Jena

RDF Stores

- **Jena** (jena.semanticweb.org)
  - Popular RDF store
  - RDF and RDFS querying
  - Limited OWL reasoning
  - Forward chaining and backward chaining rule engines
  - Open source implementation in Java
  - Command-line and Java API access
- Sesame (sesame.semanticweb.org)
  - Scalable RDF store
  - Open source implementation in Java
  - RDF and RDFS querying
  - Limited OWL reasoning
  - Forward chaining rules engine
  - Java API and HTTP access
- RDFStore
  - C-based RDF store
  - RDQL support

RDF and RDF API in Jena 2

- Framework developed by HP Labs for manipulating with metadata in Java applications
- Two versions:
  - Jena 1
    - Expressive support for RDF
    - Limited reasoning facilities (RDQL)
  - Jena 2
    - Ontology API included
    - Support for OWL included

Jena

- Includes components:
  - API for RDF (ARP: Another RDF parser)
  - API for ontologies with support for OWL, DAML and RDF schema
  - Reasoners
  - Persistent storage support
  - SPARQL: query language for RDF

Jena Installation

- Step1: Download Jena version 2.5.5 (http://jena.sourceforge.net/downloads.html) to your home directory (say C:\Jena)
- Step2: Unzip the file on the same folder, it will create the directory structure:
  - C:\Jena\Jena-2.5.5
- Step3: Set the classpath, put the lib of Jena to classpath:

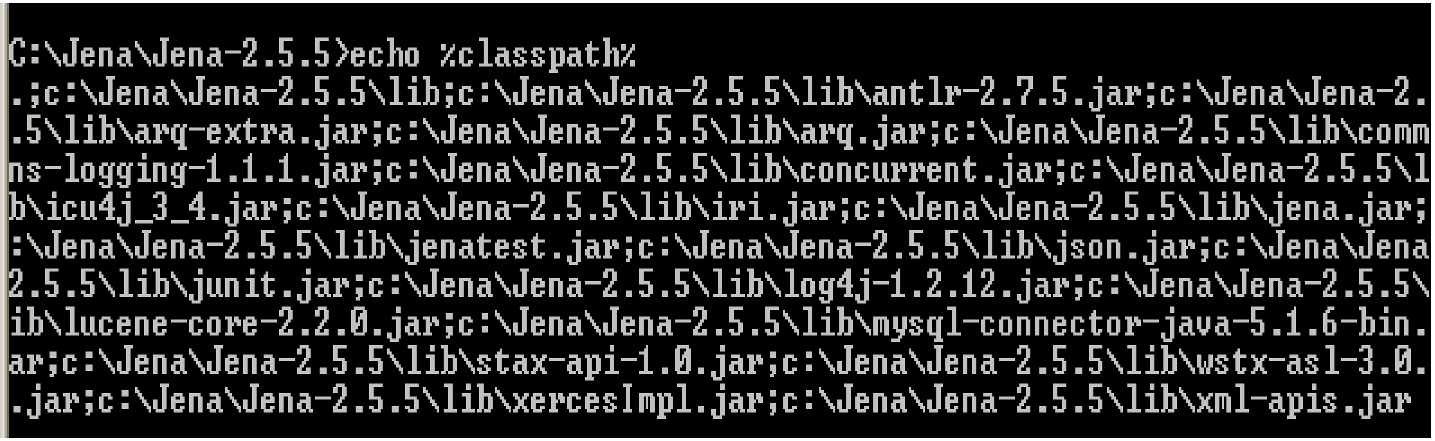
Set up many classpaths

- store setcp.bat in C:\Jena
- C:\Jena>setcp.b at C:\Jena\Jena- 2.5.5\lib
- C:\Jena>echo %classpath%

setcp.bat

```
@echo off
set CLASSPATH=
call :setall . %*
goto end

:setall
if .%1.==.. goto end
set dir=%1
set dir=%dir:"=%
if not "%CLASSPATH%"=="" set CLASSPATH=%CLASSPATH%;%dir%
if "%CLASSPATH%"=="" set CLASSPATH=%dir%
for %%i in ("%dir%\*.jar") do call :setone "%%i"
for %%i in ("%dir%\*.zip") do call :setone "%%i"
shift
goto setall

:setone
set file=%1
set file=%file:"=%
set CLASSPATH=%CLASSPATH%;%file%

:end
```

