FREE eBook

LEARNING

gson

Free unaffiliated eBook created from
Stack Overflow contributors.

#gson

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gson

It is an unofficial and free gson ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gson.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with gson

Remarks

Gson is an Open Source Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object.

Goals for Gson

- Provide easy to use mechanisms like toString() and constructor (factory method) to convert Java to JSON and vice-versa
- Allow pre-existing unmodifiable objects to be converted to and from JSON
- Allow custom representations for objects
- Support arbitrarily complex objects
- Generate compact and readable JSON output

Source code of Gson is available on Github.

User guide

Examples

Installation

In order to use Gson you have to include it in your project. You can do this by adding the following dependency of the Gson version available in Maven Central:

Maven

Add to pom.xml

```
<dependencies>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.0</version>
      <scope>compile</scope>
    </dependency>
</dependencies>
```

Gradle:

Add to build.gradle

```
compile 'com.google.code.gson:gson:2.8.0'
```

Serialization and deserialization

```
Gson gson = new Gson(); //Create a Gson object
MyType target = new MyType(); //This is the object you want to convert to JSON
String json = gson.toJson(target); // serializes target to Json
MyType target2 = gson.fromJson(json, MyType.class); // deserializes json into target2
```

Arrays

JSON:

```
[
  {
    "id": 8484,
    "name": "David",
    "height": 173.2,
    "weight": 75.42
  },
  {
    "id": 8485,
    "name": "Ronald",
    "height": 183.73,
    "weight": 83.1
  }
]
```

Person.java

```
public class Person {
    public int id;
    public String name;
    public double height;
    public double weight;

    @Override
    public String toString() {
        return "[ id: " + String.valueOf(id) + ", name: " + name + ", height: " +
String.valueOf(height) + ", weight: " + String.valueOf(weight) + " ]";
    }
}
```

Usage:

```
Gson gson = new Gson();
Person[] persons = gson.fromJson(json, Person[].class);
for(Person person : persons)
    System.out.println(person.toString());
```

Output:

```
[ id: 8484, name: David, height: 173.2, weight: 75.42 ]
```

```
[ id: 8485, name: Ronald, height: 183.73, weight: 83.1 ]
```

Simple Example

The Gson library provides `Gson.class` which handles all conversion between Java and JSON objects. An instance of this class can be created by invoking default constructor. You usually would like to have one Gson instance for the most part of operations in your program.

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

First, we need to create class of our object with which we will be working with

```
class Person {
   public String name;
   public int age;

   public Person(String name, int age){
       this.name = name;
       this.age = age;
   }
}
```

Gson class provides methods `toJson` and `fromJson` which are the main entry points for JSON and java objects

Let's try to convert java object to JSON and back to java object

```
Person person = new Person("Jason", 29);
//using gson object which we created earlier
String json = gson.toJson(person);
System.out.println(json);
//Outputs: {"name": "Jason", "age": 29}
```

And now back again

```
String json = "{\"name\": \"Jason\", \"age\": 29}";
Person person = gson.fromJson(json, Person.class);
System.out.println(person.age + "yo " + person.name + " walks into a bar");
//Outputs "29 yo Jason walks into a bar"
```

Convert String to JsonObject without POJO

```
String jsonStr = "{\"name\" : \"Abcd\", \"greeting\": \"Hello\", }"; //Sample Json String

Gson gson = new Gson(); // Creates new instance of Gson
JsonElement element = gson.fromJson (jsonStr, JsonElement.class); //Converts the json string
to JsonElement without POJO
JsonObject jsonObj = element.getAsJsonObject(); //Converting JsonElement to JsonObject

String name = jsonObj.get("name").getAsString(); //To fetch the values from json object
String greeting = jsonObj.get("greeting").getAsString();
```

Using GSON with inheritance

GSON does not support inheritance our of the box. Let's say we have the following class hierarchy:

```
public class BaseClass {
    int a;

    public int getInt() {
        return a;
   }
}

public class DerivedClass1 extends BaseClass {
     int b;

     @Override
     public int getInt() {
         return b;
     }
 }

public class DerivedClass2 extends BaseClass {
    int c;

    @Override
    public int getInt() {
        return c;
    }
}
```

And now we want to serialize an instance of `DerivedClass1` to a json string

```
DerivedClass1 derivedClass1 = new DerivedClass1();
derivedClass1.b = 5;
derivedClass1.a = 10;

Gson gson = new Gson();
String derivedClass1Json = gson.toJson(derivedClass1);
```

Now, in another place, we receive this json string and want to deserialize it - but in compile time we only know it is supposed to be an instance of `BaseClass`:

```
BaseClass maybeDerivedClass1 = gson.fromJson(derivedClass1Json, BaseClass.class);
System.out.println(maybeDerivedClass1.getInt());
```

But GSON does not know `derivedClass1Json` was originally an instance of `DerivedClass1`, so this will print out 10.

