

# Programmieren II

## JavaScript-Objekt-Verarbeitung mit GSON



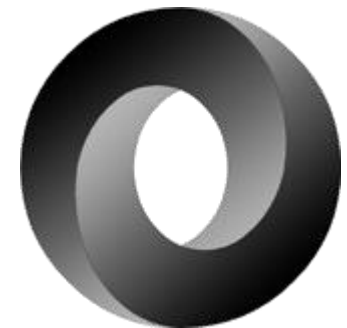
Heusch --  
Ratz --

Institut für Automation und angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

# Motivation

- In vielen Web-Anwendungen (HTML5) werden JavaScript-Objekte (JSON = **J**ava**S**cript **O**bject **N**otation) verarbeitet
- Java-Anwendungen können mit Hilfe von Zusatz-Bibliotheken solche JavaScript-Objekte verarbeiten, z.B.
  - Java-Objekte nach JSON konvertieren („Serialisierung“)
  - JSON in Java-Objekte umwandeln („Deserialisierung“)
- Eine solche Bibliothek ist GSON



JSON-Logo

# Java und JavaScript

- Trotz der Ähnlichkeit ihrer Namen besteht zwischen **Java** und **JavaScript** ein grundlegender Unterschied: Zwar sind beide objektorientiert, jedoch handelt es sich bei Java um eine **Programmiersprache**, während JavaScript eine sog. **Skriptsprache** ist.
- JavaScript wird meist in Web-Browsern verwendet und dient der Programmierung dynamischer Web-Anwendungen.  
Dabei unterliegt es aus Gründen der Sicherheit erheblichen Beschränkungen, z.B. was den Zugriff auf lokale Dateien oder Netzwerkverbindungen betrifft.

# JSON – Syntax (1)

- JSON bildet eine Untermenge von JavaScript und dient der Darstellung von JavaScript-Objekten
- JSON wird häufig zum Datenaustausch in Web-Systemen verwendet und löste hier an vielen Stellen XML ab:
  - es ist „kürzer“ (kompakter) als XML
  - es ist einfacher zu lesen und zu schreiben
  - es kennt Arrays
  - es kann direkt in JavaScript verarbeitet werden (es IST JavaScript!)

## JSON – Syntax (2)

- JSON besteht aus Schlüssel-Wert-Paaren.  
Der Schlüssel (Attributname) wird in Anführungszeichen gesetzt, der Wert (hier eine Zeichenkette) folgt nach dem Doppelpunkt

```
"prename": "Donald"
```

- JSON kennt folgende Wert-Typen:
  - Zahl (number) (integer oder floating point)
  - Zeichenkette (string) (in doppelten Anführungszeichen)
  - Logischer Wert (boolean) (`true` or `false`)
  - Feld (array) (in eckigen Klammern `[]`)
  - Objekt (object) (in geschweiften Klammern `{}`)
  - `null`

## JSON – Syntax (3)

- Beispiele für die verschiedenen *Wert*-Typen:

```
"prename": "Donald"
```

*string*

```
"surname": "Duck"
```

*string*

```
"birthyear": 1920
```

*number*

```
"size": 1.10
```

*number*

```
"liaised": true
```

*boolean*

```
"partner": { "prename": "Daisy", "surname": "Duck" }
```

*object*

```
"nephews": [ { "name": "Tick" }, { "name": "Trick" },  
              { "name": "Track" } ]
```

*array*

```
"job": null
```

*null*

## JSON – Syntax (4)

- Auch einfache Daten können (ohne Attributnamen) in JSON dargestellt werden, z.B.

"Hello World"

*string*

3.1416


*number*

42

*number*

## JSON – Syntax (5)

- JSON-Objekte werden in geschweiften Klammern geschrieben und können mehrere Schlüssel-Wert-Paare (als Attribute) enthalten.  
Mehrere Attribute werden durch Kommas getrennt.



```
{
  "prename": "Donald",
  "surname": "Duck",
  "birthyear": 1920,
  "size": 1.10,
  "liaised": true,
  "partner": { "prename": "Daisy", "surname": "Duck" },
  "nephews": [ { "name": "Tick" }, { "name": "Trick" },
               { "name": "Track" } ],
  "job": null
}
```



## JSON – Syntax (6)

- JSON-Arrays werden in eckigen Klammern geschrieben. Sie können mehrere Objekte enthalten, die wiederum durch Kommas voneinander getrennt werden:

```
"ducks": [  
  {"prename": "Donald", "surname": "Duck"},  
  {"prename": "Daisy", "surname": "Duck"},  
  {"name": "Phantomias", "alias": "Donald"}  
]
```



*Anmerkung: Die Objekte haben dabei keinen „Typ“, d.h. sie können unterschiedliche Attribute haben.*

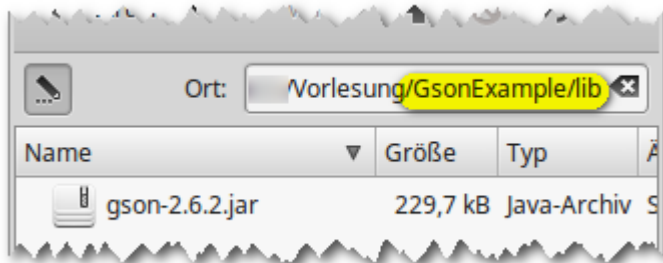
# GSON-Download und Installation

- GSON ist eine Bibliothek von Google zum Arbeiten mit JSON-Objekten in Java. Sie ist unter der Apache License 2.0 zur Verwendung in eigenen Programmen verfügbar.
- Quellen:  
<https://github.com/google/gson> (Quellcode)  
<https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.7/>  
(JAR-Bibliothek, 2.8.7 momentan aktuelle Version)
- Die Datei `gson-<version>.jar` herunterladen und als Bibliothek dem Projekt hinzugefügen.