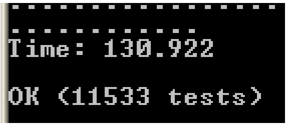
Jena classpaths



```
C:\Jena\Jena-2.5.5>echo %classpath%
.;c:\Jena\Jena-2.5.5\lib;c:\Jena\Jena-2.5.5\lib\antlr-2.7.5.jar;c:\Jena\Jena-2.
.5\lib\arq-extra.jar;c:\Jena\Jena-2.5.5\lib\arq.jar;c:\Jena\Jena-2.5.5\lib\comm
ns-logging-1.1.1.jar;c:\Jena\Jena-2.5.5\lib\concurrent.jar;c:\Jena\Jena-2.5.5\l
b\icu4j_3_4.jar;c:\Jena\Jena-2.5.5\lib\iri.jar;c:\Jena\Jena-2.5.5\lib\jena.jar;
:\Jena\Jena-2.5.5\lib\jenatest.jar;c:\Jena\Jena-2.5.5\lib\json.jar;c:\Jena\Jena
2.5.5\lib\junit.jar;c:\Jena\Jena-2.5.5\lib\log4j-1.2.12.jar;c:\Jena\Jena-2.5.5\
ib\lucene-core-2.2.0.jar;c:\Jena\Jena-2.5.5\lib\mysql-connector-java-5.1.6-bin.
ar;c:\Jena\Jena-2.5.5\lib\stax-api-1.0.jar;c:\Jena\Jena-2.5.5\lib\wstx-asl-3.0.
.jar;c:\Jena\Jena-2.5.5\lib\xercesImpl.jar;c:\Jena\Jena-2.5.5\lib\xml-apis.jar
```

Jena Installation

- Step4: test Jena by running the regression tests:
  - test.bat (on windows)
  - Regression testing is any type of software testing which seeks to uncover regression bugs.
    - Regression bugs occur when software previously worked as desired, stops working or no longer works in the same way that was previously planned.



```
.................
............
Time: 130.922

OK (11533 tests)
```

Jena RDF API

http://jena.sourceforge.net/tutorial/RDF_API/

Setting up

- Properly install Jena at Jena_home
- Go to Jena_home\doc\tutorial\RDF_API\index.html
- All tutorial java files are located at:
  - Jena_home\src-examples\jena\examples\rdf\
- Create a new folders as:
  - C:\Jena\Tutorial

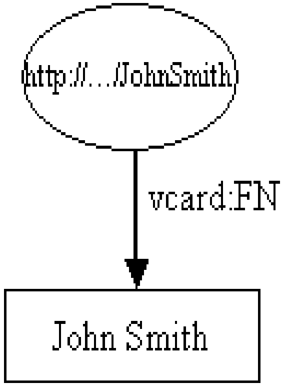
Jena Tutorials

- Jena is a Java API which can be used to create and manipulate RDF graphs.
- Jena has object classes to represent graphs, resources, properties and literals.
- The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively.
- In Jena, a graph is called a model and is represented by the model interface.

Tutorial01 – Representing a RDF graph

```
// some definitions
static String personURI    =
"http://somewhere/JohnSmith";
static String fullName     = "John Smith";

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
Resource johnSmith =
model.createResource(personURI);

// add the property
 johnSmith.addProperty(VCARD.FN, fullName);
```

