Assignment#4

Yizheng Zhao

June 9, 2021

Note: Assignment#4, due on 14:00 July 6, contributes to 10% of the total mark of the course.

Q1. **API Population — 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 Name (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 give", 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 three 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.

Q2. CW5 is all about querying implementing and OWL based querying system. Your task will be to query the reasoner for sub-classes, equivalent classes and instances, and store the results in a QueryResult object. You are asked to implement the following methods in the Java class CW5 (project layout the same as CW3!). Do not alter any class other than CW5, which is the only one you are (allowed) to submit! The classes App.java and PizzaOrderingSystemApp.java provide a, hopefully fun, way to test what you are doing is correct. You can run them in the same way as you ran the unit tests in CW3, simply by hitting the small white and green arrow icon in Eclipse when the class is open. If you are not interested in having fun, then you can, in the same way as you did last time, use the implemented unit tests to figure out whether you are on the right track. Please note that this time passing the unit tests does not automatically mean full marks. We will use some of our own tests to further validate your solution. Once you are done, please zip and submit CW5.java to the PedagogySquare platform. Good Luck!

Tasks:

1. `public Set<QueryResult> performQuery (OWLClassExpression exp, Query Type type) {...}`
   This method takes in an arbitrary expression in OWL, like "Person and hasBirthYear value 1964" or "hasTopping some MeatTopping" and queries the ontology for the following three types of knowledge:
   ⋆ EquivalentClasses: you are asked to return all those (named) classes that are equivalent to the expression (exp), using the (already fully initialized) reasoner.
   ⋆ Sub-classes: return all those classes that are (named) sub-classes of the expression, using the reasoner.
   ⋆ Instances: return all those individuals that are instances of the expression, using the reasoner.
   Query results are stored in query result objects. These are created for example as follows:
   `QueryResult qr = new QueryResult(ind, false, type);`
   Note that you need to include the information whether the inference is direct or indirect. Example: Given three classes with A subclass B subclass C, then A is a direct subclass of B and B is a direct subclass of C, but A is an indirect subclass of C (through B!). In order to solve this problem, you will query the reasoner for direct and indirect sub-class separately. After creating the QueryResult, add it to the provided set.

2. public Boolean isValidPizza (OWLClassExpression exp) {...};
   In this method, you check whether the supplied class expression exp is a valid pizza expression, i.e., whether it is inferred to be a Pizza.

3. public Set<QueryResult> filterNamedPizzas (Set<QueryResult> results) {...};
   This question is similar to the one before, only that you are asked to filter from a set of results those that correspond to NamedPizza's, such as Margherita or AmericanHot.

4. public Set<OWLClassExpression> getAllSuperClassExpressions (OWLClass ce) {...};

This question requires a bit of thinking. You are asked to query the ontology for all information you can find about the class ce. In order to get an idea of what "AL" means, you can open Protégé and look at the "Classe" tab. If you click on a class, you will find that it often has not only super-classes and equivalent-classes, but also inherited super-classes (Anonymous Ancestors), and of course indirect super-classes. Unfortunately the reasoner will not easily give you access to those. In order to get full marks for this task, it is sufficient to query for all super-classes (direct and indirect, using the reasoner), and somehow find a way to obtain the anonymous super-classes using the ontology directly (that will need a bit of thought, do not despair too quickly). A perfect solution will test, for all sub-expressions in the ontology, whether ce is a sub-concept of it. Attempt this only if you are really confident with the OWL API by now.