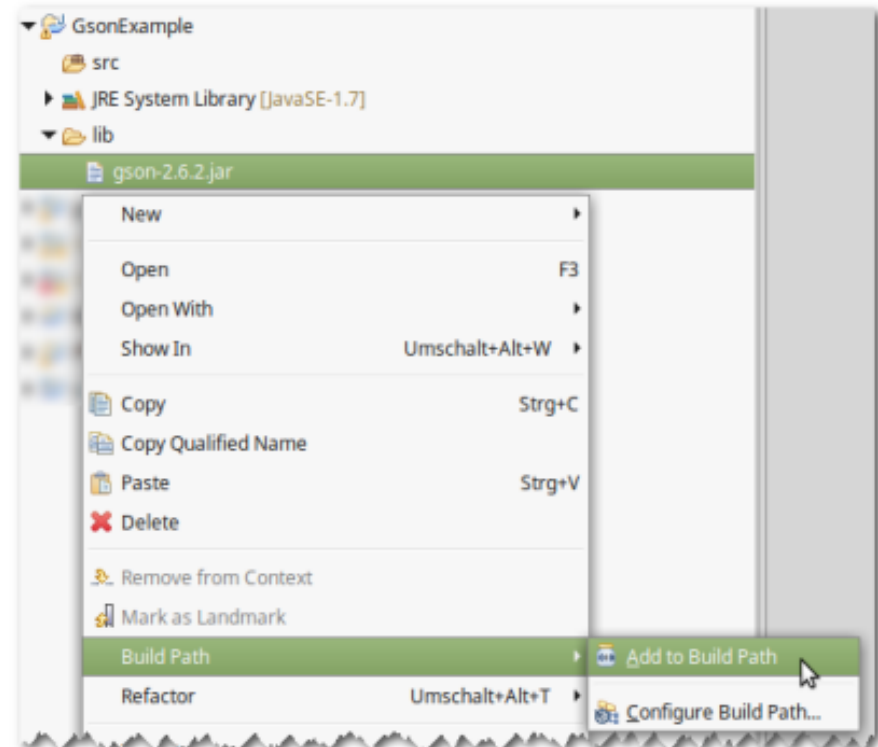
*Anmerkung: Einbindung mit Maven machen wir nächstes Mal ☺*

# Bibliothek einbinden in Eclipse (1)

- Mit dem Dateibrowser ins Projekt-Verzeichnis navigieren (hier: Projekt „GsonExample“)
- Neuen Ordner „lib“ erzeugen und JAR-Datei `gson-<version>.jar` hineinkopieren

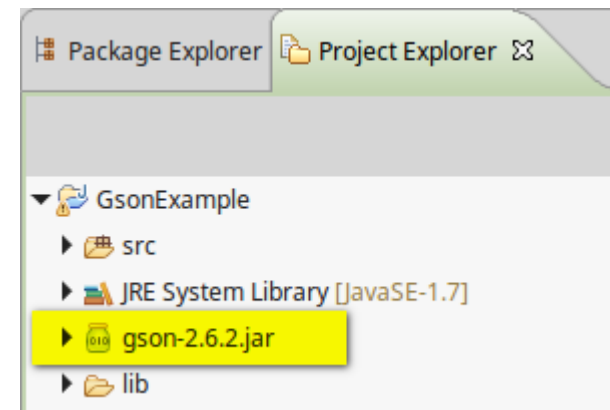
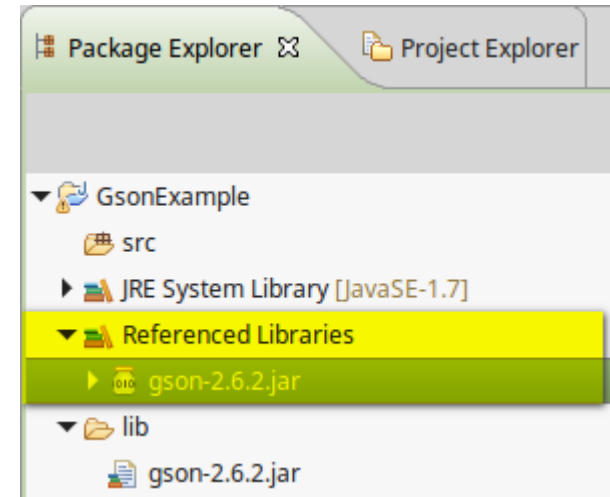


- Rechtsklick in Eclipse auf JAR-Datei im Ordner „lib“
- Build Path → Add to Build Path



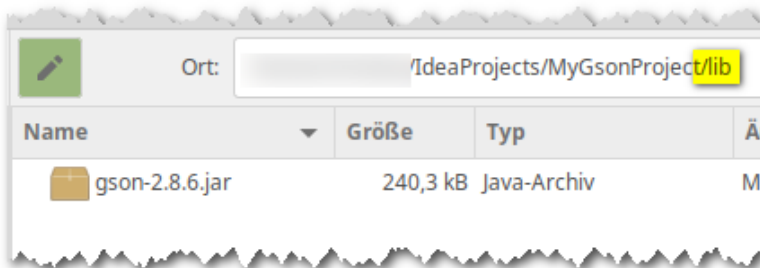
## Bibliothek einbinden in Eclipse (2)

- Im *Package Explorer* erscheint nun der Punkt „References Libraries“ mit dem Eintrag `gson-<version>.jar`
- Im *Project Explorer* erscheint nun `gson-<version>.jar` als weiterer Ressourcen-Eintrag.

