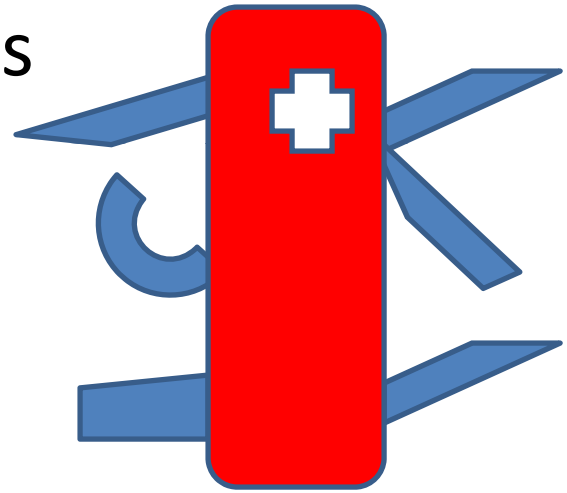
Building Semantic Web Applications with Java Jena

Frank Coyle
Lyle School of Engineering
Southern Methodist University
Dallas Texas
coyle@smu.edu

Jena Overview

- A Java API for processing RDF
- Complex with many capabilities
  - process RDF and N3
  - RDFS/ OWL inferencing
  - Rule processing
  - SPARQL queries



You will need:

- Java
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Jena Jar files
  - http://jena.sourceforge.net/downloads.html

- Useful: NetBeans to run Java Jena programs
  - http://netbeans.org/

Useful Jena Constructs

Statement

Resource

Property

Model

InfModel

Reasoner

Query

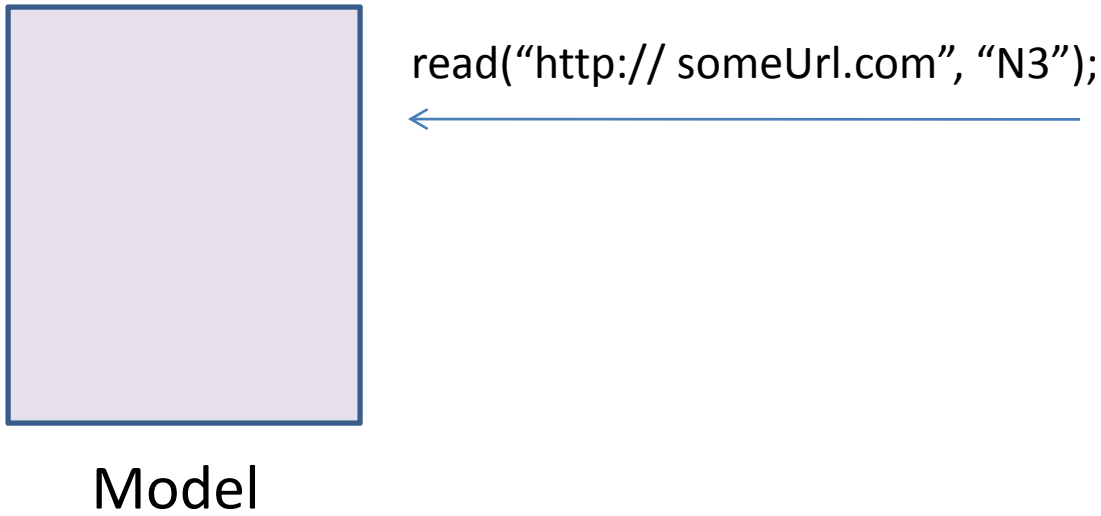
QueryExecution

ResultSet

Iterator

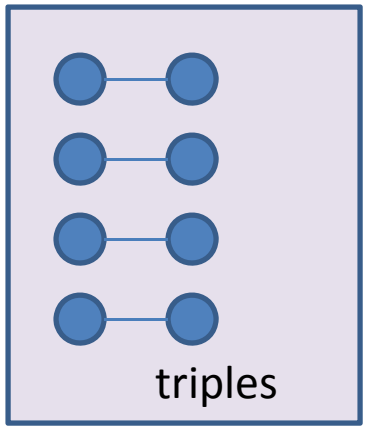
Jena: Model

- A Java Interface
- A Model allows us to perform a variety of operations on a set of statements (triples)



