Alumno: oRLAndo enrique pasaca rojas

PROfesor: ERICK GUSTAVO CORONEL CASTILLO

MANEJO DE DATOS EN FORMATO JSON

Sistemas UNI

2020

**DEDICATORIA**

El presente trabajo investigativo lo dedico principalmente a Dios, por ser el inspirador y darme fuerza para continuar en este proceso de obtener nuevos conocimientos como estudiante.  
  
A mis padres, por su amor, trabajo y sacrificio en todos estos años, gracias a ustedes he logrado seguir desarrollándome como persona y convertirme en lo que seré más adelante. Ha sido el orgullo y el privilegio de ser su hijo, son los mejores padres.  
  
A mis hermanos por estar siempre presentes, acompañándome y por el apoyo moral, que me brindan a lo largo de esta etapa de mi vida.

INDICE

|  |  |
| --- | --- |
| Resumen | 5 |
| ¿Qué es Json? | 6 |
| Sintaxis | 6 |
| Json en JAVA | 8 |
| Gson | 8 |
| Utilizando la librería Gson | 9 |
| Serializando un objeto a Json | 9 |
| Deserializando Json a un objeto propio | 12 |
| Serializando nuestro objeto propio a JSON “bonito” | 13 |
| Deserializando JSON en una lista de objetos propios | 14 |
| Deserializando JSON en un objeto genérico | 17 |
| Deserializando fechas de JSON a un objeto propio | 18  21 |
| Conclusiones |
| Recomendaciones | 22 |

Resumen

El siguiente trabajo de investigación esta orientado a conocer, explicar y aplicar correctamente el manejo de datos en formato JSON. Así mismo hacer énfasis en la importancia y su uso comúnmente realizado para transmitir datos en aplicaciones web. Nos encontraremos muy a menudo en situaciones en donde tendremos que hacer uso de este tipo de datos, es por ello que en este trabajo de investigación tratare de abarcar todo lo que se necesita saber para trabajar con JSON en JAVA y en otro tipo de lenguajes.

Veremos un análisis para poder acceder a datos en su interior, y como crear JSON. Así como también veremos ejemplos sobre su uso.

**¿Qué es JSON?**

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y a la vez legible para humanos (como [XML](http://diego.com.es/xml-principios-basicos) pero sin el marcado). Su sintaxis es un subconjunto del lenguaje JavaScript que fue estandarizado en 1999.

Lo bueno de JSON es que se puede manejar de forma nativa en JavaScript, por lo que actúa como un pegamento entre el servidor y la aplicación del lado del cliente. Además, como sintácticamente es muy simple, se tienen que transmitir menos bytes en cada transacción. En aplicaciones modernas JSON ha sustituido en bastantes situaciones a XML.

**SINTAXIS**

Los tipos de datos disponibles con JSON son:

* **Números:** Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos.

Ejemplo: 123.456

* **Cadenas:** Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape.

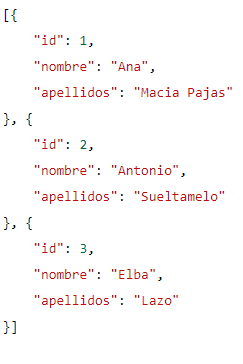
Ejemplo: "Hola"

* **Booleanos:** Representan valores booleanos y pueden tener dos valores: true y false
* **Null:** Representan el valor nulo.
* **Array:** Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes.

Ejemplo: [ "juan" , "pedro" , "jacinto" ]

* Objetos: Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena y entre ellas. El valor puede ser de cualquier tipo.

Ejemplo:



**JSON en JAVA**

Java por defecto no ofrece ninguna funcionalidad integrada para analizar o crear JSON, a diferencia de PHP u otros idiomas, en lugar de eso, tendrás que confiar en una biblioteca/paquete de terceros.

Puede darse la situación de que en nuestra aplicación Java, necesitemos atender peticiones representadas en JSON, transformarlas a Java, tratar los datos y devolver una respuesta en JSON. Los [servicios REST](https://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=client_axis_android) o los [Websockets](https://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=WebSocketsJavaTomcat) son un buen ejemplo de esto.

Para resolver este problema podemos implementar «a mano» la lógica de negocio para serializar y deserializar nuestro JSON, lo que supondrá un esfuerzo considerable. O podríamos hacer uso de alguna librería diseñada para este propósito como puede ser Gson.