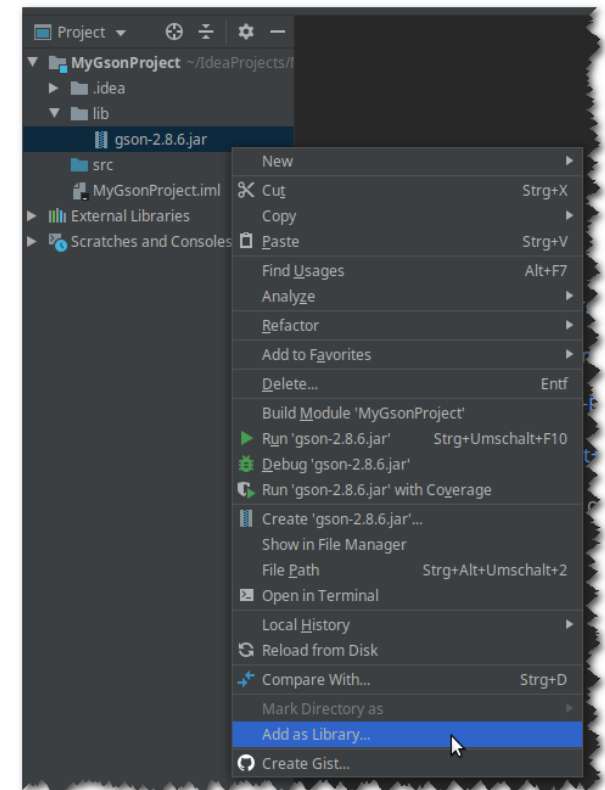


# Bibliothek einbinden in IntelliJ (1)

- Mit dem Dateibrowser ins Projekt-Verzeichnis navigieren (hier: Projekt „MyGsonProject“)
- Neuen Ordner „lib“ erzeugen und JAR-Datei gson-<version>.jar hineinkopieren

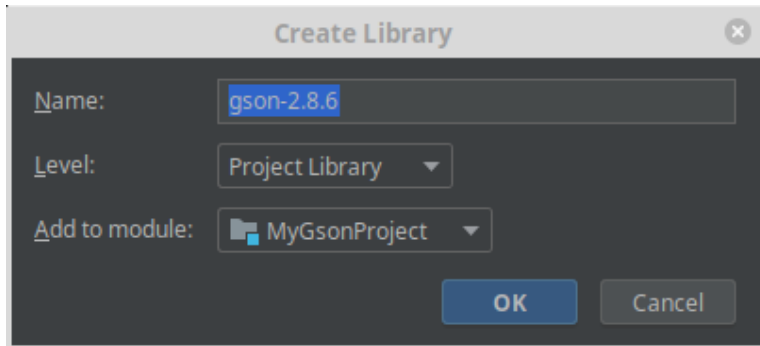


- Rechtsklick auf JAR-Datei im Ordner „lib“
- „Add as Library...“

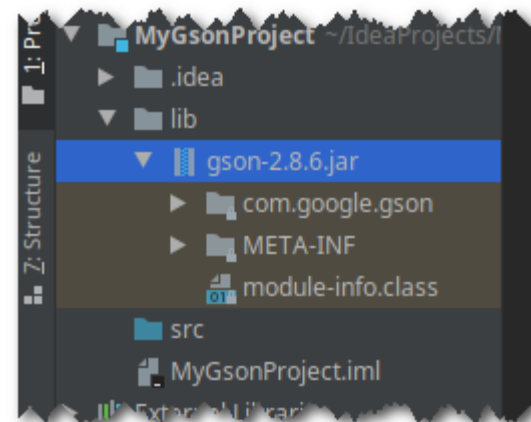


## Bibliothek einbinden in IntelliJ (2)

- Den folgenden Dialog bestätigen



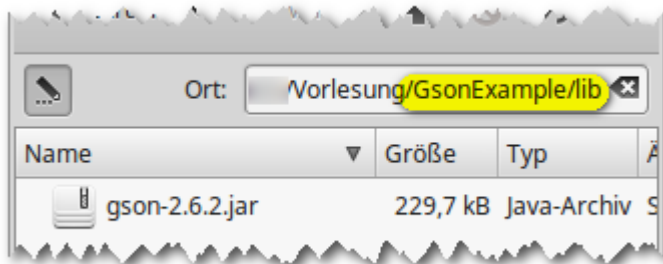
- Die Datei `gson-<version>.jar` ist nun „ausklappbar“ und als Bibliothek eingebunden



