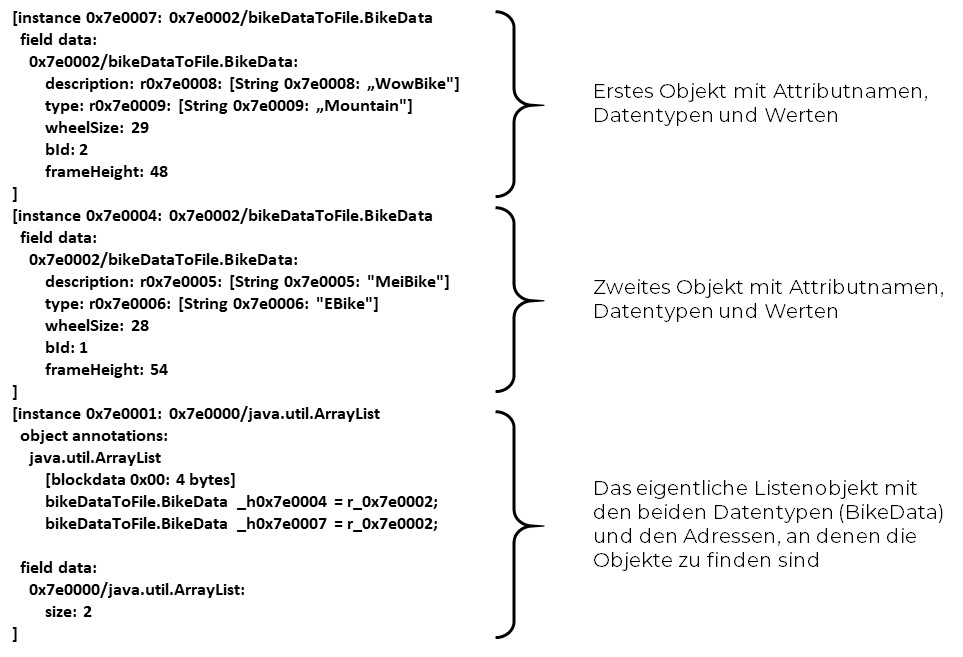
## Objekte in einer Datei speichern

Nachdem wir die Objekte serialisiert in der Datei bikedata.dat gespeichert haben, enthält diese nun die Daten in einer für uns nur teilweise lesbaren Form (die Zeichenketten sind erkennbar). Mit dem externen Tool jdserialize lassen sich die Inhalte etwas aufbereiten:



Man kann gut erkennen, dass keinerlei Informationen zu den Methoden enthalten sind. Diese werden erst beim Lesen wieder dem Objekt hinzugefügt.

Eine andere Möglichkeit bietet sich natürlich, wenn wir uns eine Möglichkeit erschließen, die Daten des Objekts als Zeichenkette in eine Datei zu schreiben.

## Objektdaten als Text mit Textzeichen getrennt in einer Datei speichern

Portabler, weil auch bequem zwischen verschiedenen Rechnerarchitekturen austauschbar und bei Bedarf auch mit dem Editor bearbeitbar sind Textdateien, in denen die eingegebenen Daten in einem regelmäßigen Format abgespeichert werden.

Ein sehr einfaches Datenformat sind Comma-Seperated-Values (csv) - Dateien, wie auch von Spreadsheets (z. B. Excel) gut eingelesen und ausgegeben werden können. Beispiel:

**1, HeiBike, Citybike, 26, 51**

**2, WowBike, Fitness, 28, 56**

Dabei können neben Kommas auch andere Texttrennzeichen zum Einsatz kommen, sie dürfen nur nicht im regulären Text vorkommen.

Um die Ausgabe zu erzeugen, erweitern wir am besten die Klasse BikeData, indem wir die Methode toString() überschreiben und die Objektdaten in der gewünschten Ausgabeform als String zurückgeben.

Beim Einlesen erhalten wir aber keine fertigen Datenelemente, sondern zunächst in Zeichenketten abgelegte Zahlen und Texte. Wie bei der Eingabe über die GUI müssen wir also die gelesenen Zeichenketten in die zugehörigen Daten umwandeln.

Die Textdatei mit den Daten muss nun Zeile für Zeile geschrieben bzw. wieder eingelesen werden. Jede einzelne Zeile entspricht den Daten eines Objekts.

### Die Textdatei im JSON-Format

Das csv-Format hat den Vorteil der großen Flexibilität, geht aber beispielsweise die Beschreibung, der Daten verloren, ist es schwierig, die Daten zu rekonstruieren. Ein Textformat, das sich ebenfalls als universelles Format anbietet, ist das JSON-Format (Beispiel siehe rechts).

**[**

**{**

**"bId": 1.0,**

**"description": "HeiBike",**

**"type": "Citybike",**

**"wheelSize": 26.0,**

**"frameHeight": 51.0**

**},**

**{**

**"bId": 2.0,**

**"description": "WowBike",**

**"type": "Fitness",**

**"wheelSize": 28.0,**

**"frameHeight": 56.0**

**},**

**{**

**"bId": 1,**

**"description": "OuhBike",**

**"type": "Racing",**

**"wheelSize": 28,**

**"frameHeight": 48**

**}**

**]**

Hier sind die Variablennamen als Zeichenkette mit angeben und die Liste kann auch wieder komplett ausgegeben werden. In der nebenstehenden Darstellung ist die Liste in Array-Klammern eingeschlossen, jedes der Objekte in der Liste wird durch geschweifte Klammern begrenzt. Sind in den Objekten wiederum Referenzen auf Objekte enthalten, so finden sich deren Daten in geschweiften Klammern in einer tieferen Ebene. Die so entstehende Text-Datei ist strukturiert, gut lesbar und immer noch mit einem Texteditor editierbar.

Die große Beliebtheit von JSON insbesondere im Netz zum universellen Datenaustausch hat leider zu einer leicht unübersichtlichen Zahl von Implementierungen geführt.

Eine einfach anzuwendende Bibliothek ist Gson (von Google geschrieben und frei verfügbar mit Sourcecode).

Eine einfache Ausgabe kann mit den beiden folgenden Zeilen erzeugt werden:

**Gson gson = new Gson();**

**String json = gson.toJson(object);**

Diese Ausgabe erscheint dann allerdings fortlaufend in einer Zeile. Um die oben gezeigte Struktur zu erzielen, erzeugen wir unser gson-Objekt mit:

**Gson gson = new GsonBuilder().setPrettyPrinting().create();**

Die Ausgabe erfolgt dann wie oben. Mittels Gson wird eine codierte Zeichenkette erzeugt, in der die Zeilenumbrüche und Einrückungen nun enthalten sind. Abschließend kann die Zeichenkette einfach als Zeichenkette geschrieben werden.

Ebenso einfach wie das Schreiben ist auch das Lesen aus der Datei. Zur Umwandlung benötigen wir jedoch den Bauplan der Klasse als 2. Parameter.

**object = gson.fromJson(json, ArrayList.class);**