Practical: Knowledge Graph Construction

1. Schema Modelling

In this part of the practical, we will look into schema modelling. You might want to remind yourself on the related slides (12 to 21) in the Knowledge Construction lecture.

Your task is to download Protégé (a widely used ontology editor), then follow the instructions and do the exercises in Chapter 4.

Acknowledgements: Many thanks to colleagues from the University of Manchester (Dr. Matthew Horridge, Prof. Alan Rector and the Protégé-OWL Team) for sharing the Protégé tutorial.

Note: These tutorials were written for version 4 of Protégé; therefore, please bear in mind that the interface shown in the tutorial might be different from the version of Protégé that you use.

Lab instructions:

1. Download Protégé from https://protege.stanford.edu/download/protege/, we recommend version 5. Note that the following tutorial is for version 4 of Protégé, so mind the UI differences.
2. Follow Chapter 4 of the following tutorial: http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf (local copy: https://knowledge-representation.org/j.z.pan/data/ProtegeOWLTutorialP4_v1_3.pdf)

Q1.1

Start building your Pizza ontology: follow instructions of Exercises 2 - 7 on pages 14 - 22 of the Protégé 4 Tutorial.

Q1.2

Create some class restrictions (using Existential and Universal restrictions) for your Pizza ontology: follow instructions of Exercises 8 - 17 on pages 23 - 43 of the Protégé 4 Tutorial.

Q1.3

Add pizzas into your Pizza ontology and run a reasoner: follow instructions of Exercises 18 - 27 on pages 43 - 43 of the Protégé 4 Tutorial.

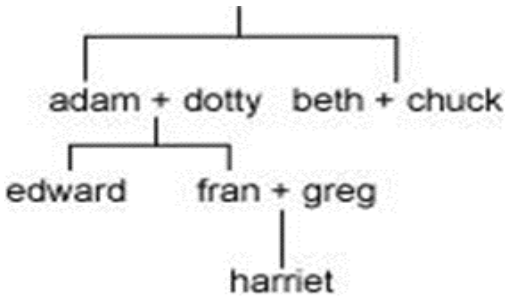
2. Data Lifting

In this part of the practical, we will deal with data lifting. You might want to remind yourself on the related slides (23 to 36) in the KG Construction lecture, where we discussed about data lifting from Wikipedia infobox, from databases and from texts. No matter what source data we use, we still need some tool to populate the data into your knowledge graph schema (ontologies).