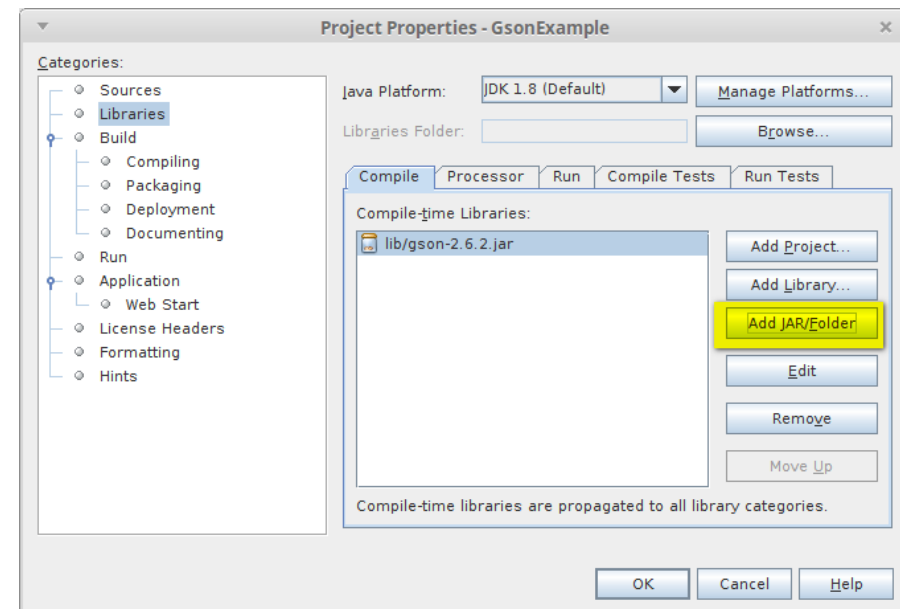
IntelliJ IDEA

# Bibliothek einbinden in Netbeans (1)

- Mit dem Dateibrowser ins Projekt-Verzeichnis navigieren (hier: Projekt „GsonExample“)
- Neuen Ordner „lib“ erzeugen und JAR-Datei `gson-<version>.jar` hineinkopieren

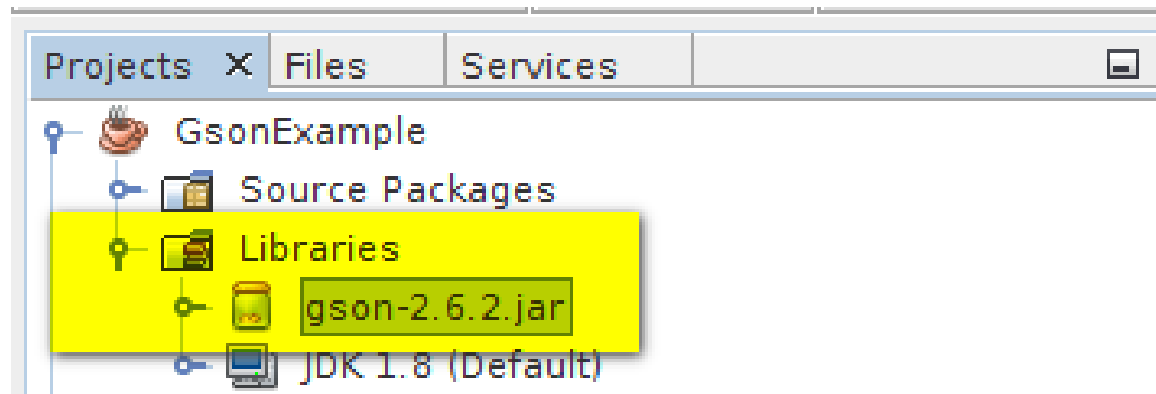


- Projekt-Properties öffnen
- Unter „Libraries“ per „Add JAR/Folder“ hinzufügen



## Bibliothek einbinden in Netbeans (2)

- Im Project-Browser erscheint nun unter „Libraries“ der Eintrag `gson-<version>.jar`





# Einstieg in GSON

- Die meisten notwendigen Klassen (insbesondere `Gson`) befinden sich im Package `com.google.gson`
- `Gson` muss instanziiert werden und bieten dann Methoden zum Wandeln von Java-Objekten nach JSON und umgekehrt:

```
Gson gson = new Gson();  
gson.fromJson( /* .. */ );           // JSON nach Java  
gson.toJson( /* .. */ );             // Java nach JSON
```

# Einfache Beispiele zur „Serialisierung“

- Die Methoden `.toJson( ... )` liefern jeweils einen `String` zurück – oder haben keinen Rückgabewert, dann wird das Ergebnis in den übergebenen `Writer` (`Appendable` oder `JsonWriter` geschrieben)

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

```
System.out.println(gson.toJson(1));
```

```
System.out.println(gson.toJson("abcd"));
```

```
System.out.println(gson.toJson(new Long(10)));
```

```
int[] values = { 1 };
```

```
System.out.println(gson.toJson(values));
```

Ausgabe

1

"abcd"

10

[1]

```

toJson(JsonElement jsonElement) : String - Gson
toJson(Object src) : String - Gson
toJson(JsonElement jsonElement, Appendable writer) : void - Gson
toJson(JsonElement jsonElement, JsonWriter writer) : void - Gson
toJson(Object src, Appendable writer) : void - Gson
toJson(Object src, Type typeOfSrc) : String - Gson
toJson(Object src, Type typeOfSrc, Appendable writer) : void - Gson
toJson(Object src, Type typeOfSrc, JsonWriter writer) : void - Gson
toJsonTree(Object src) : JsonElement - Gson
toJsonTree(Object src, Type typeOfSrc) : JsonElement - Gson

```

# Einfache Beispiele zur „Deserialisierung“

- Die Methoden `.fromJson( ... )` liefern jeweils Objekte Zurück.

Der Typ des Rückgabewertes ist generisch.

Die zugehörige Klasse wird als zweiter Parameter übergeben.

```
Gson gson = new Gson();  
int someint = gson.fromJson("1", int.class);  
Integer someInteger = gson.fromJson("1", Integer.class);  
Long someLong = gson.fromJson("1", Long.class);  
Boolean someBoolean = gson.fromJson("false", Boolean.class);  
String str = gson.fromJson("\"abc\"", String.class);
```

```
● fromJson(JsonElement json, Class<T> classOfT) : T - Gson  
● fromJson(JsonElement json, Type typeOfT) : T - Gson  
● fromJson(JsonReader reader, Type typeOfT) : T - Gson  
● fromJson(Reader json, Class<T> classOfT) : T - Gson  
● fromJson(Reader json, Type typeOfT) : T - Gson  
● fromJson(String json, Class<T> classOfT) : T - Gson  
● fromJson(String json, Type typeOfT) : T - Gson
```