Gson

Gson (también conocido como Google Gson) es una [biblioteca](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto) para el [lenguaje de programación Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) que permite la [serialización](https://es.wikipedia.org/wiki/Serializaci%C3%B3n) y deserialización entre [objetos](https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)) Java y su representación en notación [JSON](https://es.wikipedia.org/wiki/JSON).

Características

* Permite la conversión entre objetos Java y JSON de una manera sencilla, simplemente invocando los métodos toJson () o fromJson ().
* Permite la conversión de objetos inmutables ya existentes.
* Soporte para tipos genéricos de Java.
* Permite la representación personalizada de objetos.
* Soporte para "Objetos arbitrariamente complejos".

**Utilizando la librería Gson en JAVA**

Antes de comenzar a usar la librería es necesario descargarla y añadir el JAR en la carpeta Libraries de tu proyecto en NetBeans.

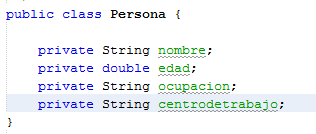
Dejare el link de descarga a continuación:

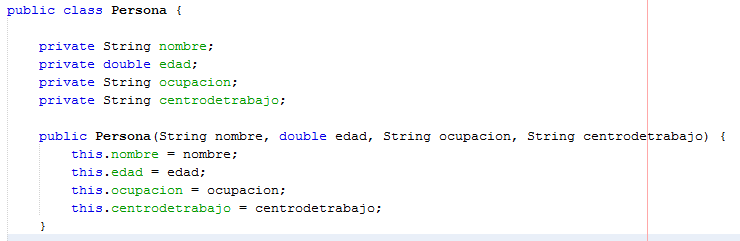
<https://search.maven.org/artifact/com.google.code.gson/gson/2.8.0/jar>

Bueno una vez realizado el paso anterior podemos comenzar a jugar con esta librería.

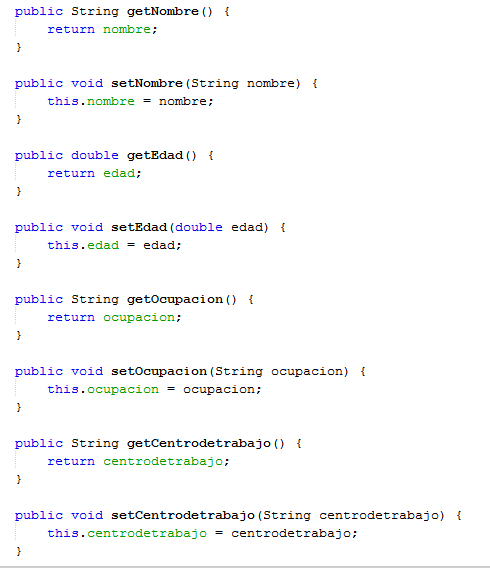
Serializando un objeto a JSON

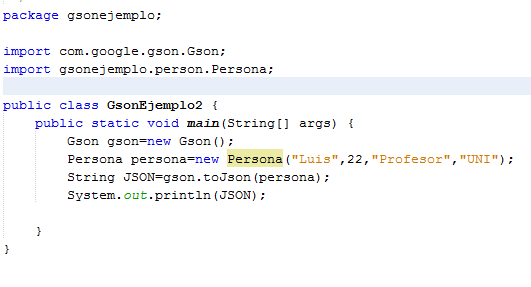
Creamos la clase Persona y las variables nombre, persona, edad, ocupación, centro de trabajo:

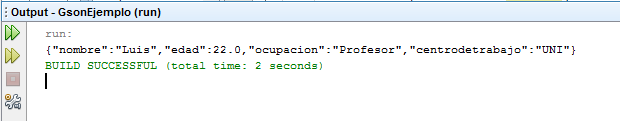


Creamos su constructor:

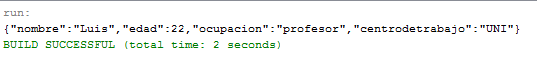
De la misma manera creamos sus getters y setters:



Ahora crearemos una instancia de Gson e invocaremos su método toJson al que le pasaremos el objeto que queremos serializaren JSON

Y nos devolvería en consola el objeto persona representado en JSON

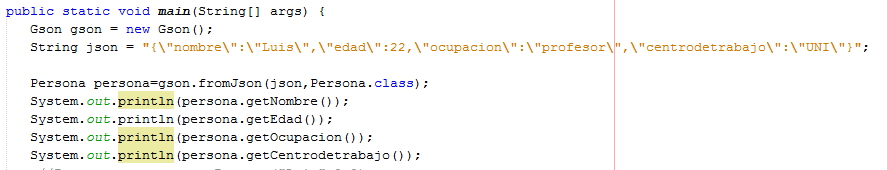
Ahora es muy importante entender que para representar un JSON mediante un STRING tendríamos que hacerlos de la siguiente manera.