Tutorial01.java



Representing a RDF graph

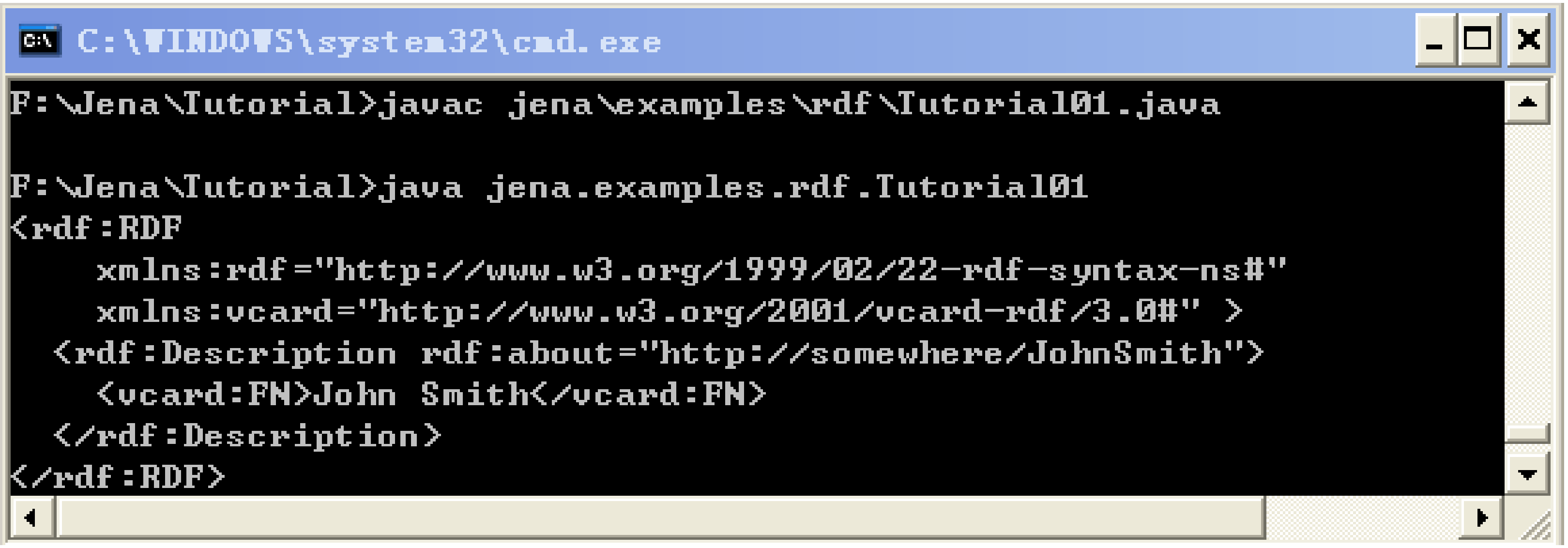
- It begins with some constant definitions and then creates an empty Model or model, using the ModelFactory method createDefaultModel() to create a memory-based model.
- Jena contains other implementations of the Model interface, e.g one which uses a relational database: these types of Model are also available from ModelFactory.
- The John Smith resource is then created and a property added to it. The property is provided by a "constant" class VCARD which holds objects representing all the definitions in the VCARD schema.
- Jena provides constant classes for other well known schemas, such as RDF and RDF schema themselves, Dublin Core and DAML.

```
package jena.examples.rdf ;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;

public class Tutorial01 extends Object {
    // some definitions
    static String personURI    = "http://somewhere/JohnSmith";
    static String fullName     = "John Smith";

      public static void main (String args[]) {
        // create an empty model
        Model model = ModelFactory.createDefaultModel();

       // create the resource
       Resource johnSmith = model.createResource(personURI);

      // add the property
      johnSmith.addProperty(VCARD.FN, fullName);

      //write the model in XML form to a file
      model.write(System.out);
      }
}
```

Tutorial01.java

Compile and Run Tutorial01

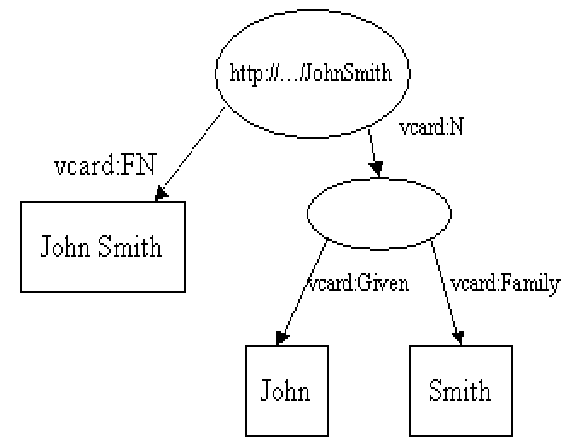
- Store Tutorial01.java in C:\Jena\Tutorial\jena\examples\rdf (because of the package stated in Tutorial01.java)
- Compile and Run



```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial01.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial01
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description rdf:about="http://somewhere/JohnSmith">
    <vcard:FN>John Smith</vcard:FN>
  </rdf:Description>
</rdf:RDF>
```