# Ein etwas komplexeres Beispiel

<http://maps.googleapis.com/maps/api/geocode/json?address=karlsruhe>

Alternativ: <https://www.iai.kit.edu/javavl/data/static/karlsruhe.json>



```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Karlsruhe",
          "short_name" : "Karlsruhe",
          "types" : [ "locality", "political" ]
        },
        ...
      ],
      "formatted_address" : "Karlsruhe, Deutschland",
      "geometry" : {
        ...
        "location" : {
          "lat" : 49.0068901,
          "lng" : 8.4036527
        },
        "location_type" : "APPROXIMATE"
      },
      ...
    },
    {
      "types" : [ "locality", "political" ]
    }
  ],
  "status" : "OK"
}
```

Ausschnitt



```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Karlsruhe",
          "short_name" : "Karlsruhe",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "Karlsruhe",
          "short_name" : "KA",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "Baden-Württemberg",
          "short_name" : "BW",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "Deutschland",
          "short_name" : "DE",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "Karlsruhe, Deutschland",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        },
        "location" : {
          "lat" : 49.0068901,
          "lng" : 8.4036527
        },
        "location_type" : "APPROXIMATE",
        "viewport" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        }
      },
      "place_id" : "ChIJCXjgokgG10cRF-63THNV_LY",
      "types" : [ "locality", "political" ]
    },
    {
      "status" : "OK"
    }
  ]
}
```

komplettes Dokument

# Beispiel Deserialisierung (1)

- **Aufgabe:**  
Schreiben Sie Java-Klassen,  
die zu diesem Beispiel passen!
- Kandidaten hierfür sind:
  - results (das gesamte Ergebnis)
  - address\_components
  - geometry
  - bounds
  - location
  - viewport
  - northeast
  - southwest



```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Karlsruhe",
          "short_name" : "Karlsruhe",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "Karlsruhe",
          "short_name" : "KA",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "Baden-Württemberg",
          "short_name" : "BW",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "Deutschland",
          "short_name" : "DE",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "Karlsruhe, Deutschland",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        },
        "location" : {
          "lat" : 49.0068901,
          "lng" : 8.4036527
        },
        "location_type" : "APPROXIMATE",
        "viewport" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        }
      },
      "place_id" : "ChIJCXjgokgG10cRF-63THNV_LY",
      "types" : [ "locality", "political" ]
    }
  ],
  "status" : "OK"
}
```

# Beispiel Deserialisierung (2)

- **Beobachtung:**  
Auf oberster Ebene haben wir ein Objekt mit den Attributen `results` und `status`.
  - `results` ist offenbar ein Array von Objekten
  - `status` ist offenbar ein einzelner String
- Wir benötigen also eine Klasse mit genau diesen Attributen!



```
{
  "results": [
    {
      "address_components": [
        {
          "long_name": "Karlsruhe",
          "short_name": "Karlsruhe",
          "types": [ "locality", "political" ]
        },
        {
          "long_name": "Karlsruhe",
          "short_name": "KA",
          "types": [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name": "Baden-Württemberg",
          "short_name": "BW",
          "types": [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name": "Deutschland",
          "short_name": "DE",
          "types": [ "country", "political" ]
        }
      ],
      "formatted_address": "Karlsruhe, Deutschland",
      "geometry": {
        "bounds": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.541658099999999
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096
          }
        },
        "location": {
          "lat": 49.0068901,
          "lng": 8.4036527
        },
        "location_type": "APPROXIMATE",
        "viewport": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.541658099999999
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096
          }
        }
      },
      "place_id": "ChIJCXjgokgG10cRF-63THNV_LY",
      "types": [ "locality", "political" ]
    }
  ],
  "status": "OK"
}
```

## Beispiel Deserialisierung (3)

- JSON-Arrays können durch GSON entweder in Java-Arrays oder in Java-Collections (z.B. `List`) umgewandelt werden.  
`Collection`-Objekte und Arrays werden bei Bedarf durch GSON instanziiert
- Das Beispiel zeigt beide Möglichkeiten für `GeocodingResult`:

### Implementierung mit `List`

```
//import java.util.ArrayList;
import java.util.List;

public class GeocodingResult {
    public List<Result> results;
    // = new ArrayList<Result>();
    // nicht unbedingt notwendig!

    public String status;
}
```

### Implementierung mit einem Array

```
public class GeocodingResult {
    public Result[] results;
    public String status;
}
```

# Beispiel Deserialisierung (4)

- **Beobachtung:**  
Unterhalb von `results` finden wir Objekte mit den folgenden Attributen:

- **address\_components**  
Array von Objekten

- **formatted\_address**  
String

- **geometry**  
Objekt

- **place\_id**  
String

- **types**  
Array von Strings



```
{
  "results": [
    {
      "address_components": [
        {
          "long_name": "Karlsruhe",
          "short_name": "Karlsruhe",
          "types": [ "locality", "political" ]
        },
        {
          "long_name": "Karlsruhe",
          "short_name": "KA",
          "types": [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name": "Baden-Württemberg",
          "short_name": "BW",
          "types": [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name": "Deutschland",
          "short_name": "DE",
          "types": [ "country", "political" ]
        }
      ],
      "formatted_address": "Karlsruhe, Deutschland",
      "geometry": {
        "bounds": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.541658099999999
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096
          }
        },
        "location": {
          "lat": 49.0068901,
          "lng": 8.4036527
        },
        "location_type": "APPROXIMATE",
        "viewport": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.541658099999999
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096
          }
        }
      },
      "place_id": "ChIJCXjgokgG10cRF-63THNV_LY",
      "types": [ "locality", "political" ]
    }
  ],
  "status": "OK"
}
```



