org.json - creating

Creating JSON using org.json

What is “JSON”?

Data in plain text, formatted in JavaScript syntax for serializing objects and sending between applications.

When do we need to create JSON?

If our Java application needs to send data from to some other application, we can use JSON for the representation of our Java objects.

The goal is to offer the JSON data from our application, so that other applications can read our data and transform it to e.g. Java objects again.

So we need to create the JSON text from our Java objects (like a List of Schedules for instance).

In this simple example, we'll show you where the data could come from (a database), what Java objects we want to send (List<Schedule>, Schedule, School, Substitute, String etc).

Example schedule for substitutes

We have an application which should create JSON about the schedule for substitute teachers.

A the whole schedule is a list of “assignments” or schedules for one substitute teacher, and one such schedule could be in JSON:

```
{
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Rikard"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72 GÖTEBORG"
    }
}
```

Note: The order of the name/value pairs are not important in JSON - it’s like a Map!

Example schedule for substitutes for a given day

Our application will create a JSON file with only the schedules for 2018-01-17, to keep the output manageable.

The data comes from a database with three tables

```
CREATE TABLE substitute(substitute_id integer primary key,
name text);
CREATE TABLE schedule(day datetime, substitute_id integer,
school_id integer);
CREATE TABLE "school"(school_id integer primary key,
school_name text, address text);
```

The data comes from a database - school table

```
CREATE TABLE "school"(school_id integer primary key, school_name text,
address text);

sqlite> select * from school;
1|Yrgo|Lärdomsgatan 3 402 72 GÖTEBORG
2|ITHS|Ebbe Lieberathsgatan 18 c 412 65 Göteborg
3|Jensen|Kruthusgatan 17 411 04 Göteborg
4|Angeredsgymnasiet|Grepgatan 9 424 65 Angered
5|Aniaragymnasiet|Kyrkogatan 46 411 15 Göteborg
6|Bernadottegymnasiet|Skånegatan 18 411 40 Göteborg
7|Bräckegymnasiet|Uppfinnaregatan 1 417 56 Göteborg
8|Burgårdens utbildningscentrum|Skånegatan 20 402 29 Göteborg
```

The data comes from a databases - substitute table

```
CREATE TABLE substitute(substitute_id integer primary key, name text);
```

```
sqlite> select * from substitute;
1|Rikard
2|Henrik
3|Anders
4|Nahid
5|Conny
6|Svante
7|Elisabeth
8|Eva
9|Kristina
10|Bengt
```

The data comes from a database - schedule table

```
CREATE TABLE schedule(day datetime, substitute_id integer, school_id
integer);

sqlite>
select * from schedule;
2018-01-15 08:00:00|1|1
2018-01-16 08:00:00|1|1
2018-01-17 08:00:00|1|1
2018-01-18 08:00:00|1|2
2018-01-16 08:00:00|2|1
2018-01-17 08:00:00|2|1
2018-01-18 08:00:00|2|3
2018-01-16 08:00:00|3|2
2018-01-17 08:00:00|3|4
2018-01-18 08:00:00|3|5
2018-01-16 08:00:00|4|8
```

The data comes from a database - SELECT

```
sqlite> select day, name, school_name,address
        from schedule join substitute
         on schedule.substitute_id=substitute.substitute_id
        join school
         on schedule.school_id = school.school_id
        where day='2018-01-17 08:00:00';
2018-01-17 08:00:00|Rikard|Yrgo|Lärdomsgatan 3 402 72 GÖTEBORG
2018-01-17 08:00:00|Henrik|Yrgo|Lärdomsgatan 3 402 72 GÖTEBORG
2018-01-17 08:00:00|Anders|Angeredsgymnasiet|Grepgatan 9 424 65 Angered
```

That’s going to be our List<Schedule> - three Schedule objects

As you see, we limit ourselves to one single day, 2018-01-17

The data **could** come from a database - fake it!

We'll hard code three Schedule objects for the list, since this is not a lecture on JDBC. In the source code repository, however, you'll find the database if you want to fetch the data from an SQLite3 database.

https://github.com/progund/java-web/tree/master/java-json/org.json/creating_

We have several lectures on JDBC in our wiki if you feel that you want to refresh or learn how to fetch data from a database in Java.