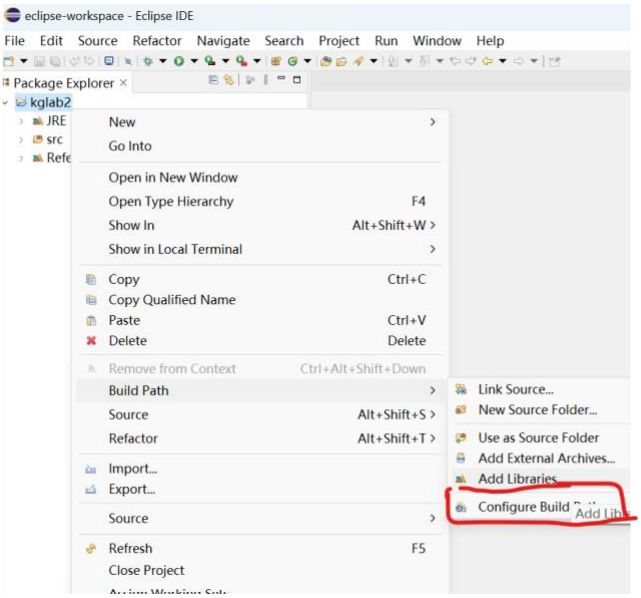
2.1 Data Lifting with Jena

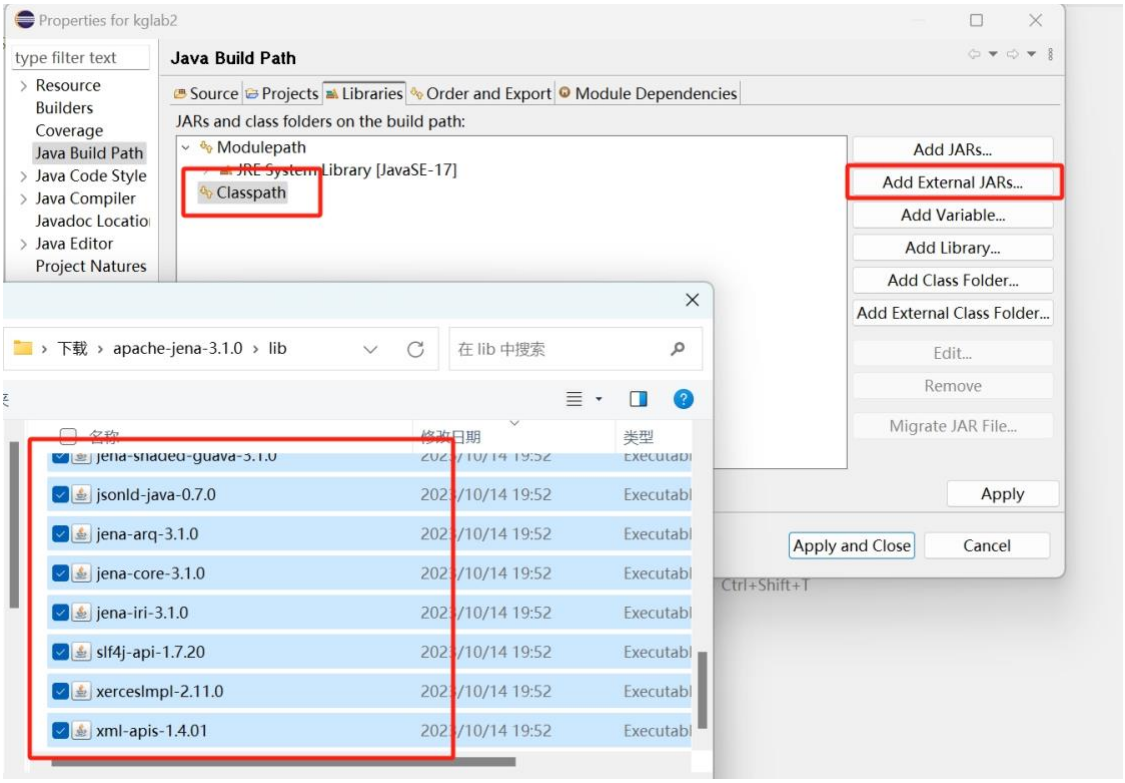
Here you will create a Java program, which uses Jena to create the family graph. These instructions describe how to achieve this using the Eclipse IDE; you are however free to use your favourite IDE if you know how to add files to the "classpath" and run the application. You can download Jena Libraries from this link Jena (https://archive.apache.org/dist/jena/binaries/). From many choices in the link, please download file `apache-jena-3.1.0.tar.gz` and unzip the contents. There will be many files after you unzip it, but two most important folders in this unzip folder is `lib/` and `lib-src/`.

More precisely, you are going to use Jena to populate an empty family basic-family.owl ontology with some initial values, which will be extended in the next practical. In the terminology of Jena, ontology is not only referred to the schema but is used as a synonym of knowledge graph --- an ontology can contain both the schema and entity triples. Your task is to add the following family tree into the knowledge graph.



Step 0. Create a new Eclipse project called `kglab2`, put the basic-family.owl file into the project folder, and add the jar files in the `lib/` folder to the build path.





Step 1. Create a new Java file (File -> New -> Class). Call it `JenaFamilyPopulation` select the "`public static void main(String[] args)`" option, click on the OK button (remember, we are using the Eclipse IDE).