# Beispiel Deserialisierung (5)

- Wir fassen diese Eigenschaften in einer Klasse **Result** zusammen.
  - Den Attributen geben wir die entsprechenden Namen
  - Zu den Datentypen kommen wir auf den folgenden Folien

```
import java.util.List;

public class Result {

    public List<AddressComponent> address_components;
    public String formatted_address;
    public Geometry geometry;
    public String place_id;
    public String[] types;

}
```



```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Karlsruhe",
          "short_name" : "Karlsruhe",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "Karlsruhe",
          "short_name" : "KA",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "Baden-Württemberg",
          "short_name" : "BW",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "Deutschland",
          "short_name" : "DE",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "Karlsruhe, Deutschland",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        },
        "location" : {
          "lat" : 49.0068901,
          "lng" : 8.4036527
        },
        "location_type" : "APPROXIMATE",
        "viewport" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        }
      },
      "place_id" : "ChIJCXjgokgG10cRF-63THNV_LY",
      "types" : [ "locality", "political" ]
    },
    {
      "status" : "OK"
    }
  ]
}
```

## Beispiel Deserialisierung (6)

- Für `address_components` fällt auf, dass alle Objekte im Array dieselben drei Attribute haben. Hierfür bilden wir eine neue Klasse, z.B. **AddressComponent**
- Die Attributnamen entsprechen denen der JSON-Notation




```
"address_components" : [  
  {  
    "long_name" : "Karlsruhe",  
    "short_name" : "Karlsruhe",  
    "types" : [ "locality", "political" ]  
  },  
  {  
    "long_name" : "Karlsruhe",  
    "short_name" : "KA",  
    "types" : [ "administrative_area_level_2", "political" ]  
  },  
  {  
    "long_name" : "Baden-Württemberg",  
    "short_name" : "BW",  
    "types" : [ "administrative_area_level_1", "political" ]  
  },  
  {  
    "long_name" : "Deutschland",  
    "short_name" : "DE",  
    "types" : [ "country", "political" ]  
  }  
],
```

```
public class AddressComponent {  
  
    public String long_name;  
    public String short_name;  
    public String[] types;  
  
}
```

# Beispiel Deserialisierung (7)

- **Beobachtung:**  
Unter `geometry` beobachten wir, dass die Muster von `bounds` und `viewport` übereinstimmen.
- Wir bilden dafür eine gemeinsame Klasse **BBox**
- **Beobachtung:**  
Auch die Muster für `location`, `northeast` und `southwest` passen zueinander.
- Dafür bilden wir die Klasse **LatLon**



```

"geometry" : {
  "bounds" : {
    "northeast" : {
      "lat" : 49.0912684,
      "lng" : 8.541658099999999
    },
    "southwest" : {
      "lat" : 48.9404298,
      "lng" : 8.2774096
    }
  },
  "location" : {
    "lat" : 49.0068901,
    "lng" : 8.4036527
  },
  "location_type" : "APPROXIMATE",
  "viewport" : {
    "northeast" : {
      "lat" : 49.0912684,
      "lng" : 8.541658099999999
    },
    "southwest" : {
      "lat" : 48.9404298,
      "lng" : 8.2774096
    }
  }
},

```

```

public class BBox {
    public LatLon northeast;
    public LatLon southwest;
}

```

```


public class LatLon {
    public double lat;
    public double lng;
}

```

## Beispiel Deserialisierung (8)

- **Beobachtung:**  
geometry selbst besteht aus 4 Attributen. Wir bilden eine Klasse Geometry mit diesen Attributen und den passenden Typen:

- bounds ist vom Typ BBox
- location ist vom Typ LatLon
- location\_type ist ein String
- viewport ist vom Typ BBox



```
"geometry" : {  
  "bounds" : {  
    "northeast" : {  
      "lat" : 49.0912684,  
      "lng" : 8.541658099999999  
    },  
    "southwest" : {  
      "lat" : 48.9404298,  
      "lng" : 8.2774096  
    }  
  },  
  "location" : {  
    "lat" : 49.0068901,  
    "lng" : 8.4036527  
  },  
  "location_type" : "APPROXIMATE",  
  "viewport" : {  
    "northeast" : {  
      "lat" : 49.0912684,  
      "lng" : 8.541658099999999  
    },  
    "southwest" : {  
      "lat" : 48.9404298,  
      "lng" : 8.2774096  
    }  
  }  
},
```

```
public class Geometry {  
    public BBox bounds;  
    public LatLon location;  
    public String location_type;  
    public BBox viewport;  
}
```

## Beispiel Deserialisierung (9)

### ■ Aus den Kandidaten wurden also folgende Java-Klassen:

■ results	→ GeocodingResult
■ address_components	→ AddressComponent
■ geometry	→ Geometry
■ bounds	→ BBox
■ location	→ LatLon
■ viewport	→ BBox
■ northeast	→ LatLon
■ southwest	→ LatLon
	→ Result

### ■ Entscheidend für die Zuordnung sind **Attributnamen** – nicht Klassennamen!

# Beispiel Deserialisierung (10)

- Damit können wir den JSON-Inhalt deserialisieren:

```
import java.io.*;      import java.net.URL;      import com.google.gson.Gson;

public class GeocodingExample {

    public static void main( String[] args ) throws IOException {
        StringBuilder content = new StringBuilder();
        URL url = new URL(
            "https://www.iai.kit.edu/javav1/data/static/karlsruhe.json" );
        try ( BufferedReader br = new BufferedReader(
            new InputStreamReader( url.openConnection().getInputStream() ) ) ) {
            String line;
            while ( (line = br.readLine()) != null ) {
                content.append( line ).append( System.LineSeparator() );
            }
            Gson gson = new Gson();
            GeocodingResult results =
                gson.fromJson( content.toString(), GeocodingResult.class );
            // Nun können wir mit dem results-Objekt arbeiten...
        }
    }
}
```

