Refactor your n3 from assignment 3 into 2 files.

- a4name.rdfs -- contains ontology
- a4name.n3 -- contains facts
- make certain you have the foaf namespace correct!

Select two of the files in the onto/a3 directory and integrate them into the two files above. You should now have more ontology and more facts.

Add some additional ontology information and facts. Make sure you have examples of:

- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:domain and rdfs:range
- rdfs:label (see text p 110)
- rdfs:seeAlso
- rdfs:isDefinedBy
- rdfs:comment
- xsd:Date
- xsd:Integer

Write at least 3 SPARQL queries that demonstrate the following:

- use of UNION
- use of DISTINCT
- use of ORDER BY
- use of FILTER based on xsd:Date and some other xsd value
- use of CONSTRUCT to add new triples to your model and a query that uses the results of those new triples

Submit to BB as separate files:

- a4<foaf:lastName>.rdfs
- a4<foaf:lastName>.n3
- a4<foaf:lastName>.txt (contains your FULL (with namespaces) SPARQL Queries and the results.)
- the java program or programs you used to generate the output seen in the .txt files.

Please do NOT send a ZIP file to Blackboard. Upload individual files. Life is then much easier on the reading end.

Extra Credit: (10 pts)

Write a Java function that can be used by Jena to report problems in the use of any of the standard namespaces (e.g. rdf, rdfs, owl, foaf, dc, rss, and any others you deem appropriate). The namespaces should
not be hard coded, but available in some form on the web so the namespaces can be changed without recompiling your program.

Extra Credit (20 points)
Write a server utility function that does the same thing and that can be called from jena.

Example jena Program using onto + data

Data: onto1.rdfs

@prefix : <http://www.codesupreme.com/onto/cse7392/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

:Person a rdf:Class.
:Musician rdfs:subClassOf :Person.

Data: onto1.n3

@prefix : <http://www.codesupreme.com/onto/cse7392/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

:rollo a :Musician.
package jenasemweb;

import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.rdf.model.InfModel;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.reasoner.Reasoner;
import com.hp.hpl.jena.reasoner.ReasonerRegistry;
import com.hp.hpl.jena.util.FileManager;

/**
 * @author Frank Coyle
 * This program illustrates how one can use the SPARQL CONSTRUCT feature
 * as a rule system to create new triples. The new triples become a new model
 * that we add to our existing model which lets us execute queries that we
 * would not be able to execute without the new triple.
 */
public class OntoDataSeparate {
    public static void main(String[] args) {
        // LOAD Ontology
        Model schema = FileManager.get().loadModel("file:data/onto1.rdfs", "N3");
        // Load Data
        Model data = FileManager.get().loadModel("file:data/onto1.n3", "N3");

        // when you do OWL, use this other line
        // Reasoner reasoner = ReasonerRegistry.getOWLReasoner();

        // get a REASONER
        Reasoner reasoner = ReasonerRegistry.getRDFSReasoner();
        reasoner = reasoner.bindSchema(schema);

        // Build the inference model == adds new triples
        InfModel infmodel = ModelFactory.createInfModel(reasoner, data);

        // Use SPARQL
        String queryString =
            "PREFIX drc: <http://www.codesupreme.com/onto/cse7392/> " +
            "PREFIX eg: <urn:x-hp:eg/> " +
            "PREFIX : <urn:x-hp:eg/> " +
            "PREFIX rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
            "PREFIX rdf: <http://www.w3.org/2001/01/rdf-schema#> " +
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
            "PREFIX owl: <http://www.w3.org/2002/07/owl#> " +
            "DESCRIBE ?x " + // gotcha: MUST be a space after last ?var
            "WHERE {" +
            " ?x a drc:Musician ." +
            " })";

        // create a Jena CONSTRUCT query from the queryString
        com.hp.hpl.jena.query.Query query = QueryFactory.create(queryString);
// create a Jena QueryExecution object that knows the query
// and the N3 over which the query will be run
QueryExecution qe = QueryExecutionFactory.create(query, infmodel);

// executing the CONSTRUCT gives us a new model with triples
// that satisfy the WHERE clause
Model newRawModel = qe.execDescribe();

// take a look at the new model
newRawModel.write(System.out, "N3");

Output: -------------------------------------------------------------
@prefix :        <urn:x-hp:eg/> .
@prefix dc:      <http://purl.org/dc/elements/1.1/> .
@prefix rdfs:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix eg:      <urn:x-hp:eg/> .
@prefix geo:     <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/2001/01/rdf-schema#> .
@prefix drc:     <http://www.codesupreme.com/onto/cse7392/> .

drc:rollo
    a drc:Musician , drc:Person .