Jena: Model

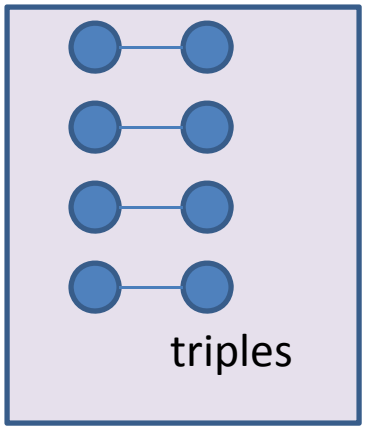
- A Java Interface
- A Model allows us to perform a variety of operations on a set of statements (triples)



Model

read("http:// someUrl.com", "N3");

We can read from:
- URLs
- files
- other Models



Model

Operations on Models

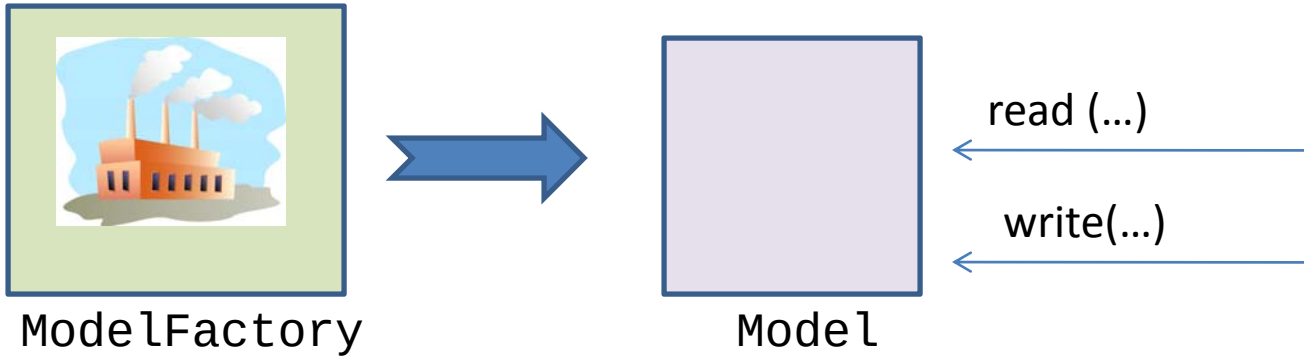
- write (…) // outputstream
- size() // number of triples in our Model
- remove (Statement s)
- contains (Statement s)
- contains (Resource s, Property p, RDFNode o) // pattern with wild cards
- add (Model otherModel)
- createResource(String uri)

http://lyle.smu.edu/~coyle/semweb/people.n3

```
@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 people: <http://lyle.smu.edu/~coyle/people#> .


# Frank
people:frank rdf:type foaf:Person .
people:frank foaf:name "Frank Coyle" .
people:frank foaf:title "Dr" .
people:frank foaf:firstName "Frank" .
people:frank foaf:surname "Coyle" .
people:frank foaf:mbox "coyle@smu.edu" .
people:frank foaf:based_near <http://dbpedia.org/resource/Dallas> .
people:frank foaf:knows _:john .
people:frank rdfs:seeAlso <http://lyle.smu.edu/~coyle/more.n3> .

# John Doe (a blank node)
_:john a foaf:Person .
_:john  foaf:name "John Doe".
```




```
// location of N3 data
String myUrl1  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/people.n3";

// create a basic no-inferencing model
Model basicModel  = ModelFactory.createDefaultModel();

// load N3 data into the model from a URL
basicModel.read(myUrl1, "N3");

// write  model data
basicModel.write(System.out, "N3");
```

basicModel.write(System.out, "N3");

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix people:  <http://lyle.smu.edu/~coyle/people#> .

people:frank
      a       foaf:Person ;
      rdfs:seeAlso <http://lyle.smu.edu/~coyle/semweb/semtech11/more.n3> ;
      foaf:based_near <http://dbpedia.org/resource/Dallas> ;
      foaf:firstName "Frank" ;
      foaf:knows
              [ a       foaf:Person ;
                foaf:name "John Doe"
              ] ;
      foaf:mbox "coyle@smu.edu" ;
      foaf:name "Frank Coyle" ;
      foaf:surname "Coyle" ;
      foaf:title "Dr" .
```