# Beispiel Deserialisierung (11)

- ...oder noch einfacher (direkt per Reader):

```
import java.io.*;
import java.net.URL;
import com.google.gson.Gson;

public class GeocodingResultWithReader {

    public static void main( String[] args ) throws IOException {
        URL url = new URL(
            "https://www.iai.kit.edu/javav1/data/static/karlsruhe.json" );
        try ( Reader reader =
            new InputStreamReader( url.openConnection().getInputStream() ) ) {

            GeocodingResult results =
                new Gson().fromJson( reader, GeocodingResult.class );

            // Nun koennen wir mit dem results-Objekt arbeiten...

        }
    }
}
```

## Beispiel Deserialisierung (12)

- Der Debugger zeigt, dass die Objekte entsprechend instanziiert und mit Inhalt gefüllt wurden.

▼ results	GeocodingResult (id=28)
▼ results	ArrayList<E> (id=32)
▼ ▲ elementData	Object[10] (id=49)
▼ ▲ [0]	Result (id=57)
▶ address_components	ArrayList<E> (id=67)
▶ formatted_address	"Karlsruhe, Germany" (id=68)
▼ geometry	Geometry (id=69)
▼ bounds	BBox (id=115)
▼ northeast	LatLon (id=124)
lat	49.0912684
lng	8.5416581
▶ southwest	LatLon (id=125)
▶ location	LatLon (id=117)
▶ location_type	"APPROXIMATE" (id=119)
▶ viewport	BBox (id=120)
▶ place_id	"ChIJCXjgokgGI0cRf-63THNV_LY" (id=71)
▼ types	String[2] (id=72)
▶ ▲ [0]	"locality" (id=108)
▶ ▲ [1]	"political" (id=110)



## Beispiel Serialisierung (1)

- Natürlich sollte eine Serialisierung des so erzeugten `results`-Objektes wieder den Ausgangscode ergeben.
- Die Standard-Serialisierung liefert einen kompakten String mit einer langen JSON-Zeile, d.h. ohne Umbrüche.
- Deshalb erzeugen wir uns eine `Gson`-Instanz mit „Pretty-Printing“-Funktion, der formatierten Variante der Ausgabe. Hierbei hilft uns die `GsonBuilder`-Klasse:

```
Gson gsonPp = new GsonBuilder().setPrettyPrinting().create();  
System.out.println(gsonPp.toJson(results));
```

# Beispiel Serialisierung (2)

Bis auf Details der Formatierung ist das Ergebnis identisch mit dem Original.

Auch bei der Serialisierung werden die Attributnamen aus den Java-Klassen verwendet.

```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "Karlsruhe",
          "short_name" : "Karlsruhe",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "Karlsruhe",
          "short_name" : "KA",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "Baden-Württemberg",
          "short_name" : "BW",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "Deutschland",
          "short_name" : "DE",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "Karlsruhe, Deutschland",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        },
        "location" : {
          "lat" : 49.0068901,
          "lng" : 8.4036527
        },
        "location_type" : "APPROXIMATE",
        "viewport" : {
          "northeast" : {
            "lat" : 49.0912684,
            "lng" : 8.541658099999999
          },
          "southwest" : {
            "lat" : 48.9404298,
            "lng" : 8.2774096
          }
        }
      },
      "place_id" : "ChIJCXjgokgG10cRf-63THNV_LY",
      "types" : [ "locality", "political" ]
    },
    {
      "status" : "OK"
    }
  ]
}
```



**Deserialisierung  
und erneute  
Serialisierung**

```
{
  "results": [
    {
      "address_components": [
        {
          "long_name": "Karlsruhe",
          "short_name": "Karlsruhe",
          "types": [ "locality", "political" ]
        },
        {
          "long_name": "Karlsruhe",
          "short_name": "KA",
          "types": [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name": "Baden-Württemberg",
          "short_name": "BW",
          "types": [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name": "Deutschland",
          "short_name": "DE",
          "types": [ "country", "political" ]
        }
      ],
      "formatted_address": "Karlsruhe, Deutschland",
      "geometry": {
        "bounds": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.5416581,
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096,
          }
        },
        "location": {
          "lat": 49.0068901,
          "lng": 8.4036527,
        },
        "location_type": "APPROXIMATE",
        "viewport": {
          "northeast": {
            "lat": 49.0912684,
            "lng": 8.5416581,
          },
          "southwest": {
            "lat": 48.9404298,
            "lng": 8.2774096,
          }
        }
      },
      "place_id": "ChIJCXjgokgG10cRf-63THNV_LY",
      "types": [ "locality", "political" ]
    },
    {
      "status": "OK"
    }
  ]
}
```



*via GSON serialisiert*

## Beispiel Serialisierung (3)

- Änderungen in den Java-Klassen führen automatisch dazu, dass auch der JSON-Output sich verändert, z.B. ein zusätzliches Attribut `alt` in der Klasse `LatLng`:

```
public class LatLng {  
    public double lat;  
    public double lng;  
    public double alt = 123.4;  
}
```

Der entsprechende Inhalt wird bei der Instanziierung der `LatLng`-Objekte erzeugt und kommt nicht aus den deserialisierten JSON-Objekt.