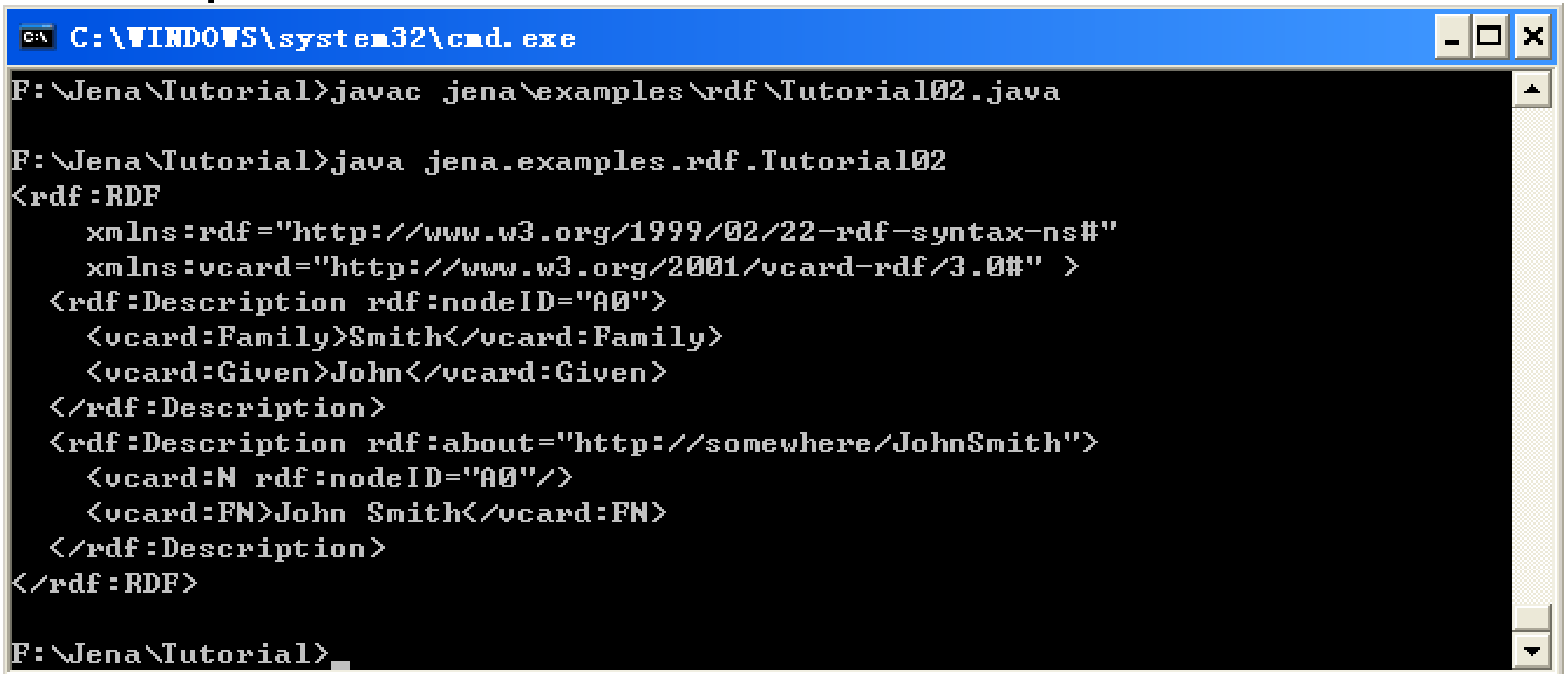
Tutorial02- RDF Graph with blank node

```
// some definitions
String personURI    = "http://somewhere/JohnSmith";
String givenName    = "John";
String familyName   = "Smith";
String fullName     = givenName + " " + familyName;

// create an empty Model
Model model = ModelFactory.createDefaultModel();

// create the resource
//   and add the properties cascading style
Resource johnSmith
  = model.createResource(personURI)
         .addProperty(VCARD.FN, fullName)
         .addProperty(VCARD.N,
                      model.createResource()
                           .addProperty(VCARD.Given,

         givenName)
                           .addProperty(VCARD.Family,

         familyName));
```



