INF3580/4580 – MANDATORY EXERCISE no. 2

Published date: 30.01.2018

Due date: 07.02.2018 23:59.

Delivery file: 1: `Simpsons.java`.

Delivery attempts: 1.

Read the whole of this document thoroughly before solving any of the exercises. The objective of this exercise is to learn and use basic Jena methods. You shall write a java program, called `Simpsons.java`, which takes two arguments. The first argument is the path to an RDF file, the path may be an URL or a local path. The second argument is a filename to which the output shall be written. You shall assume that the RDF serialisation of both files is correctly indicated by their file extension: `.rdf` = RDF/XML, `.ttl` = Turtle, `.n3` = N3 and `.nt` = N-Triples.

Your program shall create a model from the input file, add triples to this model, add more triples to the model based on data in the model, and, finally, output the model to file. The steps must be carried out in this sequence. Each step is explained in detail below.

1 Read input

Your program shall read the RDF file given in the first argument to the program and create a Jena model from it.

Tip: It seems that Jena is not so good at discovering the RDF serialisation used in files by itself, especially for files addressed with an URL. A solution to this problem is to make a method that gets the file extension of the filename by the use of string functions and returns the appropriate RDF serialisation format in Jena's predefined strings. You can then use your method both for reading input and writing to file in the correct serialisation format.

You can assume that the input file has defined the following prefix–namespaces:

prefix	namespace
xsd	http://www.w3.org/2001/XMLSchema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
sim	http://www.ifi.uio.no/INF3580/simpsons#
fam	http://www.ifi.uio.no/INF3580/family#
foaf	http://xmlns.com/foaf/0.1/

2 Adding information

Your program shall then *add* the following information about Maggie, Mona, Abraham and Herb, whose identifiers are respectively `sim:Maggie`, `sim:Mona`, `sim:Abraham` and `sim:Herb`, to the model read from the input file. You will need to use the class `foaf:Person`, the predicates `rdf:type`, `foaf:age`, `foaf:name`, `fam:hasSpouse` and `fam:hasFather`, and the standard `xsd` datatypes.[1] You shall assume that the prefixes `sim` and `fam` are defined in the model that your program has read in the section above, so do not hardcode these namespaces in your program, but get them from the model you have created. (Hint: `Model` is a subinterface of `PrefixMapping`.)

Add the following information to the model using Jena methods:

- Maggie is a person, whose name is "Maggie Simpson" and age is 1. The datatype of the age value is `xsd:int`.
- Mona is a person, whose name is "Mona Simpson" and age is 70. The datatype of the age value is `xsd:int`.
- Abraham is a person, whose name is "Abraham Simpson" and age is 78. The datatype of the age value is `xsd:int`.
- Abraham is the spouse of Mona, and Mona is the spouse of Abraham.
- Herb is a person.
- Herb has a father (but we do not know who it is, i.e., use a blank node.).

3 Locate, read and write information

For each person in the model, i.e., for every resource of type `foaf:Person`, add the following information according to the person's age:

- if age $<$ 2 year, then add a triple stating that the person is of type `fam:Infant`,

[1] The intended meaning of these classes and predicates are as "expected" and is listed in the previous mandatory exercise.

- if age $<$ 18 year, then add a triple stating that the person is of type `fam:Minor`,
- if age $>$ 70 year, then add a triple stating that the person is of type `fam:Old`.

Example: If the input model contains the triples

```
sim:Someone rdf:type foaf:Person ;
            foaf:age "75"^^xsd:int .
```

then your program shall add the triple

```
sim:Someone rdf:type fam:Old .
```

to the model.

4 Write to file

Finally, write the model to the file given as second argument to the program and in the RDF serialisation as specified by the file's extension.

5 Ending notes

5.1 Executing program

The program must successfully compile with the command

```
javac -cp "path/to/jena/lib/*:." Simpsons.java
```

where `path/to/jena/lib/*` contains the Jena jar files. If you have written the program using Eclipse, then you must probably remove any package specification from the program. The source code must be nicely structured and commented so it is easy to understand the program and the intention behind its parts.

NB: If you use windows replace ":" with ";" in the classpath string.

Running the command

```
java -cp "path/to/jena/lib/*:." Simpsons oblig/2/simpsons.ttl output.ttl
```

where `oblig/2/simpsons.ttl` is a file containing the prefixes as specified in the exercise text (rdf file from oblig1 file should do), shall produce an RDF Turtle file as specified by the exercise. To do this you can use the following `Makefile`.

```
JAVA_CP = "lib/jena/lib/*:."
SIMPSONS_FILE = oblig/2/simpsons.ttl

java = java -cp $(JAVA_CP)
javac = javac -cp $(JAVA_CP)

%.class: %.java
        @$(javac) $<

output.ttl: Simpsons.class
        @$(java) $(basename $(<F)) $(SIMPSONS_FILE) $@

run_test: Test.class
        @$(java) $(basename $(<F)) $(SIMPSONS_FILE) $@
```

Change the variable `JAVA_CP` to contain the path to where you keep your Jena jar files. Placing this `Makefile` in the same folder as your java program and writing

```
make output.ttl
```

should compile your java program and create the file `output.ttl` as specified in this exercise.

Below is a tiny test program called `Test.java` you can use to test the above `Makefile` and that compiling and running java works. Running

```
make run_test
```

from inside the same folder as where the `Makefile` and `Test.java` is located should output the two input arguments given to the program.

```
public class Test{
  public static void main(String args[]){
    System.out.println("Hello, the two input arguments are: \n\t(1)  "
                       + args[0] + " \n\t(2)  "
                       + args[1]);
  }
}
```

The `Makefile` and `Test.java` is available for download at `http://www.uio.no/studier/emner/matnat/ifi/INF3580/v18/undervisningsmateriale/oblig2-files/`.

5.2 Delivery, Devilry

Mandatory exercises are to be handed in using Devilry. Make sure that your are registered in the system by logging on and finding that an `oblig2` is available as an assignment in INF3580 or INF4580. *Check this before you*

start solving the exercises! If you are not registered in the system, give notice to `leifhka@ifi.uio.no`.

Your delivery shall contain *one* file, `Simpsons.java`, which is not to be placed inside any folders, zip-file or similar. Simply upload the source code file `Simpsons.java` to Devilry.

5.3 Mr. Oblig

You can, and should, use Mr. Oblig to test your delivery before handing it in. Mr. Oblig is located at `http://sws.ifi.uio.no/mroblig/`. Note that Mr. Oblig comes without any warranty. A non-functioning Mr. Oblig will not have any effect on the due date of this mandatory exercise, and a flawless test report from Mr. Oblig does not guarantee that your delivery will be graded with *passed*. Also, since there may be many correct answers to an exercise, it is possible that Mr. Oblig does not agree with your solution even though it is correct. However, a perfect score from Mr. Oblig is an indication that your delivery is good and that it does not contain any "stupid" errors.

Note that you can only upload the output of running your java program to Mr. Oblig, and not the java program. You should be careful to test the program with different outputs, e.g., with different values of the prefixes `sim` and `fam`, and different ages—and of course the limit cases, e.g., age 17 and age 18—as input, before handing in.

Good Luck!