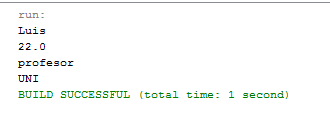
Y en consola nos imprimiría su representacion.

Deserializando JSON a un objeto propio

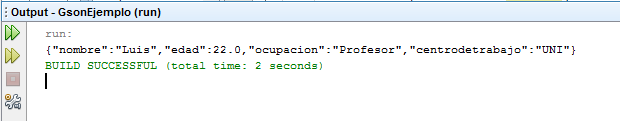
Ahora ya tenemos nuestro JSON pero, ¿cómo usamos sus datos?

Partiendo de la explicación anterior y utilizando la misma clase Persona que habíamos creado anteriormente, utilizaremos el método fromJson a la que le pasaremos la clase Persona para deserializar nuestro JSON.

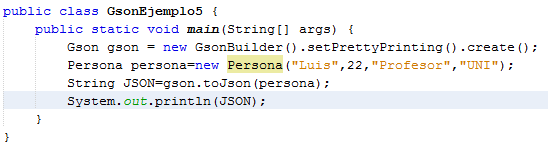
El código quedaría de la siguiente manera:

Y en la consola nos devolvería lo siguiente:

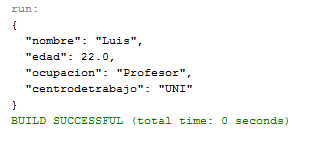
Serializando nuestro objeto propio en JSON “bonito”.

Si nos fijamos bien en la siguiente imagen, la representación JSON de nuestra “Persona” viene bastante comprimida.

Todo en una línea y sin espacios o tabulaciones. Es posible que en algunos casos queramos mostrar la representación JSON de una forma más clara, por ejemplo en un fichero de log. Si lo hiciésemos de la forma del ejemplo anterior y si el objeto fuese más complejo es muy probable que nos costase interpretar la información.

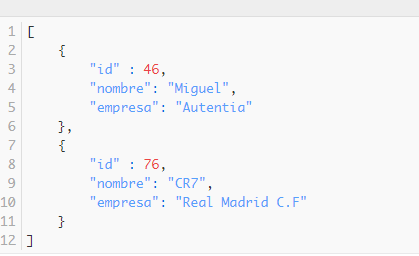
Gson nos permite crear representaciones JSON un poco más vistosas. Para ello debemos crear una instancia de Gson con **GsonBuilder**. Activamos el modo **PrettyPrinting** invocando al método **setPrettyPrinting** y creamos la instancia con el método **create**.

Tendríamos lo siguiente:



Deserializando JSON en una lista de objetos propios.

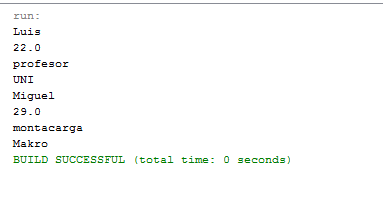
Este ejemplo es parecido al segundo, pero con la salvedad de que ahora no convertimos un objeto JSON en un objeto Java propio, sino en una lista de éstos. El array de objetos JSON es el siguiente:



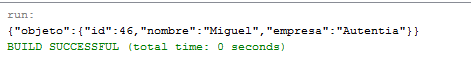
Como vemos hay dos empleados en el array. Para que Gson comprenda que tiene que deserializar el objeto JSON en una lista de objetos propios hacemos lo mismo que en los ejemplos anteriores. La única diferencia es que ahora el segundo parámetro del método fromJSON no es la clase a la que queremos deserializar el objeto JSON sino un objeto Type (java.lang.reflect.Type) que habremos creado mediante **TypeToken**. Al crear una instancia de TypeToken, lo tipamos con la lista de «Empleado», invocamos a su método **getType** y ya tenemos nuestra instancia de Type.



