```
/*
 * Created on 2004

 */
package OA;



/*
 * Created by @author ernesto
 * I_Sub Class:  This class implements the string matching method proposed in the
paper
 * "A String Metric For Ontology Alignment", published in ISWC 2005
 * Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab8/I_Sub.java
 * Type of Code: Java code
 *
 * Updated by @author Shreyas Jadhav, Apoorva Ramaiah
 * Coursework: Semantic Web and Technologies Part 2
 * Coursework Group Name: Nerds
 * Updated on 11 May 2024
 *
*/

public class I_Sub{

	public double score( String st1 , String st2 ){

		if( (st1 == null) || (st2 == null) ){
				return -1;
		}

		String s1 = st1.toLowerCase();
		String s2 = st2.toLowerCase();

		s1 = normalizeString( s1 , '.' );
		s2 = normalizeString( s2 , '.' );
		s1 = normalizeString( s1 , '_' );
		s2 = normalizeString( s2 , '_' );
		s1 = normalizeString( s1 , ' ' );
		s2 = normalizeString( s2 , ' ' );

		int l1 = s1.length(); // length of s
		int l2 = s2.length(); // length of t
```

```
		int L1 = l1;
		int L2 = l2;

		if ((L1 == 0) && (L2 == 0))
			return 1;
		if ((L1 == 0) || (L2 == 0))
			return -1;

		double common = 0;
		int best = 2;

		int max = Math.min(l1, l2); // the maximal length of a subs

		while( s1.length() >0 && s2.length() >0 && best !=0 ){
			best = 0; // the best subs length so far

			l1 = s1.length(); // length of s
			l2 = s2.length(); // length of t

			int i = 0; // iterates through s1
			int j = 0; // iterates through s2

			int startS2 = 0;
			int endS2 = 0;
			int startS1 = 0;
			int endS1 = 0;
			int p=0;

			for( i = 0; (i < l1) && (l1 - i > best); i++) {
				j = 0;
				while (l2 - j > best) {
					int k = i;
					for(;(j < l2) && (s1.charAt(k) != s2.charAt(j)); j++);
						//System.out.println( s1.charAt( k ) + " " +
s2.charAt( j ) );

					if (j != l2) { // we have found a starting point
						//System.out.println( "j: " + j );
						p = j;
						for (j++, k++;
							(j < l2) && (k < l1) && (s1.charAt(k) ==
s2.charAt(j));
							j++, k++);
```

```
                                        if( k-i > best){
                                                best = k-i;
                                                startS1 = i;
                                                endS1 = k;
                                                startS2 = p;
                                                endS2 = j;
                                        }
                                        //best = Math.max(best, k - i);
                                }
                        }
                }
                //Vector v = new Vector();
                //if( startS1 != endS1 )
                //      System.out.println(  s1.substring( startS1 , endS1 ) );
                char[] newString = new char[ s1.length() - (endS1 - startS1) ];

                j=0;
                for( i=0 ;i<s1.length() ; i++ ){
                        if( i>=startS1 && i< endS1 )
                                continue;
                        newString[j++] = s1.charAt( i );
                }

                s1 = new String( newString );

                newString = new char[ s2.length() - ( endS2 - startS2 ) ];
                j=0;
                for( i=0 ;i<s2.length() ; i++ ){
                        if( i>=startS2 && i< endS2 )
                                continue;
                        newString[j++] = s2.charAt( i );
                }
                s2 = new String( newString );

                //if( (startS1 < 1 || startS1 > 2 )
                //      ||      (startS2 < 1 || startS2 > 2) && startS1 != startS2  )
                //      best--;

                if( best > 2 )
                        common += best;
                else
                        best = 0;

//System.out.println( s1 + ":" + s2 );
```

```
                //System.out.println( "StartS1 : " + startS1 + " EndS1: " + endS1 );
                //System.out.println( "StartS2 : " + startS2 + " EndS2: " + endS2 );
                }

                double commonality = 0;
                double scaledCommon = (double)(2*common)/(L1+L2);
                commonality = scaledCommon;

                double winklerImprovement = winklerImprovement( st1 , st2 , commonality
);
                double dissimilarity = 0;

                double rest1 = L1 - common;
                double rest2 = L2 - common;

                double unmatchedS1 = Math.max( rest1 , 0 );
                double unmatchedS2 = Math.max( rest2 , 0 );
                unmatchedS1 = rest1/L1;
                unmatchedS2 = rest2/L2;

                /**
                 * Hamacher Product
                 */
                double suma = unmatchedS1 + unmatchedS2;
                double product = unmatchedS1 * unmatchedS2;
                double p = 0.6;   //For 1 it coincides with the algebraic product
                if( (suma-product) == 0 )
                        dissimilarity = 0;
                else
                        dissimilarity = (product)/(p+(1-p)*(suma-product));

                return commonality - dissimilarity + winklerImprovement;
        }

        private double winklerImprovement( String s1 , String s2 , double commonality ){

                int i;
                //int n = Math.min( 4 , Math.min( s1.length() , s2.length() ) );
                int n = Math.min( s1.length() , s2.length() );
                for( i=0 ; i<n ; i++ )
                        if( s1.charAt( i ) != s2.charAt( i ) )
                                break;

                double commonPrefixLength = Math.min( 4 , i );
```

```
		//double commonPrefixLength = i;
		double winkler = commonPrefixLength*0.1*(1-commonality);

		return winkler;
	}

	/* (non-Javadoc)
	 * @see
com.wcohen.ss.AbstractStringDistance#explainScore(com.wcohen.ss.api.StringWrappe
r, com.wcohen.ss.api.StringWrapper)
	 */
	public String explainScore(String s, String t) {
		return null;
	}

	public String normalizeString( String str , char remo ){

		StringBuffer strBuf = new StringBuffer();

		int j=0;
		for( int i=0 ; i<str.length() ; i++ ){
			if( str.charAt( i ) != remo )
				strBuf.append( str.charAt( i ) );
		}
		return strBuf.toString();
	}




}


package OA;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/LexicalSimilarity.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
```

```
* Updated on 11 May 2024
*/

import org.apache.commons.text.similarity.JaroWinklerSimilarity;
import org.apache.commons.text.similarity.LevenshteinDistance;
//import org.apache.commons.text.similarity.CosineSimilarity;

public class LexicalSimilarity {


	public LexicalSimilarity() {

		//Reference: https://commons.apache.org/proper/commons-
text/apidocs/org/apache/commons/text/similarity/
		//Another potential library: https://github.com/tdebatty/java-string-similarity

		JaroWinklerSimilarity jw_sim = new JaroWinklerSimilarity();
		LevenshteinDistance l_dist = new LevenshteinDistance();
		I_Sub isub = new I_Sub();


		System.out.println("Jaro Winkler Similarity: " + jw_sim.apply("Congo",
"Republic of Congo"));
		System.out.println("Levenshtein Distance: " + l_dist.apply("Congo",
"Republic of Congo"));

		System.out.println("I-sub Similarity: " + isub.score("Congo", "Republic of
Congo"));


	}



	public static void main(String[] args) {
		new LexicalSimilarity();
	}

}

package OA;

/*
 * Created by @author ernesto
```

```
 * Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab8/AccessEntityLabels.java
 * Type of Code: Java code
 *
 * Updated by @author Shreyas Jadhav, Apoorva Ramaiah
 * Coursework: Semantic Web and Technologies Part 2
 * Coursework Group Name: Nerds
 * Updated on 11 May 2024
*/



import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.ontology.OntClass;
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.NodeIterator;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.ResIterator;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.OWL;
import org.apache.jena.vocabulary.RDFS;

public class Task_OA_1 {

	OntModel model;

	public void loadOntologyFromURL(String sourceURL) {
```

```
                model = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM
);
      model.read(sourceURL, "RDF/XML" );


      System.out.println("Number of classes: " +
model.listNamedClasses().toList().size());


        }

        public Task_OA_1(String city, String pizza) throws FileNotFoundException{

                String pizza_restaurants = "files/PizzaOnto.ttl";
                String output_file = "files/Task_2.4_OA_1_Ontology_Alignment.ttl";
                model = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM
);
      model.read(pizza_restaurants, "RDF/XML" );


            model.setNsPrefix("pizza", pizza);
            model.setNsPrefix("city", city);


            String PizzaRestaurant = "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";
            String pizzaOnto= "http://www.co-ode.org/ontologies/pizza/pizza.owl#";

                String[][] EquivalentClasses = {
             {"Country", "Country"},
             {"Food", "Food"},
             {"Seafood", "FishTopping"},
             {"Meat", "MeatTopping"},
             {"Seafood", "FishTopping"},
             {"Ham", "HamTopping"},
             {"Anchovies", "AnchoviesTopping"},
             {"Onions", "OnionTopping"},
             {"Artichokes", "ArtichokeTopping"},
             {"AmericanPizza", "American"},
             {"Capers", "CaperTopping"},
             {"Cheese", "CheeseTopping"},
             {"Fruit", "FruitTopping"},
             {"Garlic", "GarlicTopping"},
```

```
            {"Olives", "OliveTopping"},
            {"GoatCheese", "GoatsCheeseTopping"},
            {"Gorgonzola", "GorgonzolaTopping"},
            {"GreenPepper", "GreenPepperTopping"},
            {"JalapenoPepper", "JalapenoPepperTopping"},
            {"MargheritaPizza", "Margherita"},
            {"Meat", "MeatTopping"},
            {"MeatPizza", "MeatyPizza"},
            {"Mozzarella", "MozzarellaTopping"},
            {"Mushroom", "MushroomTopping"},
            {"MushroomPizza", "Mushroom"},
            {"NamedPizza", "NamedPizza"},
            {"Pizza", "Pizza"},
            {"Rosemary", "RosemaryTopping"},
            {"Sauce", "SauceTopping"},
            {"Spinach", "SpinachTopping"},
            {"Tomato", "TomatoTopping"},
            {"VegetarianPizza", "VegetarianPizza"},
            {"Pepper", "PepperTopping"},
            {"Chicken", "ChickenTopping"}};



              for (String[] equivalence : EquivalentClasses) {

                    Resource subject_resource =
model.createResource(PizzaRestaurant+equivalence[0]);

              Resource object_resource =
model.createResource(pizzaOnto+equivalence[1]);

              model.add(subject_resource, OWL.equivalentClass, object_resource);


      };

      System.out.println("The graph contains '" + model.listStatements().toSet().size() +
"' Equivalent Classes triples.");

      OutputStream out = new FileOutputStream(output_file);
      RDFDataMgr.write(out, model, RDFFormat.TTL);

      String[][] equivalentSubProperties = {
            {"IsIngredientOf", "IsIngredientOf"},
```

```
            {"HasIngredient", "HasIngredient"}};

                for (String[] equivalence : equivalentSubProperties) {

                                        Resource subject_resource =
model.createResource(PizzaRestaurant+equivalence[0]);

                                Resource object_resource =
model.createResource(pizzaOnto+equivalence[1]);

                                model.add(subject_resource, OWL.equivalentProperty,
object_resource);

                        };


                        System.out.println("The graph contains "' +
model.listStatements().toSet().size() + "' Equivalent Sub Properties triples.");


      OutputStream outt = new FileOutputStream(output_file);
      RDFDataMgr.write(outt, model, RDFFormat.TTL);
   }

        public Set<String> getRDFSLabelsForClass(OntClass cls) {

                final NodeIterator labels = cls.listPropertyValues(RDFS.label);

                Set<String> labels_set =  new HashSet<String>();

                while( labels.hasNext() ) {
                   final RDFNode labelNode = labels.next();
                   final Literal label = labelNode.asLiteral();

                   labels_set.add(label.getString());
                }

                return labels_set;

        }

        public void iterateOverLabels() {

        for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
```

```
            OntClass c = i.next();
            if (!c.isAnon()) {
                    System.out.println(c.getURI() );
                    System.out.println("\t" + c.getLocalName());
                    System.out.println("\t" + getRDFSLabelsForClass(c) );

            }
          }
}

public void iterateOverPizzaOnologyClasses() {

          Set<String> pizza_classes =  new HashSet<String>();
                  for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
            OntClass c = i.next();
            if (!c.isAnon()) {
                    pizza_classes.add(c.getLocalName());



            }

      }
                    System.out.println("Numer of classes in pizza ontology:
"+pizza_classes.size());
          }

public void iterateOverPizzaRestaurantClasses() {

          Set<String> pizza_restaurant_classes =  new HashSet<String>();
                  for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
            OntClass c = i.next();
            if (!c.isAnon()) {

                    pizza_restaurant_classes.add(c.getLocalName());



            }

      }
                    System.out.println("Numer of classes in pizza restaurant:
"+pizza_restaurant_classes.size());
          }

public void saveGraph(Model model, String file_output) throws FileNotFoundException {
```

```
   OutputStream out = new FileOutputStream(file_output);
   RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

        public static void main(String[] args) {


                try {
                String city = "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


      String pizza= "http://www.co-ode.org/ontologies/pizza/pizza.owl#";


      String pizza_ontology = "files/pizza-ontology/pizza.ttl";
      String pizza_restaurants_onotolgy = "files/pizza-restaurants-ontology.ttl";




     Task_OA_1 computeEquivalences = new Task_OA_1(city, pizza);


      computeEquivalences.loadOntologyFromURL(pizza_ontology);
      computeEquivalences.iterateOverPizzaOnologyClasses();
      computeEquivalences.loadOntologyFromURL(pizza_restaurants_onotolgy);
      computeEquivalences.iterateOverPizzaRestaurantClasses();

                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

}
}


package OA;

/*
```

```
* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab8/CompareWithReference.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/


import org.apache.jena.query.Dataset;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model.StmtIterator;
import org.apache.jena.riot.RDFDataMgr;

public class Task_OA_2 {

        public Task_OA_2(String reference_mappings, String system_mappings) {

                Dataset reference_dataset =
RDFDataMgr.loadDataset(reference_mappings);
                Model reference_model = reference_dataset.getDefaultModel();
                StmtIterator iter_ref = reference_model.listStatements();

                Dataset system_dataset = RDFDataMgr.loadDataset(system_mappings);
                Model system_model = system_dataset.getDefaultModel();
                StmtIterator iter_syst = system_model.listStatements();


                int tp=0;
                int fp=0;