Compile and Run Tutorial02

- Store Tutorial02.java in C:\Jena\Tutorial\jena\examples\rdf
- Compile and Run



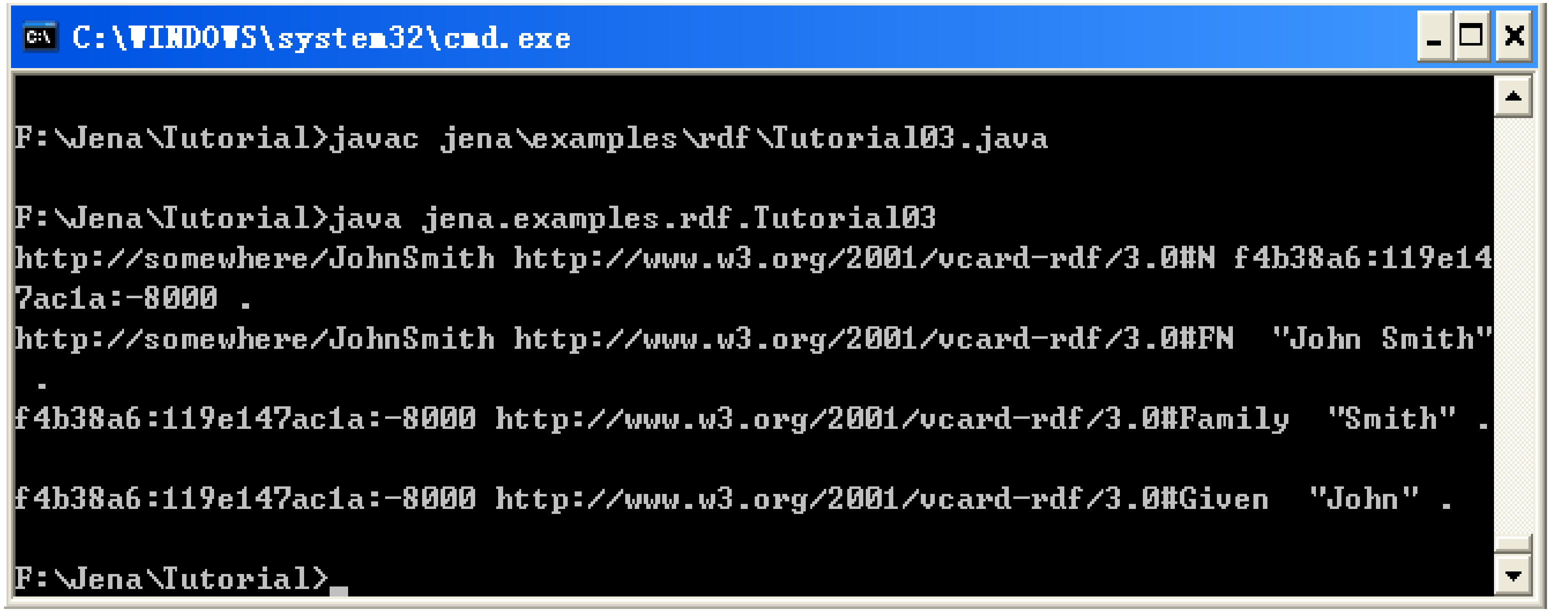
```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial02.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial02
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere/JohnSmith">
    <vcard:N rdf:nodeID="A0"/>
    <vcard:FN>John Smith</vcard:FN>
  </rdf:Description>
</rdf:RDF>

F:\Jena\Tutorial>_
```

Tutorial03- Statement

```
// list the statements in the Model
StmtIterator iter = model.listStatements();

// print out the predicate, subject and object of each statement
while (iter.hasNext()) {
    Statement stmt      = iter.nextStatement();  // get next statement
    Resource  subject   = stmt.getSubject();     // get the subject
    Property  predicate = stmt.getPredicate();   // get the predicate
    RDFNode   object    = stmt.getObject();      // get the object

    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
       System.out.print(object.toString());
    } else {
        // object is a literal
        System.out.print(" \"" + object.toString() + "\"");
    }

    System.out.println(" .");
}
```

Compile and Run Tutorial03

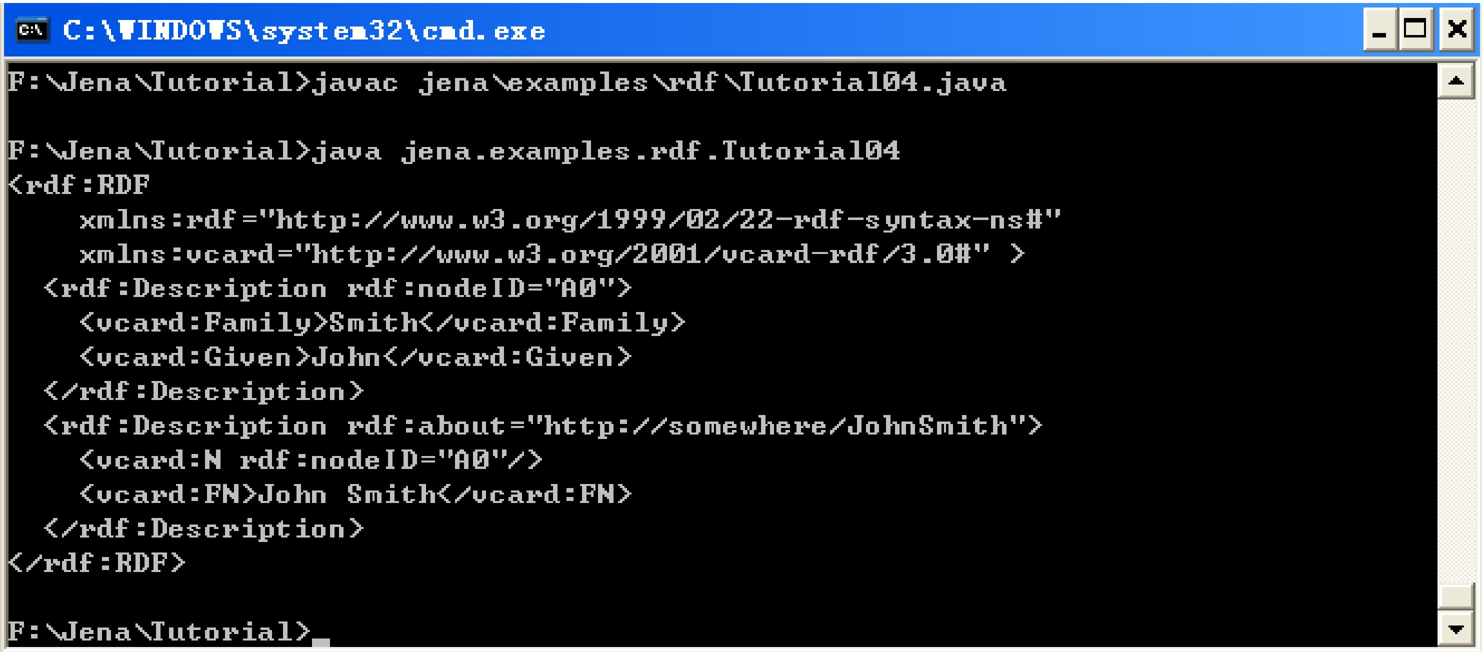
- Store Tutorial03.java in C:\Jena\Tutorial\jena\examples\rdf
- Compile and Run



```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial03.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial03
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N f4b38a6:119e14
7ac1a:-8000 .
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN  "John Smith"
 .
f4b38a6:119e147ac1a:-8000 http://www.w3.org/2001/vcard-rdf/3.0#Family  "Smith" .

f4b38a6:119e147ac1a:-8000 http://www.w3.org/2001/vcard-rdf/3.0#Given  "John" .

F:\Jena\Tutorial>_
```