```java
// load N3 data into the model from a URL
basicModel.read(myUrl1, "N3");

// write  model data
basicModel.write(System.out); // defaults to RDF
```

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:people="http://lyle.smu.edu/~coyle/people#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >

  <rdf:Description rdf:about="http://lyle.smu.edu/~coyle/people#frank">
    <rdfs:seeAlso rdf:resource="http://lyle.smu.edu/~coyle/semweb/semtech11/more.n3"/>
    <foaf:knows rdf:nodeID="A0"/>
    <foaf:based_near rdf:resource="http://dbpedia.org/resource/Dallas"/>
    <foaf:mbox>coyle@smu.edu</foaf:mbox>
    <foaf:surname>Coyle</foaf:surname>
    <foaf:firstName>Frank</foaf:firstName>
    <foaf:title>Dr</foaf:title>
    <foaf:name>Frank Coyle</foaf:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdf:Description>

  <rdf:Description rdf:nodeID="A0">
    <foaf:name>John Doe</foaf:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdf:Description>

</rdf:RDF>
```

more.n3

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix people: <http://lyle.smu.edu/~coyle/people#> .

#  we use a blank node to not pollute the global name space
_:school rdf:type foaf:Organization .
_:school foaf:name "SMU Lyle School of Engineering" .

people:frank foaf:member _:school .
```

Multiple Inputs to a Model

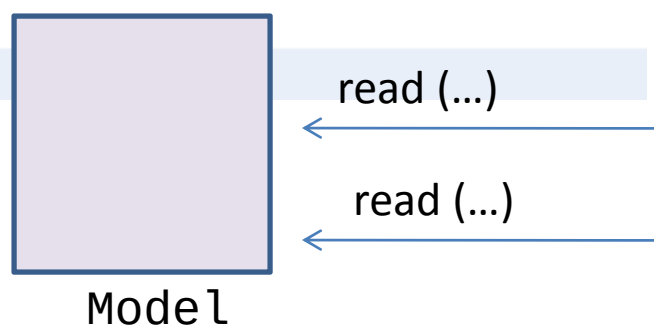
```
// location of N3 data
String myUrl1  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/people.n3";
String myUrl2  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/more.n3";

// create a basic no-inferencing model
Model basicModel  = ModelFactory.createDefaultModel();

// load N3 data into the model from a URL
basicModel.read(myUrl1, "N3");

// load N3 from another URL
basicModel.read(myUrl2, "N3");

// write  model data
basicModel.write(System.out, "N3");
```



Model augmented with more.n3

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix people:  <http://lyle.smu.edu/~coyle/people#> .

people:frank
      a        foaf:Person ;
      rdfs:seeAlso <http://lyle.smu.edu/~coyle/more.n3> ;
      foaf:based_near <http://dbpedia.org/resource/Dallas> ;
      foaf:firstName "Frank" ;
      foaf:knows
              [ a        foaf:Person ;
                foaf:name "John Doe"
              ] ;
      foaf:mbox "coyle@smu.edu" ;
      foaf:member
              [ a        foaf:Organization ;
                foaf:name "SMU Lyle School of Engineering"
              ] ;
      foaf:name "Frank Coyle" ;
      foaf:surname "Coyle" ;
      foaf:title "Dr" .
```

Jena and RDFS (RDF Schema)

- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:range
- rdfs:domain

RDFS Inferencing with Jena

```
@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     :   <http://lyle.smu.edu/~coyle/people#> .


# Willie Nelson
:willieN  rdf:type       :Musician.
:willieN  foaf:name      "Willie Nelson" .
:willieN  foaf:firstName "Willie" .
:willieN  foaf:surname   "Nelson" .
:willieN  foaf:based_near <http://dbpedia.org/resource/Austin> .


# Subclass relationship
:Musician  rdfs:subClassOf foaf:Person .
```

```
String myUrl3  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/people2.n3";

// create a basic no-inferencing model
Model basicModel  = ModelFactory.createDefaultModel();

basicModel.read(myUrl3, "N3");       // load data

basicModel.write(System.out, "N3"); // write triples
```

