Dear course participants,

Regarding the Task-7 and corresponding sample project that I have shared with you...

The SSWAP library was developed using Java 8 and is not updated any more in context of compatibility with later Java versions. The sample project was developed using Java version 1.8.0_151. Thus, if you downgrade your java version correspondingly, you will be able to adopt the sample project or create similar one from the scratch using the sample project as reference (recommended). Using later versions might lead to the version conflicts some of the jars (particularly icu4j.jar).

For example, last year, some students successfully managed to run the "sample" project using the icu4j.jar version 68.2 with the java versions 1.8.0_372 and 1.8.0_392.... Basically, if you use older versions (e.g. 1xx, 2xx) you may still use older versions of icu4j.jar file. So, at least we know that with those versions it works...

Regarding the other languages... As I mentioned before, you may do the task using any programming language as you wish... SSWAP java library is just an implementation of SADI web service that helps us quickly implement (if the issues with dependences are resolved 😉) and see a simple semantic web service in action. Alternatively, SADI service can be implemented for example in Python as well (last year some teams have tried to do it in Python adopting the idea of RDG/RIG/RRG from SSWAP protocol with minimum necessary functionality) ...

Just as a reminder...

The main steps of SADI service implementation:

- SADI Web services are stateless and atomic;
- SADI service endpoints respond to HTTP GET by returning the interface definition of the service (you may use RDG);
- Service interfaces (i.e., inputs and outputs) are defined in terms of OWL-DL classes; the property restrictions on these OWL classes define what specific data elements are required by the service and what data will be provided by the service, respectively (the idea of RDG/RIG/RRG fully satisfy the requirements);
- SADI services consume and produce data in RDF format (the idea of RDG/RIG/RRG fully satisfy the requirements);
- SADI services are invoked through plain HTTP POST of RDF data to the service endpoint. All information required for service invocation must be present in the data itself, because SADI uses a non-parameterized POST - i.e. does not use the HTTP FORM encoding;
- Input RDF data - data that is compliant with (i.e. classifies into) the input OWL Class definition – is “decorated” or “annotated” by the service provider to include new properties until it fulfills the Class definition of the service’s output OWL Class. Importantly, in so doing, the URI of the input OWL Class Instance is preserved and becomes the URI of the output OWL Class Instance. Basically, idea of RDG->RIG->RRG transformation fits this requirement as well.