Tutorial04 – output in RDF/XML

- Write the result of Tutorial03 in RDF/XML:
  - model.write(System.out);
- Store Tutorial04.java in C:\Jena\Tutorial\jena\examples\rdf
- Compile and Run



C:\WINDOWS\system32\cmd.exe

```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial04.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial04
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere/JohnSmith">
    <vcard:N rdf:nodeID="A0"/>
    <vcard:FN>John Smith</vcard:FN>
  </rdf:Description>
</rdf:RDF>

F:\Jena\Tutorial>_
```

Tutorial04 – output in other formats

- Write the result of Tutorial03 in:
  - XML: model.write(System.out, "RDF/XML-ABBREV");
  - N-Triple: model.write(System.out, "N-TRIPLE");

Tutorial05 – Reading RDF

- Read a RDF from a file and write it out

```
// create an empty model
 Model model = ModelFactory.createDefaultModel();

// use the FileManager to find the input file
 InputStream in = FileManager.get().open( inputFileName );
if (in == null) {
    throw new IllegalArgumentException(
              "File: " + inputFileName + " not found");
}

// read the RDF/XML file
model.read(in, "");

// write it to standard out
model.write(System.out);
```

Tutorial05 – Reading RDF

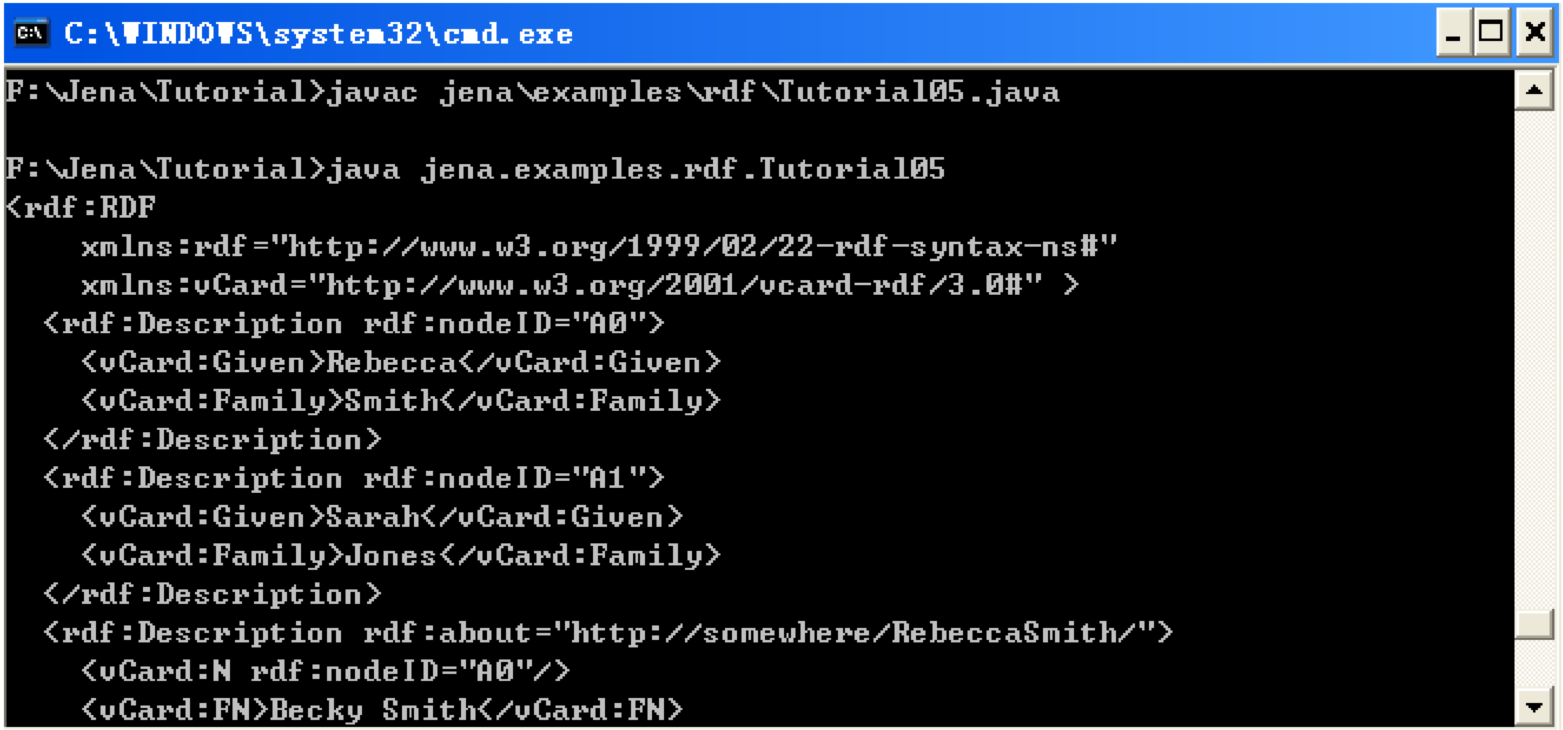
- Source rdf (vc-db-1.rdf) to be read:

