Datenaustausch mit JSON

Datenaustausch

Kodierung von Daten

- Binärformate (.png, .mp3, . . .)
  effizient, aufwändig, nicht menschenlesbar
- Textformate (.txt, .java, . . .):
  menschenlesbar, Aufwand für Ein- und Ausgabe
- generische Formate (.xml, .json, . . .):
  Datenaustausch, implementiert in Bibliotheken

JSON (JavaScript Object Notation)

- einfaches textbasiertes Datenaustauschformat
- menschenlesbar
- standardisiert in RFC 4627

JSON

- “Objekte” mit `Attribut:Wert`-Zuordnungen
- Leerzeichen außerhalb von Strings nicht relevant
- Zeilenumbrüche nicht relevant

```
{
  "type" : "node",      "id"   : "363179",
  "lat"  : 48.1408871, "long" : 11.5615991
},
{
  "type" : "way", "id" : " 372802991",
  "nd": ["3763512880", "3763512881", "1545920068"],
  "tags" : {
        "bus" : "yes",
        "name" : "Herkomerplatz",
        "highway" : "platform"
  }
}
```

JSON Werte

- Strings (z.B. `"abc"`, `"say \"hello\""`)
- Zahlen (z.B. `-12`, `12E9`, `12.9`)
- Wahrheitswerte (`true`, `false`)
- Arrays — Reihe von beliebigen Werten in eckigen Klammern
  Beispiele: `[1,2,3,4]` und `[2.4, true, { "key": "val"}]`
- Objekte
  - beliebige Anzahl von Attribut-Wert-Paaren zwischen geschweiften Klammern
  - Reihenfolge der Attribut-Wert-Paare nicht relevant
  - Attribute sind durch Strings benannt

Beispiel: `{"type": "message", "words" : ["Hi", "World!"] }`

Eine genaue Grammatik findet man auf `http://www.json.org`.

Verarbeitung von JSON-Daten

Es gibt viele Bibliotheken für die Ein- und Ausgabe von JSON-Daten:

- `org.json`
  (verfügbar von `https://github.com/stleary/JSON-java`)
- Jackson
- GSON
- ...

Im Folgenden wird `org.json` kurz vorgestellt. Diese Bibliothek müssen Sie ggf. noch aus dem Internet herunterladen.

IDEs bieten dazu auch Komfort-Funktionen an, z.B.
Intellij: *Project Structure* $\rightarrow$ *Libraries* $\rightarrow$ **+** $\rightarrow$ *fromMaven*

org.json

Wichtige Klassen:

- `JSONArray`:
  - Repräsentiereung von JSON-Arrays, z.B.
    `["Hello", "World!"]`
  - Methoden zum Zugriff auf die Arrayeinträge
- `JSONObject`:
  - Repräsentierung von JSON-Objekten, z.B.
    `{"type": "message", "words" : ["Hello", "World!"] }`
  - Methoden zum Zugriff auf die Werte von Attributen
- Für andere Werte (Strings, Zahlen, Wahrheitswerte) werden die Standardklassen von Java verwendet.
- `JSONTokener`:
  Hilfsklasse zum Einlesen z.B. aus einer Datei.

Parsen mit `org.json`

```
{ "type":"way", "nd": ["3763512880","3763512881","1545920068"],
  "tags": { "name":"Herkomerplatz", "highway": "platform" } }
```

```
String s =
JSONObject json = new JSONObject(s);

// Wert des Attributs "type" als String
String t = json.getString("type"); // "way"
JSONArray nd = json.getJSONArray("nd");

// Wert mit Arrayindex 1 als String
String id1 = nd.getString(1);
// Wert mit Arrayindex 1 als Double
double d1 = nd.getDouble(1);
```

Bei Fehlern (kein JSON-Format, angefrages Attribut nicht vorhanden, falscher Typ) wird eine `JSONException` ausgelöst.