                int fn=0;

      try {

        while ( iter_syst.hasNext() ) {
            Statement stmt = iter_syst.next();
            if (reference_model.contains(stmt))
                tp++;
```

```
            else
                fp++;

          }


          while ( iter_ref.hasNext() ) {
              Statement stmt = iter_ref.next();
              if (!system_model.contains(stmt))
                  fn++;

          }
        } finally {
          if ( iter_ref != null ) iter_ref.close();
          if ( iter_syst != null ) iter_syst.close();
        }


        //System.out.println(tp + " " + tp2);
        //System.out.println(fp);
        //System.out.println(fn);
        double precision = (double)tp/(double)(fp+tp);
        double recall = (double)tp/(double)(fn+tp);
        double f_score = (2*precision*recall)/(precision+recall);
        System.out.println("Comparing '" + system_mappings + "' with '" +
reference_mappings);
        System.out.println("\tPrecision: " + precision);
        System.out.println("\tRecall: " + recall);
        System.out.println("\tF-Score: " + f_score);

          }

          public static void main(String[] args) {


                  //Pizza
                  new Task_OA_2("files/reference-mappings-pizza.ttl",
"files/Task_2.4_OA_1_Ontology_Alignment.ttl");

          }

}

package OA;
```

```
/*
* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
* Type of Code: Java code
* Code used: performReasoning(), savegraph()
*
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/



import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.ontology.OntClass;
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.Dataset;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.NodeIterator;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.ResIterator;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
```

```
import org.apache.jena.vocabulary.OWL;
import org.apache.jena.vocabulary.RDFS;

public class Task_OA_3 {

        OntModel model;
        InfModel inf_model;
        public void loadOntologyFromURL(String sourceURL) {

                model = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM
);
      model.read(sourceURL, "RDF/XML" );


      System.out.println("Number of classes: " +
model.listNamedClasses().toList().size());



        }

        public Task_OA_3(String city, String pizza) throws FileNotFoundException{

                String pizza_restaurants = "files/PizzaOnto.ttl";
                String output_file = "files/Task_2.3_OA_1_Ontology_Alignment.ttl";
                model = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM
);
      model.read(pizza_restaurants, "RDF/XML" );


           model.setNsPrefix("pizza", pizza);
           model.setNsPrefix("city", city);


           String PizzaRestaurant = "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";
           String pizzaOnto= "http://www.co-ode.org/ontologies/pizza/pizza.owl#";

                String[][] EquivalentClasses = {
            {"Country", "Country"},
            {"Food", "Food"},
            {"Seafood", "FishTopping"},
            {"Meat", "MeatTopping"},
            {"Seafood", "FishTopping"},
            {"Ham", "HamTopping"},
```

```
            {"Anchovies", "AnchoviesTopping"},
            {"Onions", "OnionTopping"},
            {"Artichokes", "ArtichokeTopping"},
            {"AmericanPizza", "American"},
            {"Capers", "CaperTopping"},
            {"Cheese", "CheeseTopping"},
            {"Fruit", "FruitTopping"},
            {"Garlic", "GarlicTopping"},
            {"Olives", "OliveTopping"},
            {"GoatCheese", "GoatsCheeseTopping"},
            {"Gorgonzola", "GorgonzolaTopping"},
            {"GreenPepper", "GreenPepperTopping"},
            {"JalapenoPepper", "JalapenoPepperTopping"},
            {"MargheritaPizza", "Margherita"},
            {"Meat", "MeatTopping"},
            {"MeatPizza", "MeatyPizza"},
            {"Mozzarella", "MozzarellaTopping"},
            {"Mushroom", "MushroomTopping"},
            {"MushroomPizza", "Mushroom"},
            {"NamedPizza", "NamedPizza"},
            {"Pizza", "Pizza"},
            {"Rosemary", "RosemaryTopping"},
            {"Sauce", "SauceTopping"},
            {"Spinach", "SpinachTopping"},
            {"Tomato", "TomatoTopping"},
            {"VegetarianPizza", "VegetarianPizza"},
            {"Pepper", "PepperTopping"},
            {"Chicken", "ChickenTopping"}};



                for (String[] equivalence : EquivalentClasses) {

                        Resource subject_resource =
model.createResource(PizzaRestaurant+equivalence[0]);

              Resource object_resource =
model.createResource(pizzaOnto+equivalence[1]);

              model.add(subject_resource, OWL.equivalentClass, object_resource);


      };
```

```
        System.out.println("The graph contains "' + model.listStatements().toSet().size() +
"' Equivalent Classes triples.");

        OutputStream out = new FileOutputStream(output_file);
        RDFDataMgr.write(out, model, RDFFormat.TTL);

        String[][] equivalentSubProperties = {
                {"IsIngredientOf", "IsIngredientOf"},
            {"HasIngredient", "HasIngredient"}};

                for (String[] equivalence : equivalentSubProperties) {

                                        Resource subject_resource =
model.createResource(PizzaRestaurant+equivalence[0]);

                                Resource object_resource =
model.createResource(pizzaOnto+equivalence[1]);

                                model.add(subject_resource, OWL.equivalentProperty,
object_resource);

                        };


                        System.out.println("The graph contains "' +
model.listStatements().toSet().size() + "' Equivalent Sub Properties triples.");


        OutputStream outt = new FileOutputStream(output_file);
        RDFDataMgr.write(outt, model, RDFFormat.TTL);
    }

          public Set<String> getRDFSLabelsForClass(OntClass cls) {

                final NodeIterator labels = cls.listPropertyValues(RDFS.label);

                Set<String> labels_set =  new HashSet<String>();

                while( labels.hasNext() ) {
                   final RDFNode labelNode = labels.next();
                   final Literal label = labelNode.asLiteral();

                   labels_set.add(label.getString());
                }
```

```
                return labels_set;

        }

        public void iterateOverLabels() {

        for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
          OntClass c = i.next();
          if (!c.isAnon()) {
                  System.out.println(c.getURI() );
                  System.out.println("\t" + c.getLocalName());
                  System.out.println("\t" + getRDFSLabelsForClass(c) );

          }
        }
}

public void iterateOverPizzaOnologyClasses() {

        Set<String> pizza_classes =  new HashSet<String>();
                for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
          OntClass c = i.next();
          if (!c.isAnon()) {
                  pizza_classes.add(c.getLocalName());


          }

    }
                  System.out.println("Numer of classes in pizza ontology:
"+pizza_classes.size());
        }

public void iterateOverPizzaRestaurantClasses() {

        Set<String> pizza_restaurant_classes =  new HashSet<String>();
                for (Iterator<OntClass> i =  model.listClasses(); i.hasNext(); ) {
          OntClass c = i.next();
          if (!c.isAnon()) {

                  pizza_restaurant_classes.add(c.getLocalName());
```

```
        }

    }
                System.out.println("Numer of classes in pizza restaurant:
"+pizza_restaurant_classes.size());
        }


        public void performReasoning(String ontology_file) throws
FileNotFoundException {


                System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");


            Dataset dataset = RDFDataMgr.loadDataset(ontology_file);
            model.add(dataset.getDefaultModel().listStatements().toList());


            System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

            Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();
                inf_model = ModelFactory.createInfModel(reasoner, model);


            System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");

            String output_file = "files/Task_2.4_OA_3_Reasoner_Output.ttl";
            OutputStream outt = new FileOutputStream(output_file);
       RDFDataMgr.write(outt, model, RDFFormat.TTL);

        }


public void saveGraph(Model model, String file_output) throws FileNotFoundException {


   OutputStream out = new FileOutputStream(file_output);
   RDFDataMgr.write(out, model, RDFFormat.TURTLE);

 }
```

```
	public static void main(String[] args) {


		try {
		String city = "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


	String pizza= "http://www.co-ode.org/ontologies/pizza/pizza.owl#";


	String pizza_ontology = "files/pizza-ontology/pizza.ttl";
	String pizza_restaurants_onotolgy = "files/pizza-restaurants-ontology.ttl";
	String output_file = "files/output_file.ttl";



	Task_OA_3 computeEquivalences = new Task_OA_3(city, pizza);


	computeEquivalences.loadOntologyFromURL(pizza_ontology);
	computeEquivalences.iterateOverPizzaOnologyClasses();
	computeEquivalences.loadOntologyFromURL(pizza_restaurants_onotolgy);
	computeEquivalences.iterateOverPizzaRestaurantClasses();

computeEquivalences.performReasoning("files/Task_2.4_OA_1_Ontology_Alignment.ttl
");


		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}


}
}

package OA;


/*
* Created by @author ernesto
```

```
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/



import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
```

```
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_OA_4 {

        String input_file;
        Model model;
        InfModel inf_model;

        String cw_ns_str;

        List<String[]> csv_file;


    Map<String, String> stringToURI = new HashMap<String, String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_OA_4(String input_file, Map<String, Integer> column_index) throws
IOException {


                this.input_file = input_file;

                this.column_index = column_index;


                model = ModelFactory.createDefaultModel();

       cw_ns_str= "http://www.semanticweb.org/city/in3067-inm713/2024/restaurants#";


                model.setNsPrefix("cw", cw_ns_str);
                model.setNsPrefix("xsd", "http://www.w3.org/2001/XMLSchema#");
                model.setNsPrefix("dbr", "http://dbpedia.org/resource/");
```

```
            CSVReader reader = new CSVReader(new FileReader(input_file));
        csv_file = reader.readAll();
        reader.close();


    }

public void performTaskRDF() throws JsonProcessingException, IOException,
URISyntaxException {
   CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


            //Type Triples
            mappingToCreateTypeTriple(column_index.get("restaurant"), cw_ns_str +
"Restaurant", useExternalURI);
            mappingToCreateTypeTriple(column_index.get("address"), cw_ns_str +
"Address", useExternalURI);
            mappingToCreateTypeTriple(column_index.get("city"), cw_ns_str + "City",
useExternalURI);
            mappingToCreateTypeTriple(column_index.get("country"), cw_ns_str +
"Country", useExternalURI);
            mappingToCreateTypeTriple(column_index.get("postcode"), cw_ns_str +
"Postcode", useExternalURI);
            mappingToCreateTypeTriple(column_index.get("state"), cw_ns_str +
"State", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("menu_item"), cw_ns_str +
"Food", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("menu_item"), cw_ns_str +
"MenuItem", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("item_value"), cw_ns_str +
"ItemValue", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("currency"), cw_ns_str +
"Currency", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("item_description"), cw_ns_str
+ "Ingredient", useExternalURI);
        mappingToCreateTypeTriple(column_index.get("categories"), cw_ns_str +
"Categories", useExternalURI);
```

```
		//Literal Triples
		mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);
		mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode", XSDDatatype.XSDstring);

		mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount", XSDDatatype.XSDstring);
		mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName", XSDDatatype.XSDstring);

		mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount", XSDDatatype.XSDdouble);



		//Object Triples
		mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");
		mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");
		mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");
		mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
		mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");


		mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str + "servedInRestaurant");
		mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");
		mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_description"), cw_ns_str + "hasIngredient");


		mappingToCreateObjectTriple(column_index.get("item_description"),
column_index.get("menu_item"), cw_ns_str + "isIngredientOf");

		mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");
```

```
            mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");
            mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");

            mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");
            mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");

            mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

            mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");



}

protected String processLexicalName(String restaurant) {


        return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


   stringToURI.put(restaurant, cw_ns_str + processLexicalName(restaurant));


   return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int subject_column_index, String
class_type_uri, boolean useExternalURI) throws JsonProcessingException,
IOException, URISyntaxException {

        for (String[] row : csv_file) {
```

```
                if (row.length<column_index.size())
                        continue;


                String subject = row[subject_column_index].toLowerCase();
                String subject_uri;


                if (stringToURI.containsKey(subject))
                        subject_uri=stringToURI.get(subject);
        else
           subject_uri=createURIForEntity(subject, useExternalURI);


                //TYPE TRIPLE
                Resource subject_resource = model.createResource(subject_uri);
                Resource type_resource = model.createResource(class_type_uri);

                model.add(subject_resource, RDF.type, type_resource);

          }

}

private boolean is_nan(String value) {
   return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column, int object_column,
String predicate, XSDDatatype datatype) {

          for (String[] row : csv_file) {


                if (row.length<column_index.size())
                        continue;



                String subject = row[subject_column];
```

```
                String lit_value = row[object_column];

                if (is_nan(lit_value))
                        continue;


        String entity_uri = stringToURI.get(subject.toLowerCase());


        Resource subject_resource = model.createResource(entity_uri);
        Property predicate_resource = model.createProperty(predicate);

                //Literal