The JSON file - schedule for 2018-01-17

```
[
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Rikard"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72
GÖTEBORG"
    }
  },
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Henrik"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72
GÖTEBORG"
    }
  },
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Anders"
    },
    "school": {
      "school_name": "Angeredsgymnasiet",
      "address": "Grepgatan 9 424 65
Angered"
    }
  }
]
```

This is the File we want to create.

Remember, the order of the name/value pairs in a JSON object are not important, so the result might vary a little.

“school” may come before “substitute” for instance...

Schedule class

```
public class Schedule {
  private String date;
  private Substitute substitute;
  private School school;
  public Schedule(String date, Substitute substitute, School school) {
    this.date = date;
    this.substitute = substitute;
    this.school = school;
  }
  public String date() { return date; }
  public Substitute substitute() { return substitute; }
  public School school() { return school; }
  public String toString() {
    return date + " " + substitute + " " + school;
  }
}
```

Substitute class

```
public class Substitute {
  private String name;
  public Substitute(String name) {
    this.name = name;
  }

  public String name() {
    return name;
  }

  public String toString() {
    return name;
  }
}
```

School class

```
public class School {
  private String name;
  private String address;

  public School(String name, String address) {
    this.name = name;
    this.address = address;
  }

  public String name() { return name; }

  public String address() { return adress; }

  public String toString() {
    return name + " " + address;
  }
}
```

Goal - parse JSON, create List‹Schedule›

```
public class Formatter {
  // stuff which can format one Schedule to a JSONObject
  // and can format a List<Schedule> to a JSONArray of such objects

  // we will not focus on design and architecture, just the JSON code!
}
```

The JSON file we want to create

```
[
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Rikard"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72
GÖTEBORG"
    }
  },
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Henrik"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72
GÖTEBORG"
    }
  },
  {
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Anders"
    },
    "school": {
      "school_name": "Angeredsgymnasiet",
      "address": "Grepgatan 9 424 65
Angered"
    }
  }
]
```

Strategy:
Create a JSONArray
Iterate over each Schedule in List<Schedule>:
create one JSONObject for one Schedule
set Date as a JSON string
set substitute as a JSONObject
set school as a JSONObject
add the JSON Schedule object to the array
return the String with the whole JSONArray

Formatter - create JSON

```
// 1. Create a JSONArray
    JSONArray JSON = new JSONArray();


// 2. Loop over the List<Schedule>

    for (Schedule schedule : schedules) {
      // 3. add a JSONObject with a schedule to the array
      JSON.put(JSONSchedule(schedule)); // You can put a JSONObject in
    }                                   // a JSONArray

// 4. Return the JSON as a Java String
    return JSON.toString(2); // 2 Spaces indentation
  }
```

Formatter - create JSON

```
// 3. add a JSONObject with a schedule to the array
  JSON.put(JSONSchedule(schedule)); // You can put a JSONObject in
                                    // a JSONArray
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    // Create a JSONObject for the Schedule
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    // Add the date JSON string to the schedule JSONObject:
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    // Create the substitute JSONObject:
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    // Add the name JSON string to the substitute JSONObject:
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    // Add the substitute JSONObject to the schedule JSONObject:
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    // Create the school JSONObject:
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    // Add the school_name and address JSON strings to the school JSONObject:
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    // Add the school JSONObject to the Schedule JSONObject:
    JSONSchedule.put("school", JSONSchool);
    return JSONSchedule;
  }
```

Formatter - create JSON

```
// the JSONSchedule method (we wrote it ourselves):
  private static JSONObject JSONSchedule(Schedule schedule) {
    JSONObject JSONSchedule = new JSONObject();
    JSONSchedule.put("date", schedule.date());
    JSONObject JSONSubstitute = new JSONObject();
    JSONSubstitute.put("name", schedule.substitute().name());
    JSONSchedule.put("substitute", JSONSubstitute);
    JSONObject JSONSchool = new JSONObject();
    JSONSchool.put("school_name", schedule.school().name());
    JSONSchool.put("address", schedule.school().address());
    JSONSchedule.put("school", JSONSchool);
    // Return the Schedule JSONObject:
    return JSONSchedule;
  }
```

