CS 548—Fall 2023
Enterprise Software Architecture and Design
Assignment Two—JSON Schema and Binding

This assignment involves the data modeling for data transfer objects (DTOs) for a medical information system, that will be used later in a Web service. You should complete Java classes for the DTOs, as well as JSON Schema descriptions for DTOs as JSON objects. You are provided with a validation program for demonstrating your completed schemas working correctly. There is also a program for saving and reviewing JSON data, that you should complete and demonstrate working. You may find this program useful for creating instance data for testing against the validator. You should use the Intellij IDEA Ultimate IDE for this and future assignments.

A clinic database consists of a collection of patient records, one for each patient. Each patient has:
1. a patient identifier, a universally unique identifier (UUID), automatically generated when a patient record is added to the system;
2. a patient name; and
3. a patient date of birth.

The database also consists of healthcare provider records. Every provider record includes their automatically generated identifier, provider identifier (NPI) which may be assigned by the government, and their name.

There is obviously a many-to-many relationship between patients and providers, but we will represent it by two one-to-many relationships: from patients to treatments received, and from providers to treatments administered. Each treatment has a record that links back to the corresponding patient and provider (identified by their information system keys).

Every treatment record includes its own key, the key and name of the corresponding patient, the key and name of the corresponding provider, a diagnosis of the condition for which the treatment is prescribed (e.g. throat cancer, HIV/AIDS, hepatitis, etc), and a list of follow-up treatments. The remaining information for a treatment depends on what form of treatment it is. Currently there are four specific forms of treatment records:
1. A drug treatment record includes a drug and a dosage, starting and ending date for the treatment, and frequency (number of times a week, an integer).
2. A surgery treatment record includes a date of surgery, and discharge instructions.
3. A radiology treatment record includes a list of dates of radiology treatments.
4. A physiology treatment record includes a list of date of physiotherapy treatments.

The diagnosis, patient, provider and list of follow-up treatments can be common fields for all types of treatment records. Patients and providers are identified by their database identifiers. Their names are redundantly repeated in the treatment records to simplify future applications[1].

[1] This is an example of the *Subset Pattern* in data modeling for document-oriented NoSQL databases.

For the assignment, you are provided with an Intellij project that contains these Maven modules:

1. `clinic-dto`: This defines the Java classes for the DTOs, as well as factory classes. The drug treatment DTO is already defined, you must complete the remaining treatment DTO classes (treatments must be defined as Java beans) and define the factory methods for those treatment types. This module is used by the following two modules.
2. `clinic-json-bind`: This provides a program that allows patient, provider and treatment DTOs to be saved to and loaded from a "database" (a JSON file). You need to complete this code, as explained below.
3. `clinic-json-schema`: This provides JSON schemas for patients, providers and treatments, and a validation program that validates JSON data against the appropriate schema. The code is complete, but you must complete the schemas that are defined as resources in the module.
4. `clinic-root`: This is the parent module for the other modules. Its POM file defines the modules in the overall module. You should always use Maven to compile your sources and build your jar files in this module; its POM file has information that the other modules need for compilation.

When you run Maven on the root module (`mvn install`), it generates jar files for the `clinic-json-bind` and `clinic-json-schema` modules, called `jsonbind.jar` and `jsonschema.jar` respectively, and puts them in the following subfolder of your home directory: `tmp/cs548`. You should navigate to this folder to run the programs. You must run the jar files from the command line. Do not try to run these modules from the IDE.

Part 1: DTO Classes

The DTO classes for treatments are defined in the `clinic-dto` module. They subclass the abstract base class `TreatmentDto`. This defines some fields that all treatments have in common:

```java
public abstract class TreatmentDto {

   private UUID id;

   @SerializedName("patient-id")
   private UUID patientId;

   @SerializedName("patient-name")
   private String patientName;

   @SerializedName("provider-id")
   private UUID providerId;

   @SerializedName("provider-name")
   private String providerName;

   private String diagnosis;

   @SerializedName("followup-treatments")
   private Collection<TreatmentDto> followupTreatments;
```

The `@SerializedName` annotation is a Gson annotation to override the default naming conventions when serializing Java objects as JSON data. For this part of the assignment, your task is to complete the `SurgeryTreatmentDto` class and add remaining methods for the `TreatmentDtoFactory` class.

Part 2: JSON Binding

The `clinic-json-bind` module defines an app (`jsonbind.jar`) for loading and saving clinic information as a JSON file. It has a command-line interface (CLI) with these commands:

1. Use `load` to load an in-memory data store from the file specified and `save` to save it.
2. Use `list` to list all patients, providers and treatments in the data store.
3. Use `addpatient`, `addprovider` and `addtreatment` to add records to the data store. Each of these prompts you for the fields for the record.