        Literal lit = model.createTypedLiteral(lit_value, datatype);


                model.add(subject_resource, predicate_resource, lit);

          }


}


protected void mappingToCreateObjectTriple(int subject_column, int object_column,
String predicate) {

          for (String[] row : csv_file) {


                if (row.length<column_index.size())
                        continue;

                String subject = row[subject_column];
                String object = row[object_column];

                if (is_nan(object))
                        continue;


        String subject_uri = stringToURI.get(subject.toLowerCase());
        String object_uri = stringToURI.get(object.toLowerCase());
```

```
        //New triple
        Resource subject_resource = model.createResource(subject_uri);
        Property predicate_resource = model.createProperty(predicate);
        Resource object_resource = model.createResource(object_uri);


                model.add(subject_resource, predicate_resource, object_resource);

          }


}


public void performSPARQLQuery(Model model, String file_query_out) {

        WriteFile writer = new WriteFile(file_query_out);



   String queryStr =
                    "PREFIX cw: <http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
                     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
                     "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
                     "SELECT ?country ?menu_item WHERE {\n" +
                     "?country rdf:type cw:Country .\n"+
                     "?menu_item cw:hasIngredient ?Ingredient .\n"+"}";


    Query q = QueryFactory.create(queryStr);

        QueryExecution qe =
                        QueryExecutionFactory.create(q, model);
        try {
                ResultSet res = qe.execSelect();

                int solutions = 0;

                while( res.hasNext()) {
                        solutions++;
                        QuerySolution soln = res.next();
                        RDFNode country = soln.get("?country");
                        RDFNode menu_item = soln.get("?menu_item");
```

```
                        writer.writeLine(country.toString()+","+menu_item.toString());

                }
                System.out.println(solutions + " results satisfying the query.");

        } finally {
                qe.close();
        }

        writer.closeBuffer();


}


public void performReasoning(String ontology_file) {


        System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");


   Dataset dataset = RDFDataMgr.loadDataset(ontology_file);
   model.add(dataset.getDefaultModel().listStatements().toList());


   System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

   Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();
        inf_model = ModelFactory.createInfModel(reasoner, model);


   System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}


public void saveGraph(Model model, String file_output) throws FileNotFoundException {
```

```
    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

        String file = "files/50_unique_data.csv";

        //Format
        //restaurant address city country postcode state categories menu_item item_value currency item_description
        Map<String, Integer> column_index = new HashMap<String, Integer>();
        column_index.put("restaurant", 0);
        column_index.put("address", 1);
        column_index.put("city", 2);
        column_index.put("country", 3);
        column_index.put("postcode", 4);
        column_index.put("state", 5);
        column_index.put("categories", 6);
        column_index.put("menu_item", 7);
        column_index.put("item_value", 8);
        column_index.put("currency", 9);
        column_index.put("item_description", 10);

        try {
                Task_OA_4 solution = new Task_OA_4(file, column_index);

                String task = "Task_OA_4";

                if (task.equals("Task_OA_4"))
                        solution.performTaskRDF();


                solution.performReasoning("files/pizza-ontology/pizza.ttl");


                solution.saveGraph(solution.inf_model, file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

                solution.performSPARQLQuery(solution.inf_model, file.replace("50_unique_data.csv", ""+task)+"-Sparql-query-results.csv");
```

```
		} catch (Exception e) {
			e.printStackTrace();
		}




}
}

/*******************************************************************
****************
 * Copyright 2018 by The Alan Turing Institute
 *

*******************************************************************
***************/
package RDF;

/**
 *
 * Class to connect to the public DBpedia SPARQL endpoint. See
contained
 * datasets and more information here:
http://wiki.dbpedia.org/public-sparql-endpoint
 *
* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/DBpediaEndpoint.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/


public class DBpediaEndpoint extends SPARQLEndpointService {


	@Override
```

```
     public String getENDPOINT() {
          return "https://dbpedia.org/sparql";
     }




     /**
      * To extract a portion of dbpedia relevant to the subject
      * @param uri_subject
      * @return
      */
     protected String createSPARQLQueryForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT ?p ?o \n"
                    + "WHERE { <" + uri_subject + "> ?p ?o . "
                    + "FILTER (?p !=
<http://dbpedia.org/ontology/wikiPageWikiLink> "
                    + "&& ?p != <http://www.w3.org/2000/01/rdf-
schema#comment> "
                    + "&& ?p !=
<http://dbpedia.org/ontology/abstract>)"
                    + "}";

     }



     /**
      * To extract a portion of dbpedia relevant to the object
      * @param uri_subject
      * @return
      */
     protected String createSPARQLQueryForObject(String
uri_object){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT ?s ?p \n"
                    + "WHERE { ?s ?p <" + uri_object + "> . "
                    + "FILTER (?p !=
<http://dbpedia.org/ontology/wikiPageWikiLink> "
                    + "&& ?p != <http://www.w3.org/2000/01/rdf-
schema#comment> "
```

```
                    + "&& ?p !=
<http://dbpedia.org/ontology/abstract>)"
                    + "}";

     }



     /**
      * To extract class types of the subject
      * @param uri_subject
      * @return
      */
     protected String createSPARQLQuery_TypesForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?uri \n"
                    + "WHERE { <" + uri_subject + ">
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?uri . "
                    + "}";

     }


     protected String
createSPARQLQuery_AllTypesForSubject(String uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?uri \n"
                    + "WHERE {\n"
                    + "{<" + uri_subject + ">
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?dt . "
                    + "?dt <http://www.w3.org/2000/01/rdf-
schema#subClassOf>* ?uri "
                    + "}\n"
                    + "UNION \n{"
                    + "<" + uri_subject + ">
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?uri . "
//direct types
                    + "}\n"
                    + "UNION \n{"
                    + "<" + uri_subject + ">
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?dt . "
```

```
                    + "?dt
<http://www.w3.org/2002/07/owl#equivalentClass> ?uri "
                    + "}\n"
                    + "}";

     }


     protected String
createSPARQLQuery_AllSuperClassesForSubject(String uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?uri \n"
                    + "WHERE {\n"
                    + "{<" + uri_subject + ">
<http://www.w3.org/2000/01/rdf-schema#subClassOf>* ?uri "
                    + "}\n"
                    + "UNION \n{"
                    + "<" + uri_subject + ">
<http://www.w3.org/2002/07/owl#equivalentClass> ?uri "
                    + "}\n"
                    + "}";

     }




     protected String
craeteSPARQLQuery_TypeObjectsForPredicate(String uri_predicate){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?uri \n"
                    + "WHERE { ?s <" + uri_predicate + "> ?o . "
                    + "?o <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> ?uri ."
                    + "}";

     }
```

```
     public static void main(String[] args) {

          String uri_subject;
          uri_subject = "http://dbpedia.org/resource/London";


          DBpediaEndpoint dbe = new DBpediaEndpoint();

          try {

               System.out.println("Direct types for: " +
uri_subject);

     System.out.println(dbe.getTypesForSubject(uri_subject).size
() + " " + dbe.getTypesForSubject(uri_subject));
               for (String type :
dbe.getTypesForSubject(uri_subject)){
                    if
(type.startsWith("http://dbpedia.org/ontology"))
                         System.out.println("\t"+type);
               }

               System.out.println("All types for: " +
uri_subject);

     System.out.println(dbe.getAllTypesForSubject(uri_subject).s
ize() + " " + dbe.getAllTypesForSubject(uri_subject));
               for (String type :
dbe.getAllTypesForSubject(uri_subject)){
                    if
(type.startsWith("http://dbpedia.org/ontology"))
                         System.out.println("\t"+type);
               }


          //   }
          } catch (Exception e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }


     }
```

```
}
/*****************************************************************
****************
 *
 *  Copyright 2018 by The Alan Turing Institute
 *
 *

*****************************************************************
***************/
package RDF;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import org.apache.http.client.utils.URIBuilder;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;



/**
 *
 * Class to access the DBpedia look up REST API:
https://github.com/dbpedia/lookup
* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/DBpediaLookup.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
```

```
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/


public class DBpediaLookup extends LookupService{

     //Old one
     //private final String REST_URL =
"http://lookup.dbpedia.org/api/search/KeywordSearch?";
     //Old parameters:
     //private final String MaxHits = "MaxHits";
     //private final String QueryString = "QueryString";
     //private final String QueryClass = "QueryClass";

     //private final String REST_URL =
"http://akswnc7.informatik.uni-leipzig.de/lookup/api/search?";
     //private final String REST_URL =
"http://lookup.dbpedia.org/api/search?";
     private final String REST_URL =
"https://lookup.dbpedia.org/api/search?";

     private final String MaxHits = "maxResults";
     private final String QueryString = "query";
     private final String QueryClass = "typeName";
     private final String Format = "format";
     //Not supported
     //private final String language = "language";


     private int hits=5;
     //private String query;







     public Set<KGEntity> getKGEntities(String query) throws
JsonProcessingException, IOException, URISyntaxException{
          return getKGEntities(query,"", hits);
     }
```

```
     public Set<KGEntity> getKGEntities(String query, int
max_hits) throws JsonProcessingException, IOException,
URISyntaxException{
          return getKGEntities(query,"", max_hits);
     }

     public Set<KGEntity> getKGEntities(String query, String
type) throws JsonProcessingException, IOException,
URISyntaxException{
          return getKGEntities(query,type, hits);
     }



     /**
      * Return a Map with "key"=URIs containing the DBPedia
entities related to the query string, and "values"=sets of
(ontology) class uris
      * @param query
      * @return
      * @throws IOException
      * @throws JsonProcessingException
      * @throws URISyntaxException
      */
     public Set<KGEntity> getKGEntities(String query, String
cls_type, int max_hits) throws JsonProcessingException,
IOException, URISyntaxException{


          Set<KGEntity> entities = new HashSet<KGEntity>();

          //String urlToGet = REST_URL + QueryClass + "=" +
cls_type + "&" + MaxHits + "="+ max_hits + "&" + QueryString +
"=" + query;
          URL urlToGet = buildRequestURL(query, cls_type,
max_hits);

          //System.out.println(urlToGet);

          JsonNode results = jsonToNode(getRequest(urlToGet));


          //System.out.println(results);

          if (results.has("docs")){
```

```
                for (JsonNode result : results.get("docs")){

                     //System.out.println(result);

                     if (result.has("resource")) {//expected only
one

                          KGEntity entity = new KGEntity();

                          for (JsonNode uri :
result.get("resource")) {
                               entity.setId(uri.asText());
                          }

                          if (result.has("type")) {
                               for (JsonNode cls :
result.get("type")){

                                    if
(!cls.asText().equals("http://www.w3.org/2002/07/owl#Thing"))

     entity.addType(cls.asText());

                               }
                          }

                          if (result.has("label")) {//expected
only one

                               for (JsonNode label :
result.get("label")) {

     entity.setName(label.asText());
                               }
                          }

                          if (result.has("comment")) {//expected
only one

                               for (JsonNode comment :
result.get("comment")) {

     entity.setDescription(comment.asText());
                               }
                          }
```

```
                        if (result.has("score")) {//expected
only one

                             for (JsonNode score :
result.get("score")) {

     entity.setScore(score.asDouble());
                             }
                        }


                        entities.add(entity);

                   }
              }
         }

         return entities;




     }



     protected String getREST_URL() {
          return REST_URL;
     }


     protected URL buildRequestURL(String query, String
cls_type, int max_hits) throws URISyntaxException,
MalformedURLException{
          URIBuilder ub = new URIBuilder(getREST_URL());
          if (cls_type!=null && !cls_type.equals(""))
               ub.addParameter(QueryClass, cls_type);

          ub.addParameter(MaxHits, String.valueOf(max_hits));
          ub.addParameter(QueryString, query);
          ub.addParameter(Format, "json");  //neded in new API

          return ub.build().toURL();
     }
```

```
     public static void main(String[] args){
          DBpediaLookup lookup = new DBpediaLookup();

          String keywords;
          //keywords="Chicago Bulls";
          //keywords="Congo";
          keywords="City University London";


          try {

               //Look up for entities matching the string
"keywords"
               Set<KGEntity> entities =
lookup.getKGEntities(keywords);
               System.out.println("NUmber of candidates found: "
+ entities.size());
               for (KGEntity entity : entities){
                    System.out.println(entity);
               }
          } catch (JsonProcessingException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          } catch (IOException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          } catch (URISyntaxException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }

     }


}

package RDF;
```

```
/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/GoogleKGLookup.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import com.google.api.client.http.GenericUrl;
import com.google.api.client.http.HttpRequest;
import com.google.api.client.http.HttpRequestFactory;
import com.google.api.client.http.HttpResponse;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.http.javanet.NetHttpTransport;


import com.jayway.jsonpath.JsonPath;

import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;




public class GoogleKGLookup {

     //https://github.com/schemaorg/schemaorg
     //Doc: https://developers.google.com/knowledge-
graph/reference/rest/v1/
     //https://developers.google.com/knowledge-graph/
     //https://console.developers.google.com/apis/api/kgsearch.g
oogleapis.com/credentials?project=sem-tab
     String API_key = "AIzaSyA6Bf9yuMCCPh7vpElzrfBvE2ENCVWr-84";
```

```
     public TreeSet<KGEntity> getEntities(String query, String
num_hits, Set<String> types, Set<String> languages, double
minScore) {

          TreeSet<KGEntity> entities = new TreeSet<KGEntity>();

          try {

                HttpTransport httpTransport = new
NetHttpTransport();
                HttpRequestFactory requestFactory =
httpTransport.createRequestFactory();
                JSONParser parser = new JSONParser();
                GenericUrl url = new
GenericUrl("https://kgsearch.googleapis.com/v1/entities:search")
;
                url.put("query", query);
                url.put("limit", num_hits);
                url.put("indent", "true");

                url.put("types", types);

                url.put("languages", languages);

                url.put("key", API_key);

                //System.out.println(url);


                HttpRequest request =
requestFactory.buildGetRequest(url);
                HttpResponse httpResponse = request.execute();
                JSONObject response = (JSONObject)
parser.parse(httpResponse.parseAsString());
                JSONArray elements = (JSONArray)
response.get("itemListElement");

                double score;


                System.out.println("Number of canddiate
elements: " + elements.size());

                System.out.println(elements);
```

```
                for (Object element : elements) {


                 score =
Double.parseDouble(JsonPath.read(element,
"$.resultScore").toString());

                 if (score>minScore){ //filter by google score.

                      KGEntity entity = new KGEntity();

                      entity.setId(JsonPath.read(element,
"$.result.@id").toString());
                      entity.setName(JsonPath.read(element,
"$.result.name").toString());

entity.setDescription(JsonPath.read(element,
"$.result.description").toString());

                      for (Object type :
(JSONArray)JsonPath.read(element, "$.result.@type")) {
                           if (!type.toString().equals("Thing"))
                                entity.addType(type.toString());
                      }

                      entity.setScore(score);


//System.out.println(JsonPath.read(element,
"$.result.@id").toString());

//System.out.println(JsonPath.read(element,
"$.result.name").toString());

//System.out.println(JsonPath.read(element,
"$.result.description").toString());


///System.out.println(JsonPath.read(element,
"$.result.@type").toString());


                      entities.add(entity);
                 }

                }
```

```
            } catch (Exception ex) {
              ex.printStackTrace();

            }



        for (KGEntity entity : entities){

             System.out.println(entity.toString());

        }



        return entities;

    }



      public static void main(String[] args) {

           Set<String> types = new HashSet<String>();
           types.add("Person");
           types.add("Event");

           String query;
           query ="Chicago Bulls";
           query = "Congo";

           GoogleKGLookup lk = new GoogleKGLookup();

           lk.getEntities(query, "10", types,
Collections.emptySet(), 0.0);


      }
```

```
}

package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-2024/blob/main/src/main/java/lab5/KGEntity.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/



import java.util.HashSet;
import java.util.Set;

public class KGEntity implements Comparable<KGEntity> {


     private String id;
     private String name;
     private String description; //very precise type
     private Set<String> types=new HashSet<String>(); //from schema.org

     private double score;



     public String getId() {
          return id;
     }
     public void setId(String id) {
          this.id = id;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public String getDescription() {
```

```
          return description;
     }
     public void setDescription(String description) {
          this.description = description;
     }
     public Set<String> getTypes() {
          return types;
     }


     protected String getTypesStr() {
          String types_str = "";

          for (String type: types)
               types_str += type + ";";
          return types_str;
     }

     public void setTypes(Set<String> types) {
          this.types = types;
     }


     public void addType(String type) {
          this.types.add(type);
     }


     public double getScore() {
          return score;
     }
     public void setScore(double score) {
          this.score = score;
     }


     public String toString() {

          StringBuilder sb = new StringBuilder();


     sb.append(getId()).append("\n\t").append(getName()).append(
"\n\t").append(getDescription()).append("\n\t").append(getTypesS
tr()).append("\n\t").append(getScore());;

          return sb.toString();
```

```
}




public boolean equals(Object o){

     if  (o == null)
          return false;
     if (o == this)
          return true;
     if (!(o instanceof KGEntity))
          return false;

     KGEntity i =  (KGEntity)o;

     return equals(i);

}


public boolean equals(KGEntity m){

     if (!getId().equals(m.getId())){
          return false;
     }
     return true;
}



public  int hashCode() {
       int code = 10;
       code = 40 * code + getId().hashCode();
       code = 50 * code + getName().hashCode();
       return code;
}


public int compareTo(KGEntity e){

     if (equals(e))
          return 0;
```

```
          //Otherwise alphabetically
          //if (getName().compareTo(e.getName())>0)
          //Or hits
          if (getScore()>e.getScore())
               return -1;
          else
               return 1;
     }









}
```

```
package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/LookupService.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.Set;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
public abstract class LookupService {


private final ObjectMapper mapper = new ObjectMapper();



     protected JsonNode jsonToNode(String json) throws
JsonProcessingException, IOException {

          return mapper.readTree(json);

     }



     protected HttpURLConnection getConnection(URL urlToGet)
throws IOException {

          URL url;
          HttpURLConnection conn;

          //url = new URL(urlToGet);
          url = urlToGet;
          conn = (HttpURLConnection) url.openConnection();

          conn.setRequestMethod("GET");
          //conn.setRequestProperty("Authorization", "apikey
token="
          //        + API_KEY_Ernesto);
          conn.setRequestProperty("Accept", "application/json");

          return conn;

     }


     protected String getRequest(URL urlToGet) throws
IOException {

          HttpURLConnection conn;
          BufferedReader rd;
          String line;
          String result = "";
```

```
          //In some cases it fails the connection. Try several
times
          boolean success=false;
          int attempts=0;


          //TODO how many attempts?
          //while(!success && attempts<25){
          while(!success && attempts<3){


               attempts++;

               try{
                    conn = getConnection(urlToGet);

                    rd = new BufferedReader(new
InputStreamReader(conn.getInputStream()));

                    while ((line = rd.readLine()) != null) {
                         result += line;
                    }
                    rd.close();

                    if (!result.isEmpty())
                         success=true;
               }

               catch(IOException e){
                    System.out.println("Error accessing: " +
urlToGet + "  Attempt: " + attempts);
               }

          }

          if (!success)
               throw new IOException(); //We throw error to
check next page
          else if (attempts>1)
               System.out.println("SUCCESS accessing: " +
urlToGet + "  Attempt: " + attempts);

          return result;
     }
```

```
    //protected abstract URL buildRequestURL(String query,
String cls_type, int max_hits, String language) throws
URISyntaxException, MalformedURLException;

    protected abstract String getREST_URL();

    public abstract Set<KGEntity> getKGEntities(String query)
throws JsonProcessingException, IOException, URISyntaxException;
}

package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/SPARQLEndpointService.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/



import java.util.HashSet;

import java.util.Set;
import java.util.concurrent.TimeUnit;

import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;

public abstract class SPARQLEndpointService {
```

```
     public abstract String getENDPOINT();


     //Query to retrieve predicates and objects for subject
     protected abstract String
createSPARQLQueryForSubject(String uri_subject);

     //Query to retrieve predicates and subjects for object
     protected abstract String createSPARQLQueryForObject(String
uri_object);


     protected abstract String
craeteSPARQLQuery_TypeObjectsForPredicate(String uri_predicate);


     protected abstract String
createSPARQLQuery_TypesForSubject(String uri_resource);


     protected abstract String
createSPARQLQuery_AllTypesForSubject(String uri_resource);




     protected abstract String
createSPARQLQuery_AllSuperClassesForSubject(String
uri_resource);




     protected String createSPARQLQuery_LabelForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?value \n"
                    + "WHERE { <" + uri_subject + ">
<http://www.w3.org/2000/01/rdf-schema#label> ?value . "
                    + "}";

     }
```

```
     public Set<String> getTypesOfObjectForPredicate(String
uri_predicate) throws Exception{

          return getURIsForQuery(

     craeteSPARQLQuery_TypeObjectsForPredicate(uri_predicate));

     }




     public Set<String> getLabelsForSubject(String uri_resource)
throws Exception{

          return getValuesForQuery(

     createSPARQLQuery_LabelForSubject(uri_resource));


     }


     public Set<String> getTypesForSubject(String uri_resource)
throws Exception{

          return getURIsForQuery(

     createSPARQLQuery_TypesForSubject(uri_resource));


     }



     public Set<String> getAllTypesForSubject(String
uri_resource) throws Exception{

          return getURIsForQuery(

     createSPARQLQuery_AllTypesForSubject(uri_resource));


     }
```

```
     public Set<String> getAllSuperClassesForSubject(String
uri_resource) throws Exception{

          return getURIsForQuery(

     createSPARQLQuery_AllSuperClassesForSubject(uri_resource));


     }




     public Set<String> getURIsForQuery(String query) throws
Exception{


          Set<String> types = new HashSet<String>();


          //Query to retrieve predicates and objects for subject
          Query q = QueryFactory.create(query);

          //System.out.println(query);


          //In some cases it fails the connection. Try several
times
          boolean success=false;
          int attempts=0;

          while(!success && attempts<3){

               attempts++;

               QueryExecution qe =
QueryExecutionFactory.sparqlService(getENDPOINT(), q);
               try {
                    ResultSet res = qe.execSelect();

                    while( res.hasNext()) {
```

```
                        QuerySolution soln = res.next();

                        //RDFNode object_type = soln.get("?t");
                        Resource object_type =
soln.getResource("?uri");
                        //System.out.println(""+object_type);


                        //System.out.println(object_type);
                        if (object_type!=null)

     types.add(object_type.getURI().toString());

                    }

                    success=true;

               }
               catch (Exception e) {
                    System.out.println("Error accessing " +
getENDPOINT() + " with  SPARQL:\n" + query + "  Attempt: " +
attempts);
                    //e.printStackTrace();
                    System.out.println(e.getLocalizedMessage());
                    TimeUnit.MINUTES.sleep(1); //wait 1 minute
and try again
               }
               finally {
                    qe.close();
               }

          }
          if (!success)
               throw new Exception();
          else if (attempts>1)
               System.out.println("SUCCESS accessing SPARQL\n: "
+ query + "  Attempt: " + attempts);

          return types;

     }



     public Set<String> getValuesForQuery(String query) throws
Exception{
```

```
          Set<String> types = new HashSet<String>();


          //Query to retrieve predicates and objects for subject
          Query q = QueryFactory.create(query);

          //System.out.println(query);


          //In some cases it fails the connection. Try several
times
          boolean success=false;
          int attempts=0;

          while(!success && attempts<3){

               attempts++;

               QueryExecution qe =
QueryExecutionFactory.sparqlService(getENDPOINT(), q);
               try {
                    ResultSet res = qe.execSelect();

                    while( res.hasNext()) {

                         QuerySolution soln = res.next();

                         Literal object_type =
soln.getLiteral("?value");
                         //System.out.println(""+object_type);

                         if (object_type!=null)

     types.add(object_type.getValue().toString()); //TODO
language?

                    }

                    success=true;

               }
               catch (Exception e) {
```

```
                    System.out.println("Error accessing " +
getENDPOINT() + " with  SPARQL:\n" + query + "  Attempt: " +
attempts);
                    e.printStackTrace();
                    TimeUnit.MINUTES.sleep(1); //wait 1 minute
and try again
               }
               finally {
                    qe.close();
               }

          }
          if (!success)
               throw new Exception();
          else if (attempts>1)
               System.out.println("SUCCESS accessing SPARQL\n: "
+ query + "  Attempt: " + attempts);

          return types;

     }



     //TimeUnit.SECONDS.sleep(1);







     public Set<Statement> getTriplesForSubject(String
uri_subject) throws Exception{

          Set<Statement> triples = new HashSet<Statement>();

          Model model = ModelFactory.createDefaultModel();

          //subject
          Resource subject = model.createResource(uri_subject);


          //Query to retrieve predicates and objects for subject
```

```
          String query =