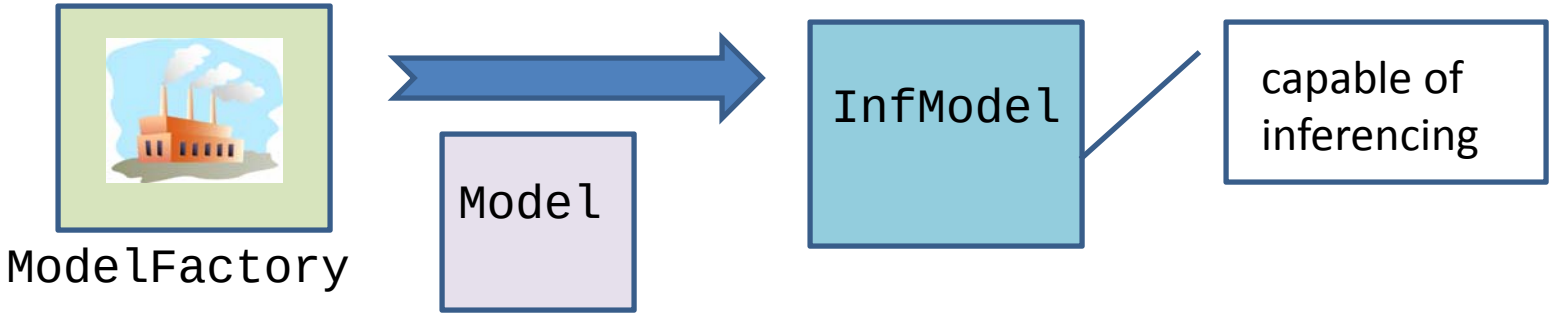
Output from Basic Model

```
@prefix :        <http://lyle.smu.edu/~coyle/people#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:Musician
      rdfs:subClassOf foaf:Person .

:willieN
      a        :Musician ;
      foaf:based_near <http://dbpedia.org/resource/Austin> ;
      foaf:firstName "Willie" ;
      foaf:name "Willie Nelson" ;
      foaf:surname "Nelson" .
```

Create InfModel using basic Model




```
String myUrl3  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/people2.n3";

Model basicModel = ModelFactory.createDefaultModel();

basicModel.read(myUrl3, "N3");       // load data

// Create RDFS InfModel using basicModel
InfModel infModel =
  ModelFactory.createRDFSModel(basicModel);

infModel.write(System.out, "N3"); // write triples
```

inferencing result

```
...

rdfs:Literal
      a        rdfs:Resource , rdfs:Class ;
      rdfs:subClassOf rdfs:Resource .

rdfs:comment
      a        rdf:Property , rdfs:Resource ;
      rdfs:range rdfs:Literal .

rdf:type
      a        rdf:Property , rdfs:Resource ;
      rdfs:range rdfs:Class .

rdfs:Datatype
      a        rdfs:Resource , rdfs:Class ;
      rdfs:subClassOf rdfs:Class , rdfs:Datatype .

:willieN
      a        foaf:Person , :Musician ;
      foaf:based_near <http://dbpedia.org/resource/Austin> ;
      foaf:firstName "Willie" ;
      foaf:name "Willie Nelson" ;
      foaf:surname "Nelson" .

...
```

Using `listStatements` to Query a Model

```
// create a reference to a Resource in the model
Resource ourWillie =
  infModel.getResource("http://lyle.smu.edu/~coyle/people#willieN");

// create a reference to a Property in the model
Property ourProp =
  infModel.getProperty("http://xmlns.com/foaf/0.1/based_near");

// query model for matching statements (null = wildcard)
StmtIterator ourIterator =
    infModel.listStatements(ourWillie, ourProp, (RDFNode)null);

// use Java iterator to get results
while (ourIterator.hasNext()) {
    Statement resultStmt = ourIterator.next();
    System.out.println(resultStmt);
    System.out.println("Subject   : " + resultStmt.getSubject());
    System.out.println("Predicate : " + resultStmt.getPredicate());
    System.out.println("Object    : " + resultStmt.getObject());

}
```

```java
// use Java iterator to get results
while (ourIterator.hasNext()) {
    Statement resultStmt = ourIterator.next();
    System.out.println(resultStmt);
    System.out.println("Subject   : " + resultStmt.getSubject());
    System.out.println("Predicate : " + resultStmt.getPredicate());
    System.out.println("Object    : " + resultStmt.getObject());

}
```

```
[http://lyle.smu.edu/~coyle/people#willieN, http://xmlns.com/foaf/0.1/based_near,
                                            http://dbpedia.org/resource/Austin]
Subject   : http://lyle.smu.edu/~coyle/people#willieN
Predicate : http://xmlns.com/foaf/0.1/based_near
Object    : http://dbpedia.org/resource/Austin
```