Formatter - full code

```
private static JSONObject JSONSchedule(Schedule schedule) {
  JSONObject JSONSchedule = new JSONObject();
  JSONSchedule.put("date", schedule.date());
  JSONObject JSONSubstitute = new JSONObject();
  JSONSubstitute.put("name", schedule.substitute().name());
  JSONSchedule.put("substitute", JSONSubstitute);
  JSONObject JSONSchool = new JSONObject();
  JSONSchool.put("school_name", schedule.school().name());
  JSONSchool.put("address", schedule.school().address());
  JSONSchedule.put("school", JSONSchool);
  return JSONSchedule;
}

public static String format(List<Schedule> schedules) {
  JSONArray JSON = new JSONArray();
  for (Schedule schedule : schedules) {
    JSON.put(JSONSchedule(schedule));
  }
  return JSON.toString(2);
}
```

Main class - full code

```
// Import statements - see next slide
public class CreateSchedule {
  public static void main(String[] args) {
    List<Schedule> schedules = Schedules.getSchedules();
    String JSON = Formatter.format(schedules);
    writeFile(JSON);
  }

  public static void writeFile(String JSON) {
    try {
      Path jsonFile = Paths.get("2018-01-17.json");
      Files.write(jsonFile, JSON.getBytes(StandardCharsets.UTF_8));
    } catch (IOException ioe) {
      System.err.println("Exception writing JSON file: " + ioe.getMessage());
    }
  }
}
```

Main class - full code - package and imports

```
package jsonexample.main;

// Stuff for writing the JSON file:
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
// The List<Schedule>
import java.util.List;

// The domain object Schedule, the Formatter and a helper class to get the List<Schedule>:
import jsonexample.domain.Schedule;
import jsonexample.formatter.Formatter;
import jsonexample.data.Schedules; // Doesn’t really use a database
```

Schedules utility class - full code

```
package jsonexample.data;
import java.util.ArrayList;
import java.util.List;
import jsonexample.domain.Schedule;
import jsonexample.domain.Substitute;
import jsonexample.domain.School;

public class Schedules { // Replace with JDBC code as an exercise!
  public static List<Schedule> getSchedules() {
    List<Schedule> schedules = new ArrayList<>();
    Substitute rikard = new Substitute("Rikard");
    Substitute henrik = new Substitute("Henrik");
    Substitute anders = new Substitute("Anders");
    School yrgo = new School("Yrgo", "Lärdomsgatan 3 402 72 GÖTEBORG");
    School angered = new School("Angeredsgymnasiet", "Grepgatan 9 424 65 Angered");
    String date = "2018-01-17 08:00:00";
    Schedule schedule1 = new Schedule(date, rikard, yrgo);
    Schedule schedule2 = new Schedule(date, henrik, yrgo);
    Schedule schedule3 = new Schedule(date, anders, angered);
    schedules.add(schedule1); schedules.add(schedule2); schedules.add(schedule3);
    return schedules;
  }
}
```

org.json api calls used

- Create the JSONArray
  ```
  JSONArray JSON = new JSONArray();
  ```
- Add a whole JSONObject (a schedule) to the JSONArray:
  ```
  JSON.put(JSONSchedule(schedule)); // the method call returns a JSONObject
  ```

org.json api calls used

- Creating a nested JSONObject for a schedule, e.g.:

```
{
    "date": "2018-01-17 08:00:00",
    "substitute": {
      "name": "Rikard"
    },
    "school": {
      "school_name": "Yrgo",
      "address": "Lärdomsgatan 3 402 72 GÖTEBORG"
    }
}
```

```
JSONObject JSONSchedule = new JSONObject();
JSONSchedule.put("date", schedule.date());
JSONObject JSONSubstitute = new JSONObject();
JSONSubstitute.put("name", schedule.substitute().name());
JSONSchedule.put("substitute", JSONSubstitute);
JSONObject JSONSchool = new JSONObject();
JSONSchool.put("school_name", schedule.school().name());
JSONSchool.put("address", schedule.school().address());
JSONSchedule.put("school", JSONSchool);
```

Further reading

- https://www.codevoila.com/post/65/java-json-tutorial-and-example-json-java-orgjson
- https://stleary.github.io/JSON-java/ API Documentation for org.json
- Download org.json.jar: https://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.json%22%20AND%20a%3A%22json%22
- Presentation source code: https://github.com/progund/java-web/tree/master/java-json/org.json/creating