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vCard='http://www.w3.org/2001/vcard-rdf/3.0#'
   >

  <rdf:Description rdf:about="http://somewhere/JohnSmith/">
    <vCard:FN>John Smith</vCard:FN>
    <vCard:N rdf:parseType="Resource">
         <vCard:Family>Smith</vCard:Family>
         <vCard:Given>John</vCard:Given>
    </vCard:N>
  </rdf:Description>

  <rdf:Description rdf:about="http://somewhere/RebeccaSmith/">
    <vCard:FN>Becky Smith</vCard:FN>
    <vCard:N rdf:parseType="Resource">
         <vCard:Family>Smith</vCard:Family>
         <vCard:Given>Rebecca</vCard:Given>
    </vCard:N>
  </rdf:Description>
.  .  .
</rdf:RDF>
```

rdf:parseType=“Resource” is used to represent blank nodes.

Tutorial05 – Reading RDF

- Store Tutorial05.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-1.rdf in C:\Jena\Tutorial\ (it has to be in this folder)
- Compile and Run



```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial05.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial05
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description rdf:nodeID="A0">
    <vCard:Given>Rebecca</vCard:Given>
    <vCard:Family>Smith</vCard:Family>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <vCard:Given>Sarah</vCard:Given>
    <vCard:Family>Jones</vCard:Family>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere/RebeccaSmith/">
    <vCard:N rdf:nodeID="A0"/>
    <vCard:FN>Becky Smith</vCard:FN>
```

Tutorial06 – Navigating a Model

- Deal with accessing information held in a Model
- Given the URI of a resource, the resource object can be retrieved from a model using Model.getResource(String uri) method.
- This method is defined to return a Resource object if one exists in the model, or otherwise to create a new one.
- For example:
  - // retrieve the John Smith vcard resource from the model
    Resource vcard = model.getResource(johnSmithURI);

Tutorial06 – Navigating a Model

- Accessing properties of a resource
  - Resource.getProperty(Property p)
  - This method returns the whole statement.
  - Then using getObject() to get the value of the property.
  - Example
    - // retrieve the value of the N property