You should complete the parts of the main app (`App.java`) that add treatments (currently only defined for drug treatments), and load and save treatments. You **must** load and save data in a streaming way, serializing individual records rather than serializing the entire database in one go. This is already done for patient and provider records. The app uses Gson for serializing to JSON; since Gson only recognizes `java.util.Date`, and the DTO classes use `java.time.LocalDate`, the module uses a custom serializer for `LocalDate` (defined in the `clinic-dto` module).

The main challenge for this part is the deserialization of treatment records to Java. Gson requires a concrete class for the JSON data being deserialized, but in general this is only known at runtime for each treatment record. The deserialization logic needs to look at the type tag for the object, and deserialize to a Java object based on that. However, if we try to register a custom deserializer to do this, it leads to an infinite loop if we try to call Gson recursively to deserialize the JSON object with the appropriate concrete subclass.

The solution to this is provided by the `RuntimeTypeAdapterFactory` class. This generates type adapters that add type tags to the serialized JSON data, and then use those type tags when deserializing the data. Your responsibility is to specify the concrete subclasses and their type tags for the `TreatmentDto` class, by completing the code for the `GsonFactory` class in the `clinic-dto` module.

Provide a video demonstration of the completed code working: adding patients, providers and each of the four kinds of treatments, show the file resulting from saving, reload that file after restarting the app, and list the records after loading.

Part 3: JSON Schema Validation

The `clinic-json-schema` module defines an app (`jsonschema.jar`) for validating JSON data against a schema. It accept as input lists of patients, providers and treatments following the format of the data in the first part of the assignment. It validates each patient, provider and treatment JSON object in the input. You can edit the output from jsonbind to introduce errors (missing an extraneous fields), and test that the validator catches these errors. If `db.json` is a file output from `jsonbind`, you can run `jsonschema` as follows:

```
java -jar jsonschema.jar < db.json > db.out
```

The schemas are defined as resources in the module (in the `schema` subfolder of `src/main/resources`). Your task here is to complete each of the schemas, and then demonstrate validation succeeding for some data and failing for others. For failure cases, show failure due to missing fields, or fields that are present that are not in the schema. Show this for patient and provider data, and for each kind of treatment data.

The format of the schemas for patients and providers is straightforward. You need to describe the types of the fields (make sure they are the same names as those in the serialized data provided by Gson from DTOs), require that these fields are present, and not allow any other fields.

The schema for treatments is the interesting part. The base for all treatment schemas is defined first as a local definition:

```
"$defs": {
       "base": {
              "type": "object",
              "properties": {
                     "id": { "type": "string", "format": "uuid" },
                     "type-tag": { "enum": [ ... ] }, ...
              },
              "required": ["id","type-tag", ... ]
       },
```

Then the `allOf` composition operation is used to combine this with the extensions for each treatment type[2]:

```
       "drug-treatment": {
              "allOf": [ { "$ref" : "#/$defs/base" } ],
              "properties": {
                     "drug": {"type": "string" }, ...
              },
              "required": [ "drug", "dosage", ... ],
              "unevaluatedProperties" : false
       }
```

Then the `oneOf` operation is used to form the overall treatment type as the disjoint union of these types:

```
{
       "$id": "https://cs548.stevens.edu/clinic/treatment",
       "$schema": "https://json-schema.org/draft/2020-12/schema",
       "description": "JSON Schema for treatments",
       "oneOf" : [
              { "$ref" : "#/$defs/drug-treatment" }, ...
       ], ...
```

[2] To understand this better, you can try changing `unevaluatedProperties` to `additionalProperties`, and see what happens with validation.

This has been done so far for the drug treatment type, you need to complete it for the other forms of treatment.

Once you have completed the JSON schemas in the module and rebuilt the jar file for the app, demonstrate the schemas working, validating well-formed data and failing to validate data that is not well-formed. Do this demonstration for patient records, provider records and treatment records of different types (You don't need to bother demonstrate schema validation for follow-up treatments). Show schema validation failing for missing fields and additional fields not recognized by the schema.

Submission

Once you have your code working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
2. In that directory you should provide all of the Maven modules, including your changes.
3. Record videos of your testing of your solutions for testing and validation (Parts 2 and 3). The video should include your name, the test input, and the console output. **Only MP4 is allowed.**

In addition, complete the rubric provided with this assignment, for your own self-evaluation. Note that a penalty may be incurred for misrepresentation in the rubric.

Your solution should be uploaded via the Canvas classroom, as a zip file. This zip file should have the same name as your name. It should unzip to a folder with this same name, which should contain the files and subfolders with your submission. In addition to the completed modules, your submission should include videos of your tests, as well as a completed rubric.