Assignment#3

***Note:**
Assignment#3 contributes to 10% of the total mark of the course.
Deadline: May 12, 2020 before 14:00
Submission: Send your work as an A#3-XXX.zip file, for XXX your student ID, via the PedagogySquare platform. The zip file should contain a .pdf file and a .java file.

Q1. Tableaux (30 marks)

1. Show how the tableaux algorithm for ALC as presented in class would be used to prove the satisfiability/unsatisfiability of the class expression:
   (R some (S some (not C))) and (R only C) (10 marks)

2. Given the ontology with a single axiom:
   Class: A SubClassOf: B
   Show how the tableaux algorithm for ALC as presented in class would be used to prove the satisfiability/unsatisfiability of the class expression:
   (R some A) and (R only (not B)) (10 marks)

3. Check whether (R some A) and (R some B) is subsumed by R some (A and B) (10 marks)

You can produce this document in any way you wish – word, LaTex, hand-drawn figures etc., but the submission must be a PDF.

Q2. APIPopulation (70 marks)

Populating Family History

A data set describing aspects of family history forms the content of this exercise. It contains information about people and the occupations or roles they played. Each individual described has associated information:

- Surname
- Married Surname (if known)
- Birth Year (if known)
- Given Names (first name)

Along with this, there will be a number (possibly zero) of roles/occupations played. For each of these there will be

- Year (if known)
- Source
- Occupation (if known – this may be “none given”, stating that the role is unknown).
- Data is provided as CSV (comma separate values). Code is given that will parse this

CSV file into a collection of Bean Objects with appropriate get and set methods. The parser returns a collection of Beans, representing each row in the spreadsheet. You can assume that given name, surname and date of birth are sufficient to distinguish different individuals. If no year, source or occupation is given in a row, this row can be ignored.

For the assignment, you will write a Java program that populates an ontology with this data using the OWL API version 5.0.0. The ontology containing class and property definitions is given in the archive. You should not need to provide any additional Classes, Properties or definitions to the ontology – all your additions should concern individuals.

Each person identified in the data should be represented as a named individual (of type Person) in the ontology. Each role played should result in an instance of RolePlayed, with appropriate relationships to a Role instance, a Source and a year (if known). As discussed in the class, the individuals representing roles played may be named, while some may be left as anonymous individuals. The choice is yours. The names of role classes in the CSV file should match those given in the Role hierarchy in the ontology, but note that you may have to do a small amount of processing to map roles/occupations in the data to Roles in the ontology. Hint: you may need to think carefully about how to handle the special case “none given”, which is not the name of a role(!), but is used to indicate that no explicit role was named.

Your submission will be tested by loading a set of unseen data and then evaluating queries against this data. A sample set of data is provided in test/test.csv. The ontology for population is provided in resources/ontology.owl.

You will need to implement 3 methods in the owlapi.khkb.fspopulation.CW3 class:

- loadOntology(OWLOntologyManager manager, IRI ontologyIRI)
  Loading ontology from physical IRI (15 marks)
- populateOntology (OWLOntologyManager manager, OWLOntology ontology, Collection<JonDataBean> beans)
  Populating the ontology using the collection of Java beans holding the data from CSV (40 marks)
- saveOntology (OWLOntologyManager manager, OWLOntology ontology, IRI locationIRI)
  Saving the ontology back to the disk (15 marks)

You can add additional methods to this class if you wish, but do not change the signature of the existing methods or of the no-argument constructor. You will embed all the required functionality in CW3.java. As this is the only file you will submit, please do not implement any essential functionality outside of CW3.java. The harness is configured as a maven project, and you can do your testing through 3 implemented Junit tests. In order to set your project up, do the following:

1. Extract the cw3.zip somewhere on your computer
2. Open your Eclipse, and select File->Import->Existing Maven Project and select the directory you just unpacked.
3. Go to the Build Path settings and make sure that the JRE selected is in fact the one from your JDK.
4. Right click on the project: Maven clean.

5. Right click on the project: Maven install

You will see your project failing to build because it does not pass the Junit tests. Now you are good to go. As a suggestion, run the Junit tests frequently within Eclipse to monitor your progress. The tests will take the input from resources/test.csv, the ontology from resources/ontology.owl and will produce a populated ontology in fhkb-ai.owl.

Once you have finished the assignment, you should submit a zip archive of the CW3.java file in your project directory.