Design Pattern:
Separate Facts from Ontology

```
String myUrl4  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/musicFacts.n3";
String myUrl5  =
   "http://lyle.smu.edu/~coyle/semweb/semtech11/musicOnto.n3";


Model basicModel = ModelFactory.createDefaultModel();

basicModel.read(myUrl4, "N3");       // load data
basicModel.read(myUrl5, "N3");       // load ontology

// Create RDFS InfModel using basicModel
InfModel infModel =
  ModelFactory.createRDFSModel(basicModel);
```

musicfacts.n3

```
@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     :   <http://lyle.smu.edu/~coyle/people#> .

# Willie Nelson
:willieN  rdf:type       :Musician.
:willieN  foaf:name      "Willie Nelson" .
:willieN  foaf:firstName "Willie" .
:willieN  foaf:surname   "Nelson" .
:willieN  foaf:based_near <http://dbpedia.org/resource/Austin> .
:willieN  :playedOn       :Highwaymen .

# Johnny Cash
:johnnyC  foaf:name      "Johnny Cash" .
:johnnyC  foaf:firstName "Johnny" .
:johnnyC  foaf:surname   "Cash" .
:johnnyC  :playedOn      :Highwaymen .

# Waylon Jennings
:waylonJ  foaf:name      "Waylon Jennings" .
:waylonJ  foaf:firstName "Waylon" .
:waylonJ  foaf:surname   "Jennings" .
:waylonJ  :playedOn      :Highwaymen .
```

musicOnto.n3

```
@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     :   <http://lyle.smu.edu/~coyle/people#> .

# Ontology : Subclass relationship
:Musician   rdfs:subClassOf foaf:Person .

#Ontology  : Properties
:playedOn  rdfs:domain  :Musician .
:playedOn  rdfs:range   :Album .
```

rdfs inferencing using two files

```
basicModel.read(myUrl4, "N3");        // load data
basicModel.read(myUrl5, "N3");        // load ontology
```

```
# Johnny Cash
:johnnyC  foaf:name       "Johnny Cash" .
:johnnyC  foaf:firstName "Johnny" .
:johnnyC  foaf:surname    "Cash" .
:johnnyC  :playedOn       :Highwaymen .
```

```
# Ontology : Subclass relationship
:Musician   rdfs:subClassOf foaf:Person .

#Ontology  : Properties
:playedOn  rdfs:domain  :Musician .
:playedOn  rdfs:range   :Album .
```

```
:johnnyC
    a       foaf:Person , :Musician , rdfs:Resource ;
    :playedOn :Highwaymen ;
    foaf:firstName "Johnny" ;
    foaf:name "Johnny Cash" ;
    foaf:surname "Cash" .
```

Jena
and
OWL Inferencing

OWL Inferencing

Capabilities include:

- owl:SymmetricProperty
- owl:TransitiveProperty
- owl:InverseProperty

n3 schema file with owl constructs

```
@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     :   <http://lyle.smu.edu/~coyle/people#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .

# Ontology : Subclass relationship
:Musician   rdfs:subClassOf foaf:Person .

#Ontology  : Properties
:playedOn  rdfs:domain  :Musician .
:playedOn  rdfs:range   :Album .

#Owl constructs
:playedOn owl:inverseOf :includesPlayer .
:friendOf rdf:type owl:SymmetricProperty .
```