### Ergebnis der Serialisierung

```
"geometry": {  
  "bounds": {  
    "northeast": {  
      "lat": 49.0912684,  
      "lng": 8.5416581,  
      "alt": 123.4  
    },  
    "southwest": {  
      "lat": 48.9404298,  
      "lng": 8.2774096,  
      "alt": 123.4  
    }  
  },  
  "location": {  
    "lat": 49.0068901,  
    "lng": 8.4036527,  
    "alt": 123.4  
  },  
}
```

*Ausschnitt*

# Einstellungen (1)

- `GsonBuilder` erlaubt verschiedene Einstellungen, die sich insb. auf die Serialisierung auswirken, z.B.
  - `.serializeNulls()`  
Per Default werden Felder mit Wert `null` ignoriert.
  - `.excludeFieldsWithModifiers(Modifier.STATIC, Modifier.TRANSIENT, Modifier.VOLATILE)`  
Die Aufzählung darf alle Modifikatoren enthalten (beliebige Anzahl), die ignoriert werden sollen.
  - `.setPrettyPrinting()`  
Formatierte (statt kompakter) Ausgabe
  - `.setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)`  
Formatierung der Attributnamen (hier: Großbuchstabe am Anfang)

## Einstellungen (2)

- Für Attribute können per Annotation abweichende Namen vergeben werden:

```
public class SomeObject {  
    @SerializedName("custom_naming")  
    private final String someField;  
    private final String someOtherField;  
  
    public SomeObject(String a, String b) {  
        this.someField = a;  
        this.someOtherField = b;  
    }  
}
```

- Anwendung:

```
SomeObject someObject = new SomeObject("first", "second");  
Gson gson =  
    new GsonBuilder().setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE).create();  
System.out.println(gson.toJson(someObject));
```

 Ausgabe

```
{"custom_naming":"first","SomeOtherField":"second"}
```

Die Formatierung wirkt sich NICHT auf den explizit vergebenen Namen (@SerializedName) aus

# Einschränkungen

- Es gibt einige Beschränkungen, insb. was die Deserialisierung betrifft, z.B.
  - Generische Datentypen wie in `class Beispiel<T>` müssen explizit angegeben werden
  - Explizite Serialisierung und Deserialisierung:  
`.registerTypeAdapter(AType.class,  
new ATypeAdapter())`  
ebenso mit `Serializer`, `Deserializer`, `InstanceCreator`  
z.B. `implements JsonSerializer<AType>`
- Details dazu unter <https://sites.google.com/site/gson/gson-user-guide>

# Wie das Ganze funktioniert: „Reflection“

- Gson betrachtet die übergebenen Klassen und identifiziert die entsprechenden Attribute per Reflection.

```
import java.lang.reflect.Field;

public class ReflectionExample {
    public static void main(String[] args) {
        Class<?> c = Geometry.class;
        System.out.println("class " + c.getName() + " {");
        for (Field publicField : c.getFields()) {
            String fieldName = publicField.getName();
            String fieldType = publicField.getType().getName();
            System.out.printf("  %s %s;%n", fieldType, fieldName);
        }
        System.out.println("}");
    }
}
```

Auf den Folien bisher nicht zu sehen:  
Alle Klassen befinden sich im Package  
`de.dhbwka.java.slides.sem2.gson`

## Ausgabe

```
class de.dhbwka.java.slides.sem2.gson.Geometry {
  de.dhbwka.java.slides.sem2.gson.BBox bounds;
  de.dhbwka.java.slides.sem2.gson.LatLon location;
  java.lang.String location_type;
  de.dhbwka.java.slides.sem2.gson.BBox viewport;
}
```

# Aufgabe 1

- Erzeugen Sie eine JSON-Repräsentation eines Wettrennens nach Aufgabe „Fahrzeuge“ aus dem Aufgabenblatt „Vererbung (1)“!

*Hinweis: Falls Sie beim „Wettrennen“ die Fahrzeuge in lokalen Variablen gespeichert haben, bauen Sie die Klasse so um, dass diese in einem Attribut abgelegt und die Fahrzeuge z.B. im Konstruktor hinzugefügt werden.*



# Aufgabe 1 (Erwartetes Ergebnis)



```
{
  "fz": [
    {
      "position": 80.0,
      "geschwindigkeit": 20.0,
      "vmax": 30.0
    },
    {
      "position": 0.0,
      "geschwindigkeit": 140.0,
      "vmax": 140.0
    },
    {
      "position": 0.0,
      "geschwindigkeit": 200.0,
      "vmax": 220.0
    },
    {
      "blaulicht": false,
      "position": 0.0,
      "geschwindigkeit": 80.0,
      "vmax": 140.0
    }
  ]
}
```

## Aufgabe 2

### ■ Unter

<http://api.geonames.org/citiesJSON?formatted=true&north=49.5&south=48.5&east=8.8&west=8.2&lang=de&username=demo&style=full>

lassen sich Informationen zu Städten (aus der Wikipedia) als JSON-Datei abrufen.

- Bauen Sie passende Java-Klassen zu diesem Dienst, sortieren Sie das Ergebnis der Abfrage aufsteigend nach der Bevölkerungsgröße („population“) und geben Sie es auf der Konsole aus!

- *Hinweis: Falls die dynamische Abfrage nicht funktioniert gibt es eine statische JSON-Datei unter*

<https://www.iai.kit.edu/javavl/data/static/cities.json>

## Aufgabe 2 (JSON-Input)

```
{ "geonames": [  
  {  
    "fcodeName": "seat of a third-order administrative division",  
    "toponymName": "Mannheim",  
    "countrycode": "DE",  
    "fcl": "P",  
    "fclName": "city, village,...",  
    "name": "Mannheim",  
    "wikipedia": "en.wikipedia.org/wiki/Mannheim",  
    "lng": 8.479547,  
    "fcode": "PPLA3",  
    "geonameId": 2873891,  
    "lat": 49.496706,  
    "population": 307960  
  }  
]
```