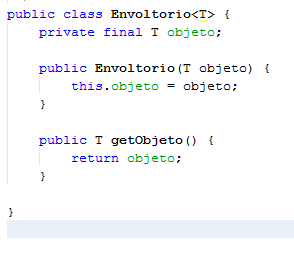
Resultado:

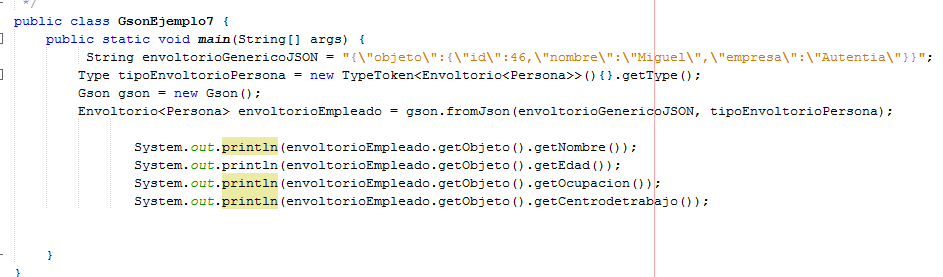


**Deserializando JSON en un objeto genérico.**



Como hemos visto en el ejemplo anterior Gson es capaz de deserializar un objeto JSON en una lista tipada. Por supuesto esto es extensible a cualquier otra clase genérica. Veamos un ejemplo. Supongamos que tenemos la clase **Envoltorio**:

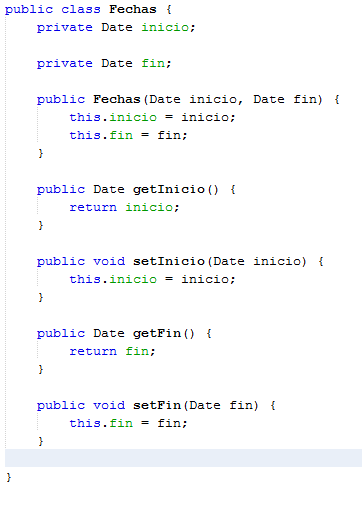


Para deserializar un objeto JSON en una de estas clases con tipo genérico, lo hacemos de la misma forma que en el ejemplo anterior. Usamos un «TypeToken»:

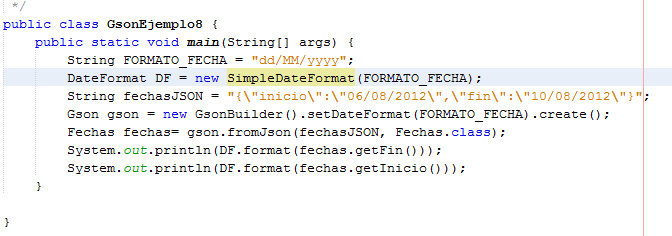
Deserializando fechas de JSON a un objeto propio.

En este ejemplo vamos a ver cómo transformar fechas de nuestro objeto JSON en fechas de nuestro objeto Java (java.util.Date). Nuestro objeto JSON representa un intervalo de fechas. Es el siguiente:

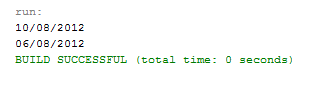
La clase Java que representará el intervalo de fechas será la siguiente:



Para transformar las fechas lo haremos de la misma forma que en los ejemplos anteriores pero, al instanciar nuestro objeto Gson, lo haremos con GsonBuilder. Inicializamos el formato de fecha con el método **setDateFormat** (en nuestro caso dd/MM/yyyy) y creamos la instancia con el método create. Lo vemos en el siguiente test:



Resultado:



**Conclusiones**

En esta oportunidad hemos visto una manera extremadamente rápida de utilizar Json en Java, también hemos presentado la librería Gson, una librería excelente para trabajar con objetos JSON y Java.

En definitiva, una librería buena, bonita y barata para trabajar con JSON y Java

Para el que no esté todavía muy convencido con Gson y quiera buscar otras alternativas, en la web hay muchas otras librerías para poder trabajar con Json

Recomendaciones

Ya hemos visto varias maneras de trabajar con Json en JAVA, pero aún hay muchos temas que faltan abarcar como por ejemplo poder leer un fichero txt con un objeto Json en su interior, por lo que es importante seguir investigando y practicando este tema que es muy utilizado para enviar y recibir datos entre aplicaciones