Working with OWL

```
// load N3 fact data into a Model instance
Model factModel = ModelFactory.createDefaultModel();
factModel.read(myUrl4, "N3");       // load data

// load N3 ontology data using alternate FileManager technique
Model schema =
   FileManager.get().loadModel(myUrl6, "N3");

// Create OWL reasoner
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
// Load ontology
reasoner = reasoner.bindSchema(schema);

// Create Inference Model
InfModel infModel = ModelFactory.createInfModel(reasoner, factModel);
```

Output from InfModel

```
:johnnyC
        a    foaf:Person , :Musician , rdfs:Resource ;
        :friendOf :willieN ;
        :playedOn :Highwaymen ;
        foaf:firstName "Johnny" ;
        foaf:name "Johnny Cash" ;
        foaf:surname "Cash" .
```

owl:SymmetricProperty

```
:Highwaymen
      a        rdfs:Resource , :Album ;
      :includesPlayer :johnnyC , :waylonJ , :willieN .
```

:playedOn owl:inverseOf :includesPlayer .

Jena
Working with Rules

rules.txt

```
@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     :   <http://lyle.smu.edu/~coyle/people#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#> .


# Rollo admires anyone who played on the Highwaymen album
[rule1:  (?x :playedOn :Highwaymen) -> (:rollo :admires ?x)]

# Anyone who is a Musician and friend of Johnny Cash is a friend of Rollo
[rule2:  (?y :friendOf :johnnyC), (?y rdf:type :Musician) ->
                                                (:rollo :friendOf ?y)]
```

More rules…

A driver is young if age is between 18 and 25 years old.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix ex: http://example.com/
@prefix xs: http://www.w3.org/2001/XMLSchema#
[youngDriver: (?d rdf:type ex:YoungDriver)
              <-
              (?d rdf:type ex:Driver)
              (?d ex:age ?a)
              greaterThan(?a,18)
              lessThan(?a,25)]
```

A driver is typical if he is neither young driver or senior driver.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix ex: http://example.com/
[eligibleDriver: (?d rdf:type ex:TypicalDriver)
                 <-
                 (?d rdf:type ex:Driver)
                 noValue(?d rdf:type ex:YoungDriver)
                 noValue(?d rdf:type ex:SeniorDriver)]
```

Working with Rules

```
Rule files may be loaded and parsed using:

List rules = Rule.rulesFromURL("file:myfile.rules");

or

BufferedReader br = /* open reader */ ;
List rules = Rule.parseRules( Rule.rulesParserFromReader(br) );

or

String ruleSrc = /* list of rules in line */
List rules = Rule.parseRules( rulesSrc );

In the first two cases (reading from a URL or a BufferedReader)
the rule file is preprocessed by a simple processor which strips
comments and supports some additional macro commands:
```

```java
// Create OWL reasoner
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
// Load ontology
reasoner = reasoner.bindSchema(schema);

// Create Inference Model
InfModel infModel = ModelFactory.createInfModel(reasoner, factModel);

//create resource to hold rules
Resource ruleResource = infModel.createResource();
ruleResource.addProperty(ReasonerVocabulary.PROPruleSet, "rules.txt");

// create Reasoner to handle the rules
Reasoner ruleReasoner =
  GenericRuleReasonerFactory.theInstance().create(ruleResource);

infModel = ModelFactory.createInfModel(ruleReasoner, infModel);

// show facts
infModel.write(System.out, "N3"); // write triples
```

```
:rollo
      :admires :johnnyC , :waylonJ , :willieN ;
      :friendOf :willieN .
```

Jena and SPARQL
the RDF Query language

Selects name and email of any foaf:Person with a name and email.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Other SPARQL Queries

Query Forms

[edit]

The SPARQL language specifies four different query variations for different purposes.

SELECT query

Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.

CONSTRUCT query

Used to extract information from the SPARQL endpoint and transform the results into valid RDF.

ASK query

Used to provide a simple True/False result for a query on a SPARQL endpoint.

DESCRIBE query

Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Each of these query forms takes a WHERE block to restrict the query although in the case of the DESCRIBE query the WHERE is optional.