createSPARQLQueryForSubject(uri_subject);
          Query q = QueryFactory.create(query);


          //In some cases it fails the connection. Try several
times
          boolean success=false;
          int attempts=0;


          while(!success && attempts<3){

               attempts++;

               QueryExecution qe =
QueryExecutionFactory.sparqlService(getENDPOINT(), q);
               try {
                    ResultSet res = qe.execSelect();

                    while( res.hasNext()) {

                         QuerySolution soln = res.next();
                         RDFNode predicate = soln.get("?p");
                         RDFNode object = soln.get("?o");
                         //System.out.println(""+predicate + " "
+ object);


     triples.add(model.createStatement(subject,
model.createProperty(predicate.toString()), object));
                    }

                    success=true;

               }
               catch (Exception e){
                    System.out.println("Error accessing " +
getENDPOINT() + " with  SPARQL:\n" + query + "  Attempt: " +
attempts);
                    TimeUnit.MINUTES.sleep(1); //wait 1 minute
and try again
               }
               finally {
                    qe.close();
               }
```

```
          }

          if (!success)
               throw new Exception();
          else if (attempts>1)
               System.out.println("SUCCESS accessing SPARQL\n: "
+ query + "  Attempt: " + attempts);



          return triples;

     }


     public Set<Statement> getTriplesForObject(String
uri_object) throws Exception{

          Set<Statement> triples = new HashSet<Statement>();

          Model model = ModelFactory.createDefaultModel();

          //subject
          Resource object = model.createResource(uri_object);


          //Query to retrieve predicates and subjects for object
          String query = createSPARQLQueryForObject(uri_object);
          Query q = QueryFactory.create(query);


          //In some cases it fails the connection. Try several
times
          boolean success=false;
          int attempts=0;


          while(!success && attempts<3){

               attempts++;

               QueryExecution qe =
QueryExecutionFactory.sparqlService(getENDPOINT(), q);
               try {
                    ResultSet res = qe.execSelect();
                    while( res.hasNext()) {
```

```
                         QuerySolution soln = res.next();
                         RDFNode subject = soln.get("?s");
                         RDFNode predicate = soln.get("?p");

                         //System.out.println(""+predicate + " "
+ object);


     triples.add(model.createStatement(subject.asResource(),
model.createProperty(predicate.toString()), object));
                    }

                    success=true;

               }
               catch (Exception e){
                    System.out.println("Error accessing " +
getENDPOINT() + " with  SPARQL:\n" + query + "  Attempt: " +
attempts);
                    TimeUnit.MINUTES.sleep(1); //wait 1 minute
and try again
               }
               finally {
                    qe.close();
               }
          }

          if (!success)
               throw new Exception();
          else if (attempts>1)
               System.out.println("SUCCESS accessing SPARQL\n: "
+ query + "  Attempt: " + attempts);



          return triples;

     }


}

package RDF;

/* Created by @author ernesto
```

```
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
* https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Wikidatalookup.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
```

```
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_RDF_3_RDF_Triples_using_GoogleKG {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     GoogleKGLookup googleKGLookup;

     List<String[]> csv_file;

     //Dictionary that keeps the URIs. Specially useful if
accessing a remote service to get a candidate URI  to avoid
repeated calls
    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();



public Task_RDF_3_RDF_Triples_using_GoogleKG(String input_file,
Map<String, Integer> column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          //1. GRAPH INITIALIZATION

         //Empty graph
          model = ModelFactory.createDefaultModel();
```

```
          //Note that this is the same namespace used in the
ontology "pizza-restaurant-ontology.ttl"
        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


        //Prefixes for the serialization
          model.setNsPrefix("cw", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://googleKGLookup.org/resource/");


        //Load data in matric to later use an iterator
          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();

         googleKGLookup = new GoogleKGLookup();
     }

public void performTaskRDFusingGoogleKG() throws
JsonProcessingException, IOException, URISyntaxException {
    CovertCSVToRDF(true);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


     //In a large ontology one would need to find a more
automatic way to use the ontology vocabulary.
    //e.g.,  via matching. In a similar way as we match entities
to a large KG like DBPedia or Wikidata
    //Since we are dealing with very manageable ontologies, we
can integrate their vocabulary
    //within the code. E.g.,: cw_ns_str + City



    //We modularize the transformation to RDF. The
transformation is tailored to the given table, but
```

```
    //he individual components/mappings are relatively generic
(especially type and literal triples).

    //Mappings may required one or more columns as input and
create 1 or more triples for an entity



     //We give subject column and target type
     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "state", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "menu_item", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "Itemvalue", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "categories", useExternalURI);




    //We give subject and object columns (they could be the
same), predicate and datatype

mappingToCreateLiteralTriple(column_index.get("item_description"
), column_index.get("item_description"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);
```

```
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);




    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "serves");

    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "containsAdress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "containsCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "containsState");

    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");
    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");




}
```

```
protected String processLexicalName(String name) {

     //Remove potential spaces and other characters not allowed
in URIs

    //This method may need to be extended
     //Other problematic characters:
    //{", "}", "|", "\", "^", "~", "[", "]", and "`"

     return name.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String name, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

    //We create fresh URI (default option)
    stringToURI.put(name, cw_ns_str + processLexicalName(name));

    if (useExternalURI) {//We connect to online KG
        String uri = getExternalKGURI(name);
        if (!uri.equals(""))
          stringToURI.put(name, uri);
    }

    return stringToURI.get(name);

}

protected String getExternalKGURI(String name) throws
JsonProcessingException, IOException, URISyntaxException {

    //Approximate solution: We get the entity with highest
lexical similarity
    //The use of context may be necessary in some cases
     Set<String> types = new HashSet<String>();


       Set<String> lang = new HashSet<String>();

    Set<KGEntity> entities = googleKGLookup.getEntities(name,
"10", types, lang, 0.0);

    double current_sim = -1.0;
    String current_uri="";
```

```
    for (KGEntity ent : entities) {
        double isub_score = isub.score(name, ent.getName());
        if (current_sim < isub_score) {
            current_uri = ent.getId();
            current_sim = isub_score;
        }
    }

    return current_uri;
}
```

```
protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {



          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          //We use the ascii restaurant to create the fresh URI
for a city in the dataset
          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
```

```
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        //Uri as already created
        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);
```

```
          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;


        //Uri as already created
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);
```

```
    }


}


public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/IN3067-
INM713_coursework_data_pizza_500.csv";

     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_RDF_3_RDF_Triples_using_GoogleKG solution = new
Task_RDF_3_RDF_Triples_using_GoogleKG(file, column_index);

          String task = "createRDF_GoogleKG";

          if (task.equals("createRDF_GoogleKG"))
```

```
                solution.performTaskRDFusingGoogleKG();  //Fresh
entity URIs


          //Graph with only data
          solution.saveGraph(solution.model, file.replace(".csv",
"-"+task)+".ttl");



     } catch (Exception e) {
          e.printStackTrace();
     }




}
}

package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
* https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Wikidatalookup.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/


import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
```

```
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_RDF_3_RDF_Triples_Using_WikiData {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     WikidataLookup wikidataLookup;

     List<String[]> csv_file;
```

```
     //Dictionary that keeps the URIs. Specially useful if
accessing a remote service to get a candidate URI  to avoid
repeated calls
    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();



public Task_RDF_3_RDF_Triples_Using_WikiData(String input_file,
Map<String, Integer> column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          //1. GRAPH INITIALIZATION

        //Empty graph
          model = ModelFactory.createDefaultModel();


          //Note that this is the same namespace used in the
ontology "pizza-restaurant-ontology.ttl"
        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


        //Prefixes for the serialization
          model.setNsPrefix("cw", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("wiki",
"http://wikidata.org/resource/");


        //Load data in matric to later use an iterator
          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
```

```
            reader.close();

            wikidataLookup = new WikidataLookup();
        }

public void performTaskRDFusingGoogleKG() throws
JsonProcessingException, IOException, URISyntaxException {
    CovertCSVToRDF(true);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


     //In a large ontology one would need to find a more
automatic way to use the ontology vocabulary.
    //e.g.,  via matching. In a similar way as we match entities
to a large KG like DBPedia or Wikidata
    //Since we are dealing with very manageable ontologies, we
can integrate their vocabulary
    //within the code. E.g.,: cw_ns_str + City



    //We modularize the transformation to RDF. The
transformation is tailored to the given table, but
    //he individual components/mappings are relatively generic
(especially type and literal triples).

    //Mappings may required one or more columns as input and
create 1 or more triples for an entity



     //We give subject column and target type
     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);
```

```
     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "state", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "menu_item", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "Itemvalue", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "categories", useExternalURI);




    //We give subject and object columns (they could be the
same), predicate and datatype

mappingToCreateLiteralTriple(column_index.get("item_description"
), column_index.get("item_description"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);




    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");
```

```
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "serves");

    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "containsAdress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "containsCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "containsState");

    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");
    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");




}

protected String processLexicalName(String name) {

     //Remove potential spaces and other characters not allowed
in URIs

    //This method may need to be extended
     //Other problematic characters:
    //{", "}", "|", "\", "^", "~", "[", "]", and "`"

     return name.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String name, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

    //We create fresh URI (default option)
    stringToURI.put(name, cw_ns_str + processLexicalName(name));
```

```
    if (useExternalURI) {//We connect to online KG
        String uri = getExternalKGURI(name);
        if (!uri.equals(""))
          stringToURI.put(name, uri);
    }

    return stringToURI.get(name);

}

protected String getExternalKGURI(String name) throws
JsonProcessingException, IOException, URISyntaxException {

    //Approximate solution: We get the entity with highest
lexical similarity
    //The use of context may be necessary in some cases

     final String lang="en";
    Set<KGEntity> entities =
wikidataLookup.getKGEntities("name");

    double current_sim = -1.0;
    String current_uri="";

    for (KGEntity ent : entities) {
        double isub_score = isub.score(name, ent.getName());
        if (current_sim < isub_score) {
            current_uri = ent.getId();
            current_sim = isub_score;
        }
    }

    return current_uri;
}




protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {
```

```
          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          //We use the ascii restaurant to create the fresh URI
for a city in the dataset
          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;
```

```
          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        //Uri as already created
        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];
```

```
          if (is_nan(object))
               continue;


        //Uri as already created
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/IN3067-
INM713_coursework_data_pizza_500.csv";

     //Format
```

```
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_RDF_3_RDF_Triples_Using_WikiData solution = new
Task_RDF_3_RDF_Triples_Using_WikiData(file, column_index);

          String task = "createRDF_Wikidata";

          if (task.equals("createRDF_Wikidata"))
               solution.performTaskRDFusingGoogleKG();  //Fresh
entity URIs


          //Graph with only data
          solution.saveGraph(solution.model, file.replace(".csv",
"-"+task)+".ttl");



     } catch (Exception e) {
          e.printStackTrace();
     }




}
}

package RDF;
```

```
/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;
```