Step 2. Import the relevant package as follow.

```
import java.io.File;
import java.io.FileOutputStream;
import org.apache.jena.ontology.Individual;
```

```
import org.apache.jena.ontology.OntClass;
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.ModelFactory;
```

Step 3. Create a global variable called `ns` with the value you set the namespace of the ontology to; use the following example code.

```
private static final String ns = "http://www.ed.ac.uk/Practical#";
```

Step 4. Create another global variable called filename, with the value of your basic-family.owl file.

```
private static final String filename = "basic-family.owl";
```

Step 5. Insert the following code to the `main` method to create a new OntModel, and read the family ontology.

```
// create an ontology model
OntModel ontology = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM)
;

// add an ontology stored in a file to the ontology model ñ change the file n
ame to your own
File file = new File(filename);
ontology.read(file.toURI().toString());
```

Step 6. To create the first Man individual.

```
// create men
OntClass manCls = ontology.getOntClass(ns+"Man");
Individual man1 = ontology.createIndividual(ns+"man1",manCls);
man1.addProperty(ontology.getProperty(ns+"hasGender"), ontology.getIndividual
(ns+"Male"));
```

Step 7. To create the first Woman individual.

```
// create women
OntClass womanCls = ontology.getOntClass(ns+"Woman");
Individual woman1 = ontology.createIndividual(ns+"woman1",womanCls);
woman1.addProperty(ontology.getProperty(ns+"hasGender"), ontology.getIndividu
al(ns+"Female"));
```

Step 8. To set the hasConsort property of man1 to be woman1.

```
// set man1 hasConsort to woman1
man1.setPropertyValue(ontology.getProperty(ns+"hasConsort"), woman1);
```

Step 9. Repeat this to define the hasConsort and hasChild relationships described below.

M1 hasConsort F1
M1 hasChid M2, F2, F3

M2 hasConsort F4
M2 hasChild M3 M5 F5
M3 hasConsort F8
M3 hasChild F9
M4 hasConsort F3
M4 hasChild M6 F6
M6 hasConsort F10
M6 hasChild M9
M7 hasConsort F7
M7 hasChild M4
M8 hasConsort F6
M8 hasChild M10

Step 10. Finally, save the populated ontology.

```
// save the ontology
FileOutputStream out = new FileOutputStream(filename);
ontology.write(out);
out.close();
ontology.write(System.out);
System.out.println("Completed");
```

You will need to add `throws Exception` to the main method declaration

```
public static void main(String[] args) throws Exception {
    // ...
}
```

Eventually, the project structure should look like this:

```
└─kglab2
    │  .classpath
    │  .project
    │  basic-family.owl
    └─src
        └─kglab2
                JenaFamilyPopulation.java
```

and the source code file should look like this:

```
package kglab2;

import java.io.File;
import java.io.FileOutputStream;
import org.apache.jena.ontology.Individual;
import org.apache.jena.ontology.OntClass;
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.ModelFactory;
```

```
public class JenaFamilyPopulation {

	private static final String ns = "http://www.ed.ac.uk/Practical#";
	private static final String filename = "basic-family.owl";

	public static void main(String[] args) throws Exception {
		// create an ontology model
		OntModel ontology = ModelFactory.createOntologyModel(OntModelSpec
.OWL_DL_MEM);

		// add an ontology stored in a file to the ontology model ñ chang
e the file name to your own
		File file = new File(filename);
		ontology.read(file.toURI().toString());

		// create men
		OntClass manCls = ontology.getOntClass(ns+"Man");
		Individual man1 = ontology.createIndividual(ns+"man1",manCls);
		man1.addProperty(ontology.getProperty(ns+"hasGender"), ontology.g
etIndividual(ns+"Male"));

		// create women
		OntClass womanCls = ontology.getOntClass(ns+"Woman");
		Individual woman1 = ontology.createIndividual(ns+"woman1",womanCl
s);
		woman1.addProperty(ontology.getProperty(ns+"hasGender"), ontology
.getIndividual(ns+"Female"));

		// set man1 hasConsort to woman1
		man1.setPropertyValue(ontology.getProperty(ns+"hasConsort"), woma
n1);


	// ==================================
	//          more code here
	// ==================================


		// save the ontology
		FileOutputStream out = new FileOutputStream(filename);
		ontology.write(out);
		out.close();
		ontology.write(System.out);
		System.out.println("Completed");
	}
}
```

Step 11. Finally, run the program: Run, Run As, Java Application. See if you could open the saved ontology with Protégé.

Step 12. Next, see if you could add the above family tree into the knowledge graph.

2.2 Lifting Unstructured Data

Please follow the instructions here:
https://colab.research.google.com/drive/1SYd8x3oIM0-d--irzJkrQ353cPtKMCMu?usp=sharing