugar_nacimiento;  
    this.exámenes = examenes;  
  }  
  //getters & setters & toString()  
}
```

Java class
(POJO)

See more in:
[1_fromJson_Example](#)

```
{  
  "nombre": "Juan Garcia",  
  "lugar_nacimiento": "Madrid",  
  "exámenes": [  
    {"asignatura": "Programacion I",  
     "nota": 4.5},  
    {"asignatura": "Redes II",  
     "nota": 9.0}  
  ]  
}
```

JSON doc

toJson()

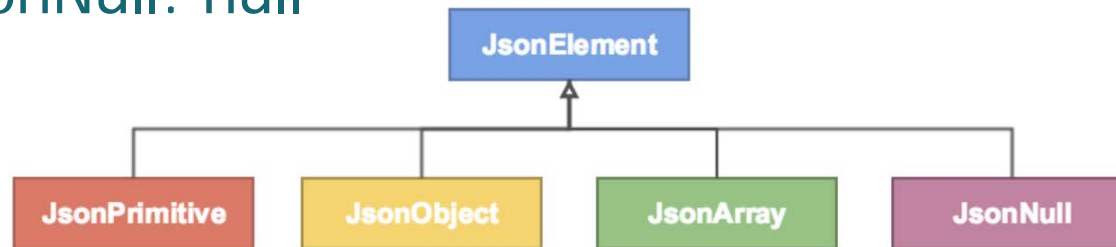


```
class Alumno{  
  
  private final String nombre;  
  private final String lugar_nacimiento;  
  private final List<Examen> examenes;  
  public Alumno (String nombre, String lugar_nacimiento,  
                  List<Examen> examenes){  
    this.nombre = nombre;  
    this.lugar_nacimiento = lugar_nacimiento;  
    this.exámenes = examenes;  
  }  
  //getters & setters & toString()  
}
```

Java class
(POJO)

See more in:
2_toJson_Example

- Tree Model API creates an in-memory tree of `JsonElement` that it is the representation of the JSON.
- It builds a tree of `JsonElement`.
- `JsonElement` is a class representing an element of JSON. It could either be a:
 - `JsonObject`: set of pairs *key* ↔ *value*
 - `JsonArray`: ordered list of `JsonElements`.
 - `JsonPrimitive`: Boolean, String, Number
 - `JsonNull`: null



<https://jarroba.com/gson-json-java-ejemplos/>

- To parse a JSON document into a in-memory tree we can use the `JsonParser()` to parse the JSON string into a Tree Model of type `JsonElement`. (1)
- The `getAsJsonObject()` method of `JsonElement` can be used to get the element as `JsonObject`. (2)
- The `getAsJsonArray()` method of `JsonElement` can be used to get the element as `JsonArray`. (3)

See more in: [3_TreeModelRead_Example](#)

```
JsonParser parser = new JsonParser(); //(1)
JsonElement tree = parser.parse(reader); //(1)
JsonArray arrayAlumnos = tree.getAsJsonArray(); //(3)
for (JsonElement element : arrayAlumnos) {
    if (element.isJsonObject()) {...} //(2)
...}
```

reader must be an object of the Reader class
(fileReader, BufferedReader,)

In (3) we get the array of Alumnos.*

In the for sentence each object Alumno is obtained using
(2), and it starts its analysis.

See more in: 3_TreeModelRead_Example

* (see alumnosSalidaWrite.json of project
3_TreeModelRead_Example know the structure)

- To write the content of a in-memory tree in a Json document we can use `toJsonTree()` to serialize the specified object into its equivalent representation as a tree of `JsonElement`.

See more in: [4_TreeModelWrite_Example](#)

```
List<Alumno> alumnos = new ArrayList<>();  
alumnos.add (.....);  
alumnos.add (.....);    //(1)  
  