How to solve this?

You need to build your own `JsonDeserializer`, that handles such cases. The solution is not perfectly clean, but I could not come up with a better one.

First, add the following field to your base class

```
@SerializedName("type")
private String typeName;
```

And initialize it in the base class constructor

```
public BaseClass() {
    typeName = getClass().getName();
}
```

Now add the following class:

```
public class JsonDeserializerWithInheritance<T> implements JsonDeserializer<T> {

 @Override
 public T deserialize(
     JsonElement json, Type typeOfT, JsonDeserializationContext context)
     throws JsonParseException {
     JsonObject jsonObject = json.getAsJsonObject();
     JsonPrimitive classNamePrimitive = (JsonPrimitive) jsonObject.get("type");

     String className = classNamePrimitive.getAsString();

     Class<?> clazz;
     try {
     clazz = Class.forName(className);
     } catch (ClassNotFoundException e) {
     throw new JsonParseException(e.getMessage());
     }
     return context.deserialize(jsonObject, clazz);
 }
}
```

All there is left to do is hook everything up -

```
GsonBuilder builder = new GsonBuilder();
 builder
 .registerTypeAdapter(BaseClass.class, new JsonDeserializerWithInheritance<BaseClass>());
 Gson gson = builder.create();
```

And now, running the following code-

```
 DerivedClass1 derivedClass1 = new DerivedClass1();
 derivedClass1.b = 5;
 derivedClass1.a = 10;
 String derivedClass1Json = gson.toJson(derivedClass1);

 BaseClass maybeDerivedClass1 = gson.fromJson(derivedClass1Json, BaseClass.class);
 System.out.println(maybeDerivedClass1.getInt());
```

Will print out 5.

Read Getting started with gson online: https://riptutorial.com/gson/topic/4804/getting-started-with-

gson

Chapter 2: Using Gson with JAX-RS (RESTful web services)

Examples

JAX-RS provider to use Gson

This is a custom JAX-RS `@Provider` to use Gson as the JSON parser. The example also shows how to use custom Java 8 date/time converters.

```
@Provider
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class JerseyServerGson
        implements MessageBodyWriter<Object>, MessageBodyReader<Object>
{
  @Override
  public boolean isReadable(Class<?> type,
                            Type genericType,
                            Annotation[] annotations,
                            MediaType mediaType)
  {
    return true;
  }

  @Override
  public Object readFrom(Class<Object> type,
                         Type genericType,
                         Annotation[] annotations,
                         MediaType mediaType,
                         MultivaluedMap<String, String> httpHeaders,
                         InputStream entityStream)
        throws IOException, WebApplicationException
  {
    try ( InputStreamReader input =
                new InputStreamReader(entityStream, "UTF-8") ) {
      Gson gson = getGson();
      return gson.fromJson(input, genericType);
    }
  }

  @NotNull
  private Gson getGson() {
    return new GsonBuilder()
            .registerTypeAdapter(LocalDateTime.class,
                                 new AdapterLocalDateTime().nullSafe())
            .registerTypeAdapter(LocalDate.class,
                                 new AdapterLocalDate().nullSafe())
            .setPrettyPrinting()
            .serializeNulls()
            .create();
  }

  @Override
```

```
  public boolean isWriteable(Class<?> type,
                             Type genericType,
                             Annotation[] annotations,
                             MediaType mediaType)
  {
    return true;
  }

  @Override
  public long getSize(Object o,
                      Class<?> type,
                      Type genericType,
                      Annotation[] annotations,
                      MediaType mediaType)
  {
    // Deprecated and ignored in Jersey 2
    return -1;
  }

  @Override
  public void writeTo(Object o,
                      Class<?> type,
                      Type genericType,
                      Annotation[] annotations,
                      MediaType mediaType,
                      MultivaluedMap<String, Object> httpHeaders,
                      OutputStream entityStream)
          throws IOException, WebApplicationException
  {
    try ( OutputStreamWriter writer =
                  new OutputStreamWriter(entityStream, "UTF-8") ) {
      getGson().toJson(o, genericType, writer);
    }
  }
}
```

Read Using Gson with JAX-RS (RESTful web services) online:
https://riptutorial.com/gson/topic/4893/using-gson-with-jax-rs--restful-web-services-

Credits

S. No	Chapters	Contributors
1	Getting started with gson	Community, Daniil Dubrovsky, Derlin, Egor Neliuba, Ginandi, James, Maverick, pr0gramist, Uttam
2	Using Gson with JAX-RS (RESTful web services)	Derlin, sargue