Introduction to Gson

JSON (JavaScript Object Notation) is one of the most common formats for sending data over a network, writing configuration files, and saving local data in the form of key-value pairs.

The Jackson JSON library is often thought of as the default Java JSON library (it's even configured by default in Spring Boot), Google's **Gson library** for JSON is a capable alternative.

Serialization:

Serialization is the process of converting data objects (in our case, Java objects) into a format that is easy to store or transfer, and can be easily reconstructed back into a data object at a later time.

What is the purpose of JSON? Why do we use it? The answer is because it helps us to serialize our data.

JSON is not the only format for serializing objects; XML is also commonly used for that purpose.

JSON is typically more lightweight and readable, especially when the data being formatted makes sense as a set of keys and their corresponding values.

One of the main benefits of using a universal data format like JSON or XML is that the data is readable in its serialized form.

However, if we want to do anything with the data, we will need to deserialize it. In other words, the data can be deserialized into an object of any language that has objects.

Installing Gson

We take 2.8.8 to be the most recent version, but you can always check MavenCentral for updated versions.

If you are using Maven as your build tool, you can simply paste the following code into your *pom.xml* file's <dependencies> section.

```
<dependency>
   <groupId>com.google.code.gson</groupId>
   <artifactId>gson</artifactId>
   <version>2.8.8</version>
</dependency>
```

If you are using Gradle as your build tool, you can use this snippet instead,

placing it into the dependencies section of the *build.gradle* file:
implementation 'com.google.code.gson:gson:2.8.8'

Default serialization

Let's say we are building a program for a chain of music stores, and one of the many pieces of information we need to keep track of is guitar brands.

So, we'll create a data object containing some basic information about guitar brands: when they were founded, their name, the country they were founded in, and a list of a few notable artists/bands who use this brand.

```
public class GuitarBrand {
    Date dateFounded;
    String name;
    String country;
    List<String> artistsUsedBy;

    // getters, setters, contructor
}
```

Gson can handle dates and lists without issue.

```
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
GuitarBrand espGuitar = new GuitarBrand(sdf.parse("03-08-1975"),
                                "ESP Guitars",
                                "Japan",
                                Arrays.asList("Metallica",
                                "Children of Bodom",
                                "The Rolling Stones", "Rammstein"));
String espGuitarJson = new Gson().toJson(espGuitar);
```

Viewing the results

One point worth mentioning is that the JSON standard technically doesn't care about the ordering of JSON elements, because they are backed by a Map under the hood and so are accessed by key, not by their index.

Some JSON libraries will change the order a bit to optimize the underlying map, however, Gson was specifically designed to keep the order intact. Here's our JSON:

{"dateFounded":"Aug 3, 1975, 12:00:00 AM","name":"ESP Guitars","country":
"Japan",
"artistsUsedBy":["Metallica","Children of Bodom","The Rolling Stones",
"Rammstein"]}

Everything looks the way it should! The date is expressed as a String using Gson's default date formatting.

This can of course be customized if you need it to be expressed in a particular format.

The List we passed in to keep track of the artists is represented by a JsonArray.

By default, Gson will not serialize object properties that have a value of null. You must explicitly set Gson to force serialization of null values.

Default deserialization

GuitarBrand espGuitar = new Gson().fromJson(jsonInput, GuitarBrand.class);

If we store or transmit data in JSON, we are going to need a way to read the data back into our Java program as an object.