Gson gson = new Gson();  
JsonElement tree = gson.toJsonTree(alumnos); //(2)  
Gson.toJson(tree,writer);    //(3)
```

In (1) we fill in the structure (List, Array ...) of Alumnos.*

In the for sentence each object Alumno is obtained using (2), and it starts its analysis.

writer must be an object of the Writer class (fileWriter, BufferedWriter,). *

See more in: 4_TreeModelWrite_Example

* (see alumnosOUT.json in the folder of project 4_TreeModelWrite_Example)


- Permite procesar documentos JSON como una secuencia de token que son proporcionados en el orden en el que aparecen los objetos JSON en el documento.
- Es implementado mediante la API GSON Streaming usando las clases `JsonReader` y `JsonWriter`.
- Es útil en situaciones en las que no es necesario cargar el árbol completo que representa al documento en memoria, haciendo el parser más ligero y con menos gasto de recursos que en Tree Model.

- Lee un documento JSON y proporciona un stream de tokens (JsonToken).
- El stream incluye literales (string, numbers, boolean y null) y principio y final de objetos ({,}) y de arrays ([,]).
- El orden de aparición de los token es el mismo que la jerarquía en la que aparecen en el documento JSON.
- Cada objeto JSON es representado como un único token.

Enum Constant	Description
<u>BEGIN_ARRAY</u>	The opening of a JSON array.
<u>BEGIN_OBJECT</u>	The opening of a JSON object.
<u>BOOLEAN</u>	A JSON true or false.
<u>END_ARRAY</u>	The closing of a JSON array.
<u>END_DOCUMENT</u>	The end of the JSON stream.
<u>END_OBJECT</u>	The closing of a JSON object.
<u>NAME</u>	A JSON property name.
<u>NULL</u>	A JSON null.
<u>NUMBER</u>	A JSON number represented in this API by a Java double, long, or int.
<u>STRING</u>	A JSON string.


<https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/com/google/gson/stream/JsonToken.html>

```
{
  "nombre":"Juan Garcia",
  "examenes":[
    {
      "asignatura":"PIAT",
      "nota":7.5
    },
    {
      "asignatura":"Redes",
      "nota":null
    }
  ]
  "expulsado":false
}
```



	Token	JsonToken
1	{	BEGIN_OBJECT
2	nombre	NAME
3	Juan Garcia	STRING
4	examenes	NAME
5	[BEGIN_ARRAY
6	{	BEGIN_OBJECT
7	asignatura	NAME
8	PIAT	STRING
9	nota	NAME
10	7.5	NUMBER
11	}	END_OBJECT


```
{  
  "nombre":"Juan Garcia",  
  "exámenes":[  
    {  
      "asignatura":"PIAT",  
      "nota":4.5  
    },  
    {  
      "asignatura":"Redes",  
      "nota":null  
    }  
  ]  
  "expulsado":false  
}
```



	Token	Tipo
12	{	BEGIN_OBJECT
13	asignatura	NAME
14	Redes	STRING
15	nota	NAME
16	null	NULL
17	}	END_OBJECT
18]	END_ARRAY
19	expulsado	NAME
20	false	BOOLEAN
21	}	END_OBJECT
22		END_DOCUMENT

- Primero se crea el método de entrada (puede ser `main()`) que crea un `JsonReader`.
- Se crea un manejador para cada estructura (objeto o array) del documento JSON:
 - Objeto:
 - Se invoca a `beginObject()` que consume "{".
 - Se hace un bucle que asigna valores a las variables locales hasta que `hasnext()` sea false.
 - Se invoca a `endObject()` que consume "}".
 - Array:
 - Se invoca a `beginArray()` que consume "[".
 - Se hace un bucle de proceso hasta que `hasnext()` sea false.
 - Se invoca a `endArray()` que consume "]".

- Cuando se encuentra un nuevo objeto o array se delega al manejador correspondiente.
- Cuando se encuentra un objeto desconocido, salta una excepción. Si el parser es "lenient" (tolerante) se puede saltar ese valor con `skipvalue()`.
- El método `peek()` (ojear) devuelve el tipo del siguiente objeto que se va a devolver, pero sin devolver su valor y por tanto, no se produce ni se consume.
- Un `null` se puede consumir con `nextnull()`.

See more in: `5_StreamReaderAlumno_Example` and
`6_StreamReaderAlumnos_Example`

- Escribe un documento JSON a partir de tokens.
- Los token son literales (string, numbers, boolean y null) y principio y final de objetos ({,}) y de arrays ([,]).
- El orden de aparición de los token es el mismo que la jerarquía en la que aparecerán en el documento JSON.
- Cada token representará un objeto JSON.

- Primero se crea el método de entrada que crea un `JsonWriter`.
- Se crea un manejador para cada estructura (objeto o array) del documento JSON:
 - Objeto:
 - Se invoca a `beginObject()` que produce "{".
 - `name()` y `value()` producen objetos y valores
 - Se invoca a `endObject()` que produce "}".
 - Array:
 - Se invoca a `beginArray()` que produce "[".
 - `name()` y `value()` producen objetos y/o valores
 - Se invoca a `endArray()` que produce "]".

See more in: `7_StreamWriterAlumno_Example` and
`8_StreamWriterAlumnos_Example`

- Se crea un objeto GSON.
- Mediante el uso de `fromJson()` se obtienen los objetos del documento JSON en el parser.
- Mediante el uso de `toJson()` se pasan los objetos al documento JSON de salida.
- Se ha de redefinir el método `toString()` en cada clase para que tenga el comportamiento deseado.

See more in: `9_MixedReaderAlumnos_Example` and
`9_Z_MixedReaderAlumnos_Example`

API GSON

Building JSON parsers