Workshop JSON-B

**Lab 1**

**1.1 Set up a working environment**

See README.md for some information on where to find JSON-B related libraries

and documentation.

Make sure you can write and execute Java code referencing the JSON-B and JSON-P

APIs and reference implementations.

**1.2 Serializing objects**

Run the Lab1 class and check out the output.

**1.3 Exercise**

Serialize some other basic objects like strings, numbers, booleans, date/times, and

look at the text output. Just play around!

**1.4 Exercise**

Serialize custom objects. Look at the Person class, and see how the sample data

gets serialized.

Make some changes to the code and see what effect they have on the serialized

text. (Add fields, change access modifiers, add or remove accessor methods, ...)

**1.5 Questions**

What happens when fields are non-public?

What happens when fields have a null value?

In what order are the fields serialized?

**1.6 Basic serialization configuration**

Use the JsonbConfig class to configure the way Jsonb handles your objects.

JsonbConfig config = new JsonbConfig();

Jsonb jsonb = JsonbBuilder.create(config);

As exercise, try out the following configurations, and lookup further options in

the spec documentation or libraries.

**1.7 Pretty-print**

To create fancy output (e.g. for a demo), you can use formatting. Try it!

config.withFormatting(true);

**1.8 Property naming strategy**

By default JSON object keys are identical to the Java property names.

A different property naming strategy can be configured using the line below.

config.withPropertyNamingStrategy(PropertyNamingStrategy.LOWER\_CASE\_WITH\_UNDERSCORES);

**1.9 Localized date format**

Formatting the date can be done using DateFormat.

config.withDateFormat("EEEE dd MMMM yyyy", new Locale("nl", "NL"));

**1.10 Deserializing objects**

Person p = jsonb.fromJson("{\"firstName\":\"Charles\",\"lastName\":\"Babbage\"}", Person.class);

Take a look at the "fromJson" line above.

Note that it is important that the field can be set for deserialization. Public getter and private or no setter may lead to unintended results, for instance.

You can try this by setting the firstName field in the Person class to private.

**1.11 Exercise**

Try and deserialize some more JSON strings.

What happens if the JSON text contains key names that are not present as

property in the Java target class?

**Lab 2**

In Lab 1 we saw some ways of starting to configure the output of JSON-B format,

but we can customize the behaviour even more adjusting the objects being mapped, using JSON-B annotations.

Try it out in Lab 2, make sure you only use JSON-B annotations!

Modify the Person class to answer the questions.

This link may come in handy to grasp the idea:

<https://json-b.net/docs/user-guide.html#changing-property-names>

**2.1 Properties**

Suppose consumers of the JSON representation of Person would like the JSON to be

in their own language (other than English).

Try using @JsonbProperty annotations in the Person class from Lab1 to make the

JSON change the values of the fields.

**2.2 Dates**

If the language uses custom date formatting, can you make sure the date of birth

is formatted appropriately according to the nationale?

**2.3 Null values**

Can you make sure the date of birth is always included in the output, even as

null value, avoiding any other null values to be printed?

**2.4 Ordering**

In Dutch the order of the serialized properties is last name ("achternaam"),

date of birth ("geboortedatum"), first name ("voornaam").

Technically it does not matter, but suppose you'd like the name fields together,

instead of separated by the birth date.

Can you change the order of the fields in the output using @JsonbPropertyOrder?

**2.5 JsonbCreator**

The parameterless constructor added to the Person class to help deserialization

does not look nice. Let's remove it and see what happens if we try (example in

Dutch, as that was used in earlier customization, you can use your own):

Person p = jsonb.fromJson("{\"voornaam\":\"John\",\"achternaam\":\"Doe\",\"geboortedatum\":\"01-01-1900\"}", Person.class);

Create your own constructor with the addition of @JsonbCreator to be able to deserialize the JSON text to a Person object.

**Lab 3 – Bonus**

In the first two labs quite a few ways of configuring JSON binding appeared.

Sometimes, however, you need to go even further. This is possible by using

adapters, or even specially crafted (de)serializers.

**3.1 Adapters**

First we will look at an external library of code we cannot change (at least we'll assume it is).

It has an UnmappablePerson that (although it can be mapped to JSON quite well) does not map to the serialized form we used for Person in Lab1.

Create a javax.json.bind.adapter.JsonbAdapter implementation that adapts UnmappablePerson to Person for serialization, register the adapter with Jsonb using JsonbConfig::withAdapters and check things work by executing code like:

JsonbConfig config = new JsonbConfig();

config.withAdapters(new UnmappablePersonAdapter());

Jsonb jsonb = JsonbBuilder.create(config);

UnmappablePerson unmappablePerson = new UnmappablePerson(new Name("New", "Person"), LocalDate.now());

System.out.println(jsonb.toJson(unmappablePerson));

Look at the JSON output of serializing the UnmappablePerson.

Can it be deserialized to a Person? And to an UnmappablePerson?

**3.2 Serializers/Deserializers**

Sometimes default mapping, annotations or adapters can't help with JSON-Object mapping needs.

Let's look at an example with different kinds of shapes.

Create serializers for the different shape types (e.g. CircleSerializer implements JsonbSerializer<Circle>).

Have the shapes serialized like:

{"square":{"side":2.0}}

{"rectangle":{"height":3.0,"width":2.0}}

{"circle":{"radius":1.0}}

Create a deserializer for shapes (ShapeDeserializer implements JsonbDeserializer<Shape>).

Serialize and deserialize some shape instances (register the serializers and deserializer with the JsonbConfig).

Deserializing a list of shapes is a little trickier.

Can you figure out how to?

String jsonShapes = "[{\"square\":{\"side\":2.0}},

{\"rectangle\":{\"height\":3.0,\"width\":2.0}},

{\"circle\":{\"radius\":1.0}}]";