Jena API

Java String containing SPARQL query

```
"PREFIX..
..
SELECT ?x
WHERE { ... }  "
```

QueryFactory

QueryExecutionFactory

Model

Jena API

Java String containing SPARQL query

```
"PREFIX..
..
SELECT ?x
WHERE { ... }  "
```

QueryFactory

create(queryStr)

Query
object
(knows query)

QueryExecutionFactory

Model

Jena API

Java String containing SPARQL query

```
"PREFIX..
..
SELECT ?x
WHERE { ... }  "
```

QueryFactory

Query
object
(knows query)

QueryExecutionFactory

create(myQuery, myModel)

Model

Jena API

Java String containing SPARQL query

```
"PREFIX..
..
SELECT ?x
WHERE { ... }  "
```

QueryFactory

Query object (knows query)

QueryExecutionFactory

create(myQuery, myModel)

Model

QueryExecution object

ResultSet

Jena Query and QueryExecution objects

```
// Create new SPARQL Query
String queryStr =
        "PREFIX drc: <http://lyle.smu.edu/~coyle/people#> " +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "SELECT ?x " +
        "WHERE {" +
        " ?x drc:playedOn drc:Highwaymen " +
        " }";

//Create Jena Query from string
Query myQuery = QueryFactory.create(queryStr);

// Create QueryExecution object to run query over a Model
QueryExecution qe = QueryExecutionFactory.create(myQuery, infModel);

// Execute the SELECT query -- returns ResultSet
ResultSet rs = qe.execSelect();
```

Jena Query and QueryExecution objects

```
// Create new SPARQL Query
String queryStr =
        "PREFIX drc: <http://lyle.smu.edu/~coyle/people#> " +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "SELECT ?x " +
        "WHERE {" +
        " ?x drc:playedOn drc:Highwaymen " +
        " }";

//Create Jena Query from string
Query myQuery = QueryFactory.create(queryStr);

// Create QueryExecution object to run query over a Model
QueryExecution qe = QueryExecutionFactory.create(myQuery, infModel);

// Execute the SELECT query -- returns ResultSet
ResultSet rs = qe.execSelect();
```

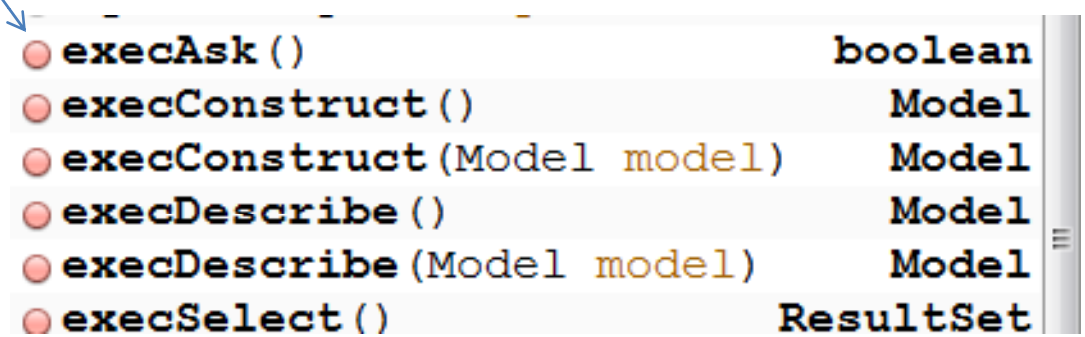
Process ResultSet

```
// Iterate over results
while(rs.hasNext() ) {
    QuerySolution sol = rs.next();
    System.out.println("Result: " + sol.toString() );
}
```

```
Result: ( ?x = <http://lyle.smu.edu/~coyle/people#waylonJ> )
Result: ( ?x = <http://lyle.smu.edu/~coyle/people#johnnyC> )
Result: ( ?x = <http://lyle.smu.edu/~coyle/people#willieN> )
```

Other SPARQL Query Options

```
// Create QueryExecution object to run query over a Model
QueryExecution qe = QueryExecutionFactory.create(myQuery, infModel);
```



Summary

- Java Jena may be used in a variety of ways to create Semantic Web apps
- RDF -> N3
- N3 -> RDF
- Inferencing with RDFS/OWL/DAML+OIL
- Rule Processing
- SPARQL Queries

Questions

??????