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_RDF_4_Reasoning {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     DBpediaLookup dbpedia;

     List<String[]> csv_file;

     //Dictionary that keeps the URIs. Specially useful if
accessing a remote service to get a candidate URI  to avoid
repeated calls
    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_RDF_4_Reasoning(String input_file, Map<String,
Integer> column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          //1. GRAPH INITIALIZATION

        //Empty graph
          model = ModelFactory.createDefaultModel();


          //Note that this is the same namespace used in the
ontology "pizza-restaurant-ontology.ttl"
```

```
        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


        //Prefixes for the serialization
          model.setNsPrefix("restaurant", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


        //Load data in matric to later use an iterator
          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


     //In a large ontology one would need to find a more
automatic way to use the ontology vocabulary.
    //e.g.,  via matching. In a similar way as we match entities
to a large KG like DBPedia or Wikidata
    //Since we are dealing with very manageable ontologies, we
can integrate their vocabulary
    //within the code. E.g.,: cw_ns_str + City



    //We modularize the transformation to RDF. The
transformation is tailored to the given table, but
    //he individual components/mappings are relatively generic
(especially type and literal triples).
```

```
    //Mappings may required one or more columns as input and
create 1 or more triples for an entity



     //We give subject column and target type
     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "state", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "menu_item", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "Itemvalue", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "categories", useExternalURI);




    //We give subject and object columns (they could be the
same), predicate and datatype

mappingToCreateLiteralTriple(column_index.get("item_description"
), column_index.get("item_description"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);
```

```
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);




    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "serves");

    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "containsAdress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "containsCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "containsState");

    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");
    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");




}

protected String processLexicalName(String restaurant) {
```

```
     //Remove potential spaces and other characters not allowed
in URIs

    //This method may need to be extended
     //Other problematic characters:
    //{", "}", "|", "\", "^", "~", "[", "]", and "`"

     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

    //We create fresh URI (default option)
    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {



          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          //We use the ascii restaurant to create the fresh URI
for a city in the dataset
          if (stringToURI.containsKey(subject))
```

```
                subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        //Uri as already created
        String entity_uri =
stringToURI.get(subject.toLowerCase());
```

```
        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;


        //Uri as already created
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
```

```
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}



public void performReasoning(String ontology_file) {

    //We expand the graph with the inferred triples
    //Instead of RDFS Semantics, we use approximate OWL 2
reasoner close to OWL 2 RL (but not exactly)
    //More about OWL 2 RL Semantics in lecture/lab 7

    //Option 2
     //Uses a RDFS reasoner internally
     //InfModel inf_model = ModelFactory.createRDFSModel(model);

     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");

    //We should load the ontology first
    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);
```

```
    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}

public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/IN3067-
INM713_coursework_data_pizza_reduced.csv";

     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_RDF_4_Reasoning solution = new
Task_RDF_4_Reasoning(file, column_index);

          String task = "createRDF";

          if (task.equals("createRDF"))
               solution.performTaskRDF();  //Fresh entity URIs
```

```
          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");

     //solution.performReasoning("files_lab5/ontology_lab5.owl");

          //#Graph with ontology triples and entailed triples
          solution.saveGraph(solution.inf_model,
file.replace(".csv", "-"+task)+"-reasoning.ttl");



     } catch (Exception e) {
          e.printStackTrace();
     }




}
}

package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_RDF_Triples {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     List<String[]> csv_file;

    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;
```

```
    I_Sub isub = new I_Sub();


public Task_RDF_Triples(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;

          model = ModelFactory.createDefaultModel();

        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";

          model.setNsPrefix("restaurants", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {

     //Type Triples
     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);
```

```
     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);
     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);
    mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



    //Literal Triples
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);

    mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);
    mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);

    mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDdouble);



    //Object Triples
```

```
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
    mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");


    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");
    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");
    mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");

    mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");
    mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");
    mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");

    mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");
    mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");

    mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

    mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");
```

```
}

protected String processLexicalName(String restaurant) {

     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {


    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));

    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
```

```
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;

        String entity_uri =
stringToURI.get(subject.toLowerCase());

        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);
```

```
          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;

        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}
```

```
public void saveGraph(Model model, String file_output) throws
FileNotFoundException {


    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/IN3067-
INM713_coursework_data_pizza_500_new_entries.csv";

     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_RDF_Triples solution = new Task_RDF_Triples(file,
column_index);

          String task = "TASK_RDF_1_solution";

          if (task.equals("TASK_RDF_1_solution"))
               solution.performTaskRDF();


          //Graph with only data
```

```
          solution.saveGraph(solution.model,
file.replace("IN3067-
INM713_coursework_data_pizza_500_new_entries.csv",""+task)+".ttl
");



     } catch (Exception e) {
          e.printStackTrace();
     }







}
}

package RDF;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/WikidataEndpoint.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.util.Set;

import org.apache.jena.rdf.model.Statement;


public class WikidataEndpoint extends SPARQLEndpointService{


     //https://www.mediawiki.org/wiki/Wikidata_Query_Service/Use
r_Manual#SPARQL_endpoint
     //One coudl also use the toolkit to access items. See if
more efficient
```

```
     @Override
     public String getENDPOINT() {
          return "https://query.wikidata.org/sparql";
     }









     @Override
     protected String
createSPARQLQuery_AllTypesForSubject(String uri_resource) {
          // TODO Auto-generated method stub
          return null;
     }




     @Override
     protected String
createSPARQLQuery_AllSuperClassesForSubject(String uri_resource)
{
          // TODO Auto-generated method stub
          return null;
     }



     @Override
     protected String createSPARQLQueryForObject(String
uri_object) {
          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT ?s ?p \n"
                    + "WHERE { ?s ?p <" + uri_object + "> . "
                    + "}";
     }
```

```
     protected String createSPARQLQueryForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT ?p ?o \n"
                    + "WHERE { <" + uri_subject + "> ?p ?o . "
                    + "}";

     }


     /**
      * To extract class types of the subject
      * @param uri_subject
      * @return
      */
     protected String createSPARQLQuery_TypesForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
                    "SELECT DISTINCT ?uri \n"
                    + "WHERE { <" + uri_subject + ">
<http://www.wikidata.org/prop/direct/P31> ?uri . "
                    + "}";

     }

     @Override
     protected String
craeteSPARQLQuery_TypeObjectsForPredicate(String uri_predicate)
{
          return "SELECT DISTINCT ?uri \n"
                    + "WHERE { ?s <" + uri_predicate + "> ?o . "
                    + "?o
<http://www.wikidata.org/prop/direct/P31> ?uri ."
                    + "}";
     }

     /*protected String createSPARQLQuery_LabelForSubject(String
uri_subject){

          return //"PREFIX foaf: <http://xmlns.com/foaf/0.1/> \n
"+
```

```
                    "SELECT DISTINCT ?l \n"
                    + "WHERE { <" + uri_subject + ">
<http://www.wikidata.org/prop/Q722218> ?l . "
                    + "}";

     }*/
     //TODO: query for labels!




     public static void main(String[] args) {


          String subject =
"http://www.wikidata.org/entity/Q974";

          WikidataEndpoint wde =  new WikidataEndpoint();

          try {

     //System.out.println(wde.getValuesForQuery(wde.createSPARQL
Query_LabelForSubject(subject)));

     //System.out.println(wde.getTriplesForSubject(subject));

     System.out.println(wde.getLabelsForSubject(subject));

     //System.out.println(wde.getTypesForSubject(subject));

          } catch (Exception e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }


     }




}

package RDF;
```

```
/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/WikidataLookup.java
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import org.apache.http.client.utils.URIBuilder;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;



public class WikidataLookup extends LookupService{

     //Help:
//https://www.wikidata.org/w/api.php?action=help&modules=wbsearc
hentities
     //Example:
https://www.wikidata.org/w/api.php?action=wbsearchentities&searc
h=virgin%20media&language=en


     private final String REST_URL =
"https://www.wikidata.org/w/api.php?action=wbsearchentities&form
at=json&";

     private final String limit = "limit";
     private final String search = "search";
     private final String language = "language";
     //Not supported
     //private final String QueryClass = "QueryClass";
```