Parsen mit org.json

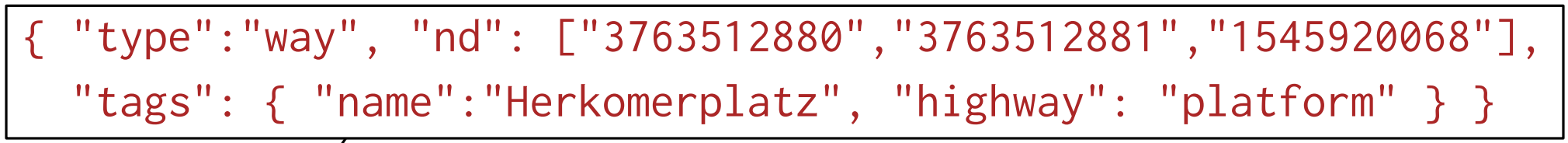
```
{ "type":"way", "nd": ["3763512880","3763512881","1545920068"],
  "tags": { "name":"Herkomerplatz", "highway": "platform" } }
```

```
String s =
JSONObject json = new JSONObject(s);

// geschachteltes Objekt
JSONObject tags = json.getJSONObject("tags");
String n = tags.getString("name");

// Alle Attribute durchlaufen.
// Gibt "type", "nd" und "tags" aus.
for (String k: json.keySet()) {
  System.out.println(k);
}
```

Parsen mit org.json



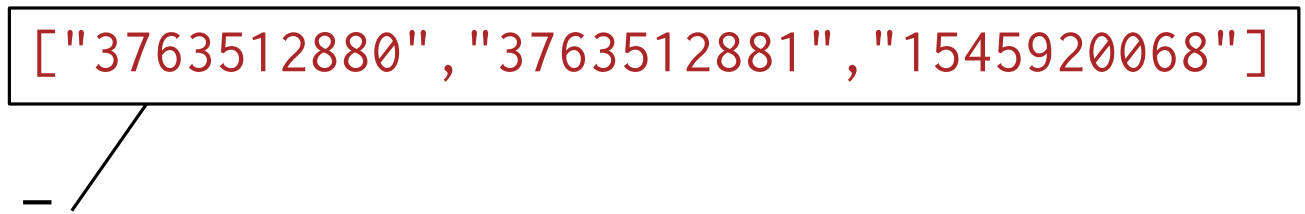

```
String s =
JSONObject json = new JSONObject(s);

// Schlaegt fehl, da der Wert des Attributs "type"
// ein String ist, kein Objekt.
JSONObject o = json.getJSONObject("type");

// Abfragen, ob ein Attribut gesetzt ist
if (!json.has("data")) {
  ... // Behandle Fall, dass Attribut "data" fehlt.
}

// Wert des Attributs "data" oder ein Defaultwert, wenn
// das Attribut nicht existiert
String tag_or_default = json.optString("data", "default")
```

Parsen mit `org.json`




```
String s =
JSONArray arr = new JSONArray(s);

double d1 = nd.getDouble(1);
```

Einlesen von Dateien

Einlesen eines JSON-Objekts aus einer Datei:

```
InputStream is = new FileInputStream("filename.json");
JSONObject json = new JSONObject(new JSONTokener(is));
```

Einlesen eines JSON-Arrays aus einer Datei:

```
InputStream is = new FileInputStream("filename.json");
JSONArray arr = new JSONArray(new JSONTokener(is));
```

Einlesen von Dateien

Wenn eine JSON-Datei zusammen mit den class-Dateien des Programms geliefert werden soll, kann die Datei mit dem Classloader geladen werden.

```
InputStream is =
   ClassLoader.getSystemResourceAsStream(
                        "package/filename.json");
```

Die Datei `filename.json` wird dabei genau wie Klasssen aus der Package `package` geladen, sollte also im gleichen Verzeichnis stehen.

Das funktoniert auch, wenn Klassen und JSON-Datei in einem jar-Archiv verpackt sind.

Ausgabe von JSON-Daten

Beispiel:

```
JSONObject json = new JSONObject();
json.put("type", "node");
json.put("id", "34");
json.put("lat", 31.3);
json.put("long", 12.8);
System.out.println(json);
```

Ausgabe:

```
{"id":"34","type":"node","lat":31.3,"long":12.8}
```

Allgemeine Hinweise

JSON dient nur zum Datenaustausch. Verwenden Sie z.B. `JSONObject` nicht zur Datenspeicherung.

Beim Austausch von Text ist auf die Textkodierung zu achten. Wir verwenden `utf8`.

Behandeln Sie bei der Programmierung alle möglichen Fehlerfälle. JSON-Daten, die aus einer Datei gelesen werden, können nicht als wohlgeformt angenommen werden.