```
     private int hits=10;
     private String lang="en";

     protected URL buildRequestURL(String query, int max_hits,
String lang)
               throws URISyntaxException, MalformedURLException
{
          URIBuilder ub = new URIBuilder(getREST_URL());
          //ub.addParameter(QueryClass, cls_type);
          ub.addParameter(limit, String.valueOf(max_hits));
          ub.addParameter(language, lang);
          ub.addParameter(search, query);
          return ub.build().toURL();

     }

     @Override
     protected String getREST_URL() {
          return REST_URL;
     }


     public Set<KGEntity> getKGEntities(String query, String
cls_type, int max_hits, String language)
               throws JsonProcessingException, IOException,
URISyntaxException {
          return getKGEntities(query, max_hits, language);
     }



     public Set<KGEntity> getKGEntities(String query) throws
JsonProcessingException, IOException, URISyntaxException{
          return getKGEntities(query, hits, lang);
     }


     /**
      *
      * @param query
      * @param max_hits
      * @param language
      * @return
      * @throws JsonProcessingException
      * @throws IOException
      * @throws URISyntaxException
```

```
     */
    public Set<KGEntity> getKGEntities(String query, int
max_hits, String language)
               throws JsonProcessingException, IOException,
URISyntaxException {

          Set<KGEntity> entities = new HashSet<KGEntity>();


          URL urlToGet = buildRequestURL(query, max_hits,
language);

          //System.out.println(urlToGet);
          //System.out.println(getRequest(urlToGet));


          for (JsonNode result :
jsonToNode(getRequest(urlToGet)).get("search")){

               //System.out.println(result.toString());

               KGEntity ent = new KGEntity();
               ent.setId(result.get("concepturi").asText());
               ent.setName(result.get("label").asText());

    ent.setDescription(result.get("description").asText());
               entities.add(ent);


    //entities.add(result.get("concepturi").asText());
          }

          return entities;
    }




    public static void main(String[] args){
          WikidataLookup lookup = new WikidataLookup();

          String keywords;
          keywords="Chicago Bulls";
          keywords="Congo";
```

```
          try {

               //Look up for entities matching the string
"keywords"
               Set<KGEntity> entities =
lookup.getKGEntities(keywords);
               System.out.println("NUmber of candidates found: "
+ entities.size());

               for (KGEntity ent : entities){
                    System.out.println(ent);
               }
          } catch (JsonProcessingException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          } catch (IOException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          } catch (URISyntaxException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }

     }



}

package SPARQL;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
```

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_2_3_SPARQL_1 {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;


     List<String[]> csv_file;
```

```
    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_2_3_SPARQL_1(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          model = ModelFactory.createDefaultModel();


        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


          model.setNsPrefix("cw", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}
```

```
protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {

          //Type Triples

     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);
          mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);
          mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);
          mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);
          mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);
         mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);

mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



          //Literal Triples

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("restaurant"),
```

```
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDdouble);



        //Object Triples

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");



mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");

mappingToCreateObjectTriple(column_index.get("menu_item"),
```

```
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");


mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");


mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");

mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");

        mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");


mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");


}

protected String processLexicalName(String restaurant) {


     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {
```

```
    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;


          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}
```

```
private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}
```

```
protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;


        //Uri as already created
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void performSPARQLQuery(Model model, String
file_query_out) {

     WriteFile writer = new WriteFile(file_query_out);
```

```
   String queryStr =
             "PREFIX cw:
<http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>\n" +
             "SELECT ?restaurant ?state ?postcode WHERE {\n" +
             "?restaurant rdf:type cw:Restaurant .\n" +
             "?restaurant cw:locatedInState ?state .\n" +
             "?restaurant cw:postcode ?postcode .\n" +
             "FILTER (?postcode = '97701')\n" +"}\nORDER BY
(?restaurant)";

    Query q = QueryFactory.create(queryStr);

     QueryExecution qe =
               QueryExecutionFactory.create(q, model);
     try {
          ResultSet res = qe.execSelect();

          int solutions = 0;

          while( res.hasNext()) {
               solutions++;
               QuerySolution soln = res.next();
               RDFNode restaurant = soln.get("?restaurant");
               RDFNode state = soln.get("?state");
               RDFNode postcode = soln.get("?postcode");



     writer.writeLine(restaurant.toString()+","+state.toString()
+","+postcode.toString());

          }
          System.out.println(solutions + " results satisfying
the query.");

     } finally {
          qe.close();
     }

     writer.closeBuffer();
```

```
}


public void performReasoning(String ontology_file) {


     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");

    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);


    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}



public void saveGraph(Model model, String file_output) throws
FileNotFoundException {


    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/50_unique_data.csv";

     //Format
```

```
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_2_3_SPARQL_1 solution = new
Task_2_3_SPARQL_1(file, column_index);

          String task = "Sparl_Query_1_Solution";

          if (task.equals("Sparl_Query_1_Solution"))
               solution.performTaskRDF();


          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");


          solution.saveGraph(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

          solution.performSPARQLQuery(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+".csv");


     } catch (Exception e) {
          e.printStackTrace();
     }
```

```
}
}

package SPARQL;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
```

```
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_2_3_SPARQL_2 {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;


     List<String[]> csv_file;

    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_2_3_SPARQL_2(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          model = ModelFactory.createDefaultModel();


        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


          model.setNsPrefix("cw", cw_ns_str);
```

```
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {

               //Type Triples

     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
```

```
mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);

mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);

mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



              //Literal Triples

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDinteger);

mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDinteger);



              //Object Triples
```

```
mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");



mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");


mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");
```

```
mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");

mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");


mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");


mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");



}

protected String processLexicalName(String restaurant) {



     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

    //We create fresh URI (default option)
    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {
```

```
     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
           subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;
```

```
          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          //Ignore rows with less elements than expected
          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;
```

```
        //Uri as already created
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void performSPARQLQuery(Model model, String
file_query_out) {

     WriteFile writer = new WriteFile(file_query_out);


   String queryStr =
            "PREFIX cw:
<http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>\n" +
                    "PREFIX xsd:
<http://www.w3.org/2001/XMLSchema#>\n" +
                    "SELECT DISTINCT ?restaurant ?menu_item
?state ?postcode ?item_value WHERE {\n" +
                    "    ?restaurant rdf:type cw:Restaurant
.\n" +
```

```
                    "    ?restaurant cw:locatedInState ?state
.\n" +
                    "    ?restaurant cw:postcode ?postcode .\n"
+
                    "    ?menu_item cw:amount ?item_value .\n"
+
                    "    FILTER (?postcode = '63105' &&
xsd:double(?item_value) = 15)\n" +
                    "}\nORDER BY (?restaurant)";

    Query q = QueryFactory.create(queryStr);

     QueryExecution qe =
               QueryExecutionFactory.create(q, model);
     try {
          ResultSet res = qe.execSelect();

          int solutions = 0;

          while( res.hasNext()) {
               solutions++;
               QuerySolution soln = res.next();
               RDFNode restaurant = soln.get("?restaurant");
               RDFNode state = soln.get("?state");
               RDFNode menu_item = soln.get("?menu_item");
               RDFNode item_value = soln.get("?item_value");
               RDFNode postcode = soln.get("?postcode");




     writer.writeLine(restaurant.toString()+","+state.toString()
+","+menu_item.toString()+","+item_value.toString()+","+postcode
.toString());

     //writer.writeLine(restaurant.toString()+","+item_value.toS
tring());

          }
          System.out.println(solutions + " results satisfying
the query.");

     } finally {
          qe.close();
     }
```

```
     writer.closeBuffer();


}


public void performReasoning(String ontology_file) {


     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");

    //We should load the ontology first
    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);


    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}



public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/50_unique_data.csv";
```

```
     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_2_3_SPARQL_2 solution = new
Task_2_3_SPARQL_2(file, column_index);

          String task = "Sparl_Query_2_Solution";

          if (task.equals("Sparl_Query_2_Solution"))
               solution.performTaskRDF();  //Fresh entity URIs


          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");

          solution.saveGraph(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

          solution.performSPARQLQuery(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+".csv");


     } catch (Exception e) {
          e.printStackTrace();
     }
```

```
}
}

package SPARQL;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
```

```
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_2_3_SPARQL_3 {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     List<String[]> csv_file;


    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_2_3_SPARQL_3(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;

          model = ModelFactory.createDefaultModel();


        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


          model.setNsPrefix("cw", cw_ns_str);
```

```
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


     //Type Triples

     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);
```

```
mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);

mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);

mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



              //Literal Triples

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDdouble);



              //Object Triples
```

```
mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");



mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");


mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");
```

```
mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");

mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");


mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");


mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");




}

protected String processLexicalName(String restaurant) {


     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {


    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {
```

```
     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;
```

```
          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;

        String entity_uri =
stringToURI.get(subject.toLowerCase());


        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {

          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;
```

```
        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void performSPARQLQuery(Model model, String
file_query_out) {

     WriteFile writer = new WriteFile(file_query_out);



   String queryStr =
             "PREFIX cw:
<http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>\n" +
             "SELECT ?restaurant ?state WHERE {\n" +
              "{\n" +
              "?restaurant cw:locatedInState ?state .\n" +
              "}\n" +
              "UNION\n" +
              "{\n" +
              "?restaurant cw:locatedInState ?state .\n" +
              "}\n" +
              "MINUS\n" +
              "{\n" +
```

```
            "?restaurant cw:Blockbuster ?state .\n" +
            "}\n" +
            "}";

    Query q = QueryFactory.create(queryStr);

     QueryExecution qe =
               QueryExecutionFactory.create(q, model);
     try {
          ResultSet res = qe.execSelect();

          int solutions = 0;

          while( res.hasNext()) {
               solutions++;
               QuerySolution soln = res.next();
               RDFNode restaurant = soln.get("?restaurant");
               RDFNode state = soln.get("?state");



     writer.writeLine(restaurant.toString()+","+state.toString()
);


          }
          System.out.println(solutions + " results satisfying
the query.");

     } finally {
          qe.close();
     }

     writer.closeBuffer();


}


public void performReasoning(String ontology_file) {


     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");
```

```
    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);


    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}



public void saveGraph(Model model, String file_output) throws
FileNotFoundException {


    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/50_unique_data.csv";

     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
```

```
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_2_3_SPARQL_3 solution = new
Task_2_3_SPARQL_3(file, column_index);

          String task = "Sparl_Query_3_Solution";

          if (task.equals("Sparl_Query_3_Solution"))
               solution.performTaskRDF();


          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");

          solution.saveGraph(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

          solution.performSPARQLQuery(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+".csv");


     } catch (Exception e) {
          e.printStackTrace();
     }




}
}

package SPARQL;

/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
```

```
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_2_3_SPARQL_4 {
```

```
     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     List<String[]> csv_file;


    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;

    I_Sub isub = new I_Sub();


public Task_2_3_SPARQL_4(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          model = ModelFactory.createDefaultModel();


        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


          model.setNsPrefix("cw", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();
```

```
     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {

               //Type Triples

     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);

mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);
```

```
mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



              //Literal Triples

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDdouble);



              //Object Triples

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");
```

```
mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");



mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");


mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");


mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");

mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");


mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");


mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");
```

```
}

protected String processLexicalName(String restaurant) {



     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}


protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {


    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;

          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
```

```
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }

}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;



          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        //Uri as already created
        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
```

```
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}


protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;


        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
```

```
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void performSPARQLQuery(Model model, String
file_query_out) {

     WriteFile writer = new WriteFile(file_query_out);



     String queryStr =
                  "PREFIX cw:
<http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
                  "PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>\n" +
                  "SELECT ?city (COUNT(?restaurant) AS
?restaurantCount) WHERE {\n" +
                  "?restaurant rdf:type cw:Restaurant .\n"+
                  "?restaurant cw:locatedInCity ?city .\n"+"}
GROUP BY ?city HAVING(COUNT(?restaurant)>0)";



    Query q = QueryFactory.create(queryStr);

     QueryExecution qe =
               QueryExecutionFactory.create(q, model);
     try {
          ResultSet res = qe.execSelect();

          int solutions = 0;

          while( res.hasNext()) {
               solutions++;
               QuerySolution soln = res.next();
```

```
                RDFNode restaurantCount =
soln.get("?restaurantCount");
                RDFNode city = soln.get("?city");

                //writer.writeLine(city.toString());

     writer.writeLine(city.toString()+","+restaurantCount.toStri
ng());

     //writer.writeLine(restaurant.toString()+","+item_value.toS
tring());

          }
          System.out.println(solutions + " results satisfying
the query.");

     } finally {
          qe.close();
     }

     writer.closeBuffer();


}


public void performReasoning(String ontology_file) {


     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");

    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);
```

```
    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}



public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/50_unique_data.csv";

     //Format
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_2_3_SPARQL_4 solution = new
Task_2_3_SPARQL_4(file, column_index);

          String task = "Sparl_Query_4_Solution";

          if (task.equals("Sparl_Query_4_Solution"))
               solution.performTaskRDF();
```

```
          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");

          solution.saveGraph(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

          solution.performSPARQLQuery(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+".csv");


     } catch (Exception e) {
          e.printStackTrace();
     }




}
}

package SPARQL;


/* Created by @author ernesto
* Available at: https://github.com/city-knowledge-graphs/java-
2024/blob/main/src/main/java/lab5/Lab5_Solution.java
*
* Type of Code: Java code
*
* Updated by @author Shreyas Jadhav, Apoorva Ramaiah
* Coursework: Semantic Web and Technologies Part 2
* Coursework Group Name: Nerds
* Updated on 11 May 2024
*/

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;
import java.util.Set;

import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.apache.jena.datatypes.xsd.XSDDatatype;
import org.apache.jena.query.Dataset;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
import org.apache.jena.vocabulary.OWL;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.opencsv.CSVReader;

import util.WriteFile;

public class Task_2_3_SPARQL_5 {

     String input_file;
     Model model;
     InfModel inf_model;

     String cw_ns_str;

     List<String[]> csv_file;

    Map<String, String> stringToURI = new HashMap<String,
String>();

    Map<String, Integer> column_index;
```

```
    I_Sub isub = new I_Sub();


public Task_2_3_SPARQL_5(String input_file, Map<String, Integer>
column_index) throws IOException {


          this.input_file = input_file;

          this.column_index = column_index;


          model = ModelFactory.createDefaultModel();


        cw_ns_str= "http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#";


          model.setNsPrefix("cw", cw_ns_str);
          model.setNsPrefix("xsd",
"http://www.w3.org/2001/XMLSchema#");
          model.setNsPrefix("dbr",
"http://dbpedia.org/resource/");


          CSVReader reader = new CSVReader(new
FileReader(input_file));
         csv_file = reader.readAll();
         reader.close();


     }

public void performTaskRDF() throws JsonProcessingException,
IOException, URISyntaxException {
    CovertCSVToRDF(false);
}

protected void CovertCSVToRDF(boolean useExternalURI) throws
JsonProcessingException, IOException, URISyntaxException {


               //Type Triples
```

```
     mappingToCreateTypeTriple(column_index.get("restaurant"),
cw_ns_str + "Restaurant", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("address"),
cw_ns_str + "Address", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("city"),
cw_ns_str + "City", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("country"),
cw_ns_str + "Country", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("postcode"),
cw_ns_str + "Postcode", useExternalURI);

     mappingToCreateTypeTriple(column_index.get("state"),
cw_ns_str + "State", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "Food", useExternalURI);

mappingToCreateTypeTriple(column_index.get("menu_item"),
cw_ns_str + "MenuItem", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_value"),
cw_ns_str + "ItemValue", useExternalURI);

mappingToCreateTypeTriple(column_index.get("currency"),
cw_ns_str + "Currency", useExternalURI);

mappingToCreateTypeTriple(column_index.get("item_description"),
cw_ns_str + "Ingredient", useExternalURI);

mappingToCreateTypeTriple(column_index.get("categories"),
cw_ns_str + "Categories", useExternalURI);



              //Literal Triples

mappingToCreateLiteralTriple(column_index.get("restaurant"),
column_index.get("restaurant"), cw_ns_str + "restaurantName",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("restaurant"),
```

```
column_index.get("postcode"), cw_ns_str + "postcode",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDstring);

mappingToCreateLiteralTriple(column_index.get("menu_item"),
column_index.get("menu_item"), cw_ns_str + "itemName",
XSDDatatype.XSDstring);


mappingToCreateLiteralTriple(column_index.get("item_value"),
column_index.get("item_value"), cw_ns_str + "amount",
XSDDatatype.XSDdouble);



              //Object Triples

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("country"), cw_ns_str + "locatedInCountry");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("city"), cw_ns_str + "locatedInCity");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("address"), cw_ns_str + "locatedInAddress");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("state"), cw_ns_str + "locatedInState");

mappingToCreateObjectTriple(column_index.get("restaurant"),
column_index.get("menu_item"), cw_ns_str + "servesMenuItem");



mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("restaurant"), cw_ns_str +
"servedInRestaurant");

mappingToCreateObjectTriple(column_index.get("menu_item"),
column_index.get("item_value"), cw_ns_str + "hasValue");

mappingToCreateObjectTriple(column_index.get("menu_item"),
```

```
column_index.get("item_description"), cw_ns_str +
"hasIngredient");



mappingToCreateObjectTriple(column_index.get("item_description")
, column_index.get("menu_item"), cw_ns_str + "isIngredientOf");


mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("address"), cw_ns_str + "containsAdress");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("city"), cw_ns_str + "containsCity");

mappingToCreateObjectTriple(column_index.get("country"),
column_index.get("currency"), cw_ns_str + "amountCurrency");


mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("state"), cw_ns_str + "containsState");

mappingToCreateObjectTriple(column_index.get("address"),
column_index.get("city"), cw_ns_str + "containsCity");


mappingToCreateObjectTriple(column_index.get("city"),
column_index.get("country"), cw_ns_str + "locatedInCountry");


mappingToCreateObjectTriple(column_index.get("item_value"),
column_index.get("currency"), cw_ns_str + "amountCurrency");




}

protected String processLexicalName(String restaurant) {


     return restaurant.replaceAll(" ", "_").replaceAll(",", "");
}
```

```
protected String createURIForEntity(String restaurant, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

    stringToURI.put(restaurant, cw_ns_str +
processLexicalName(restaurant));


    return stringToURI.get(restaurant);

}


protected void mappingToCreateTypeTriple(int
subject_column_index, String class_type_uri, boolean
useExternalURI) throws JsonProcessingException, IOException,
URISyntaxException {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;


          String subject =
row[subject_column_index].toLowerCase();
          String subject_uri;


          if (stringToURI.containsKey(subject))
               subject_uri=stringToURI.get(subject);
        else
            subject_uri=createURIForEntity(subject,
useExternalURI);


          //TYPE TRIPLE
          Resource subject_resource =
model.createResource(subject_uri);
          Resource type_resource =
model.createResource(class_type_uri);

          model.add(subject_resource, RDF.type, type_resource);

     }
```

```
}

private boolean is_nan(String value) {
    return (!value.equals(value));
}


protected void mappingToCreateLiteralTriple(int subject_column,
int object_column, String predicate, XSDDatatype datatype) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;

          String subject = row[subject_column];
          String lit_value = row[object_column];

          if (is_nan(lit_value))
               continue;


        String entity_uri =
stringToURI.get(subject.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(entity_uri);
        Property predicate_resource =
model.createProperty(predicate);

          //Literal
        Literal lit = model.createTypedLiteral(lit_value,
datatype);


          model.add(subject_resource, predicate_resource, lit);

     }


}
```

```
protected void mappingToCreateObjectTriple(int subject_column,
int object_column, String predicate) {

     for (String[] row : csv_file) {


          if (row.length<column_index.size())
               continue;

          String subject = row[subject_column];
          String object = row[object_column];

          if (is_nan(object))
               continue;


        String subject_uri =
stringToURI.get(subject.toLowerCase());
        String object_uri =
stringToURI.get(object.toLowerCase());


        //New triple
        Resource subject_resource =
model.createResource(subject_uri);
        Property predicate_resource =
model.createProperty(predicate);
        Resource object_resource =
model.createResource(object_uri);


          model.add(subject_resource, predicate_resource,
object_resource);

     }


}


public void performSPARQLQuery(Model model, String
file_query_out) {

     WriteFile writer = new WriteFile(file_query_out);
```

```
     String queryStr =
                    "PREFIX cw:
<http://www.semanticweb.org/city/in3067-
inm713/2024/restaurants#>\n" +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>\n" +
                    "SELECT ?menu_item (COUNT(?restaurant) AS
?restaurantCount) WHERE {\n" +
                    "?menu_item rdf:type cw:Menu_item .\n"+
                    "?menu_item cw:servedInRestaurant ?restaurant
.\n"+"} GROUP BY ?menu_item HAVING(COUNT(?restaurant)>0) ORDER
BY COUNT(?restaurant) ?menu_item";


    Query q = QueryFactory.create(queryStr);

     QueryExecution qe =
               QueryExecutionFactory.create(q, model);
     try {
          ResultSet res = qe.execSelect();

          int solutions = 0;

          while( res.hasNext()) {
               solutions++;
               QuerySolution soln = res.next();
               RDFNode menu_item = soln.get("?menu_item");
               RDFNode restaurantCount =
soln.get("?restaurantCount");



     writer.writeLine(menu_item.toString()+","+restaurantCount.t
oString());


          }
          System.out.println(solutions + " results satisfying
the query.");

     } finally {
          qe.close();
     }

     writer.closeBuffer();
```

```
}


public void performReasoning(String ontology_file) {

     System.out.println("Data triples from CSV: '" +
model.listStatements().toSet().size() + "'.");

    Dataset dataset = RDFDataMgr.loadDataset(ontology_file);

model.add(dataset.getDefaultModel().listStatements().toList());


    System.out.println("Triples including ontology: '" +
model.listStatements().toSet().size() + "'.");

    Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();

     inf_model = ModelFactory.createInfModel(reasoner, model);


    System.out.println("Triples after reasoning: '" +
inf_model.listStatements().toSet().size() + "'.");


}



public void saveGraph(Model model, String file_output) throws
FileNotFoundException {

    //SAVE/SERIALIZE GRAPH
    OutputStream out = new FileOutputStream(file_output);
    RDFDataMgr.write(out, model, RDFFormat.TURTLE);

  }

public static void main(String[] args) {

     String file = "files/50_unique_data.csv";

     //Format
```

```
     //restaurant    address    city    country    postcode
state    categories    menu_item    item_value    currency
     item_description
     Map<String, Integer> column_index = new HashMap<String,
Integer>();
     column_index.put("restaurant", 0);
     column_index.put("address", 1);
     column_index.put("city", 2);
     column_index.put("country", 3);
     column_index.put("postcode", 4);
     column_index.put("state", 5);
     column_index.put("categories", 6);
     column_index.put("menu_item", 7);
     column_index.put("item_value", 8);
     column_index.put("currency", 9);
     column_index.put("item_description", 10);

     try {
          Task_2_3_SPARQL_5 solution = new
Task_2_3_SPARQL_5(file, column_index);

          String task = "Sparl_Query_5_Solution";

          if (task.equals("Sparl_Query_5_Solution"))
               solution.performTaskRDF();  //Fresh entity URIs


          solution.performReasoning("files/pizza-restaurants-
ontology.ttl");

          solution.saveGraph(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+"-reasoning.ttl");

          solution.performSPARQLQuery(solution.inf_model,
file.replace("50_unique_data.csv", ""+task)+".csv");


     } catch (Exception e) {
          e.printStackTrace();
     }



}
}
```