```
Resource name = (Resource) vcard.getProperty(VCARD.N)
                                            .getObject();
```

Tutorial06 – Navigating a Model

- The object of a statement could be a resource or a literal
  - Knowing the value to be a resource
    - // retrieve the value of the FN property

      ```
      Resource name = vcard.getProperty(VCARD.N)
                                      .getResource();
      ```
  - Knowing the value to be literal
    - // retrieve the given name property

      ```
      String fullName = vcard.getProperty(VCARD.FN)
                                   .getString();
      ```

Tutorial06 – Navigating a Model

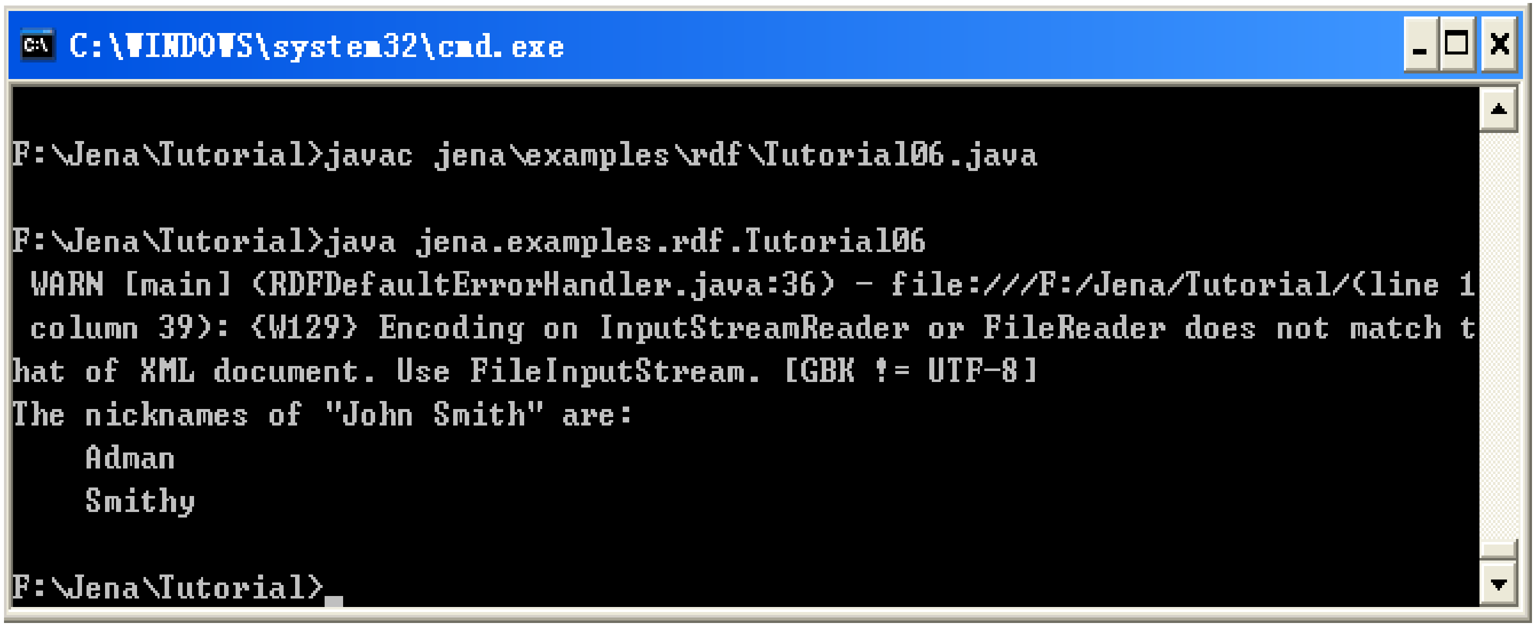
- RDF permits a resource to repeat a property,
  - **e.g. // add two nickname properties to vcard vcard.addProperty(VCARD.NICKNAME, "Smithy")**

    ```
    .addProperty(VCARD.NICKNAME, "Adman");
    ```
- The result of calling vcard.getProperty(VCARD.NICKNAME) is indeterminate. Jena will return one of the values.
- It is possible to list all the properties by using Resource.listProperties(Property p)
  - StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);

    ```
    while (iter.hasNext()) {
      System.out.println(" " + iter.nextStatement()
                                          .getObject()
                                          .toString());
      }
    ```

Tutorial06 – Navigating a Model

- Store Tutorial06.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-1.rdf in C:\Jena\Tutorial\ (it has to be in this folder)
- Compile and Run



```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial06.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial06
 WARN [main] (RDFDefaultErrorHandler.java:36) - file:///F:/Jena/Tutorial/(line 1
 column 39): {W129} Encoding on InputStreamReader or FileReader does not match t
hat of XML document. Use FileInputStream. [GBK != UTF-8]
The nicknames of "John Smith" are:
    Adman
    Smithy

F:\Jena\Tutorial>_
```

Tutorial07-Query a Model I

- Here introduces some limited query primitive
  - model.listStatements(): lists all the statements in a model, not recommended on large models
    - model.listStatements(Selector s): returns an iterator over all the statements in the model selected by s.
  - model.listSubjects(): returns an iterator over all resources which are subjects of the statements
  - model.listSubjectsWithProperty(Property p, RDFNode o): returns an iterator over all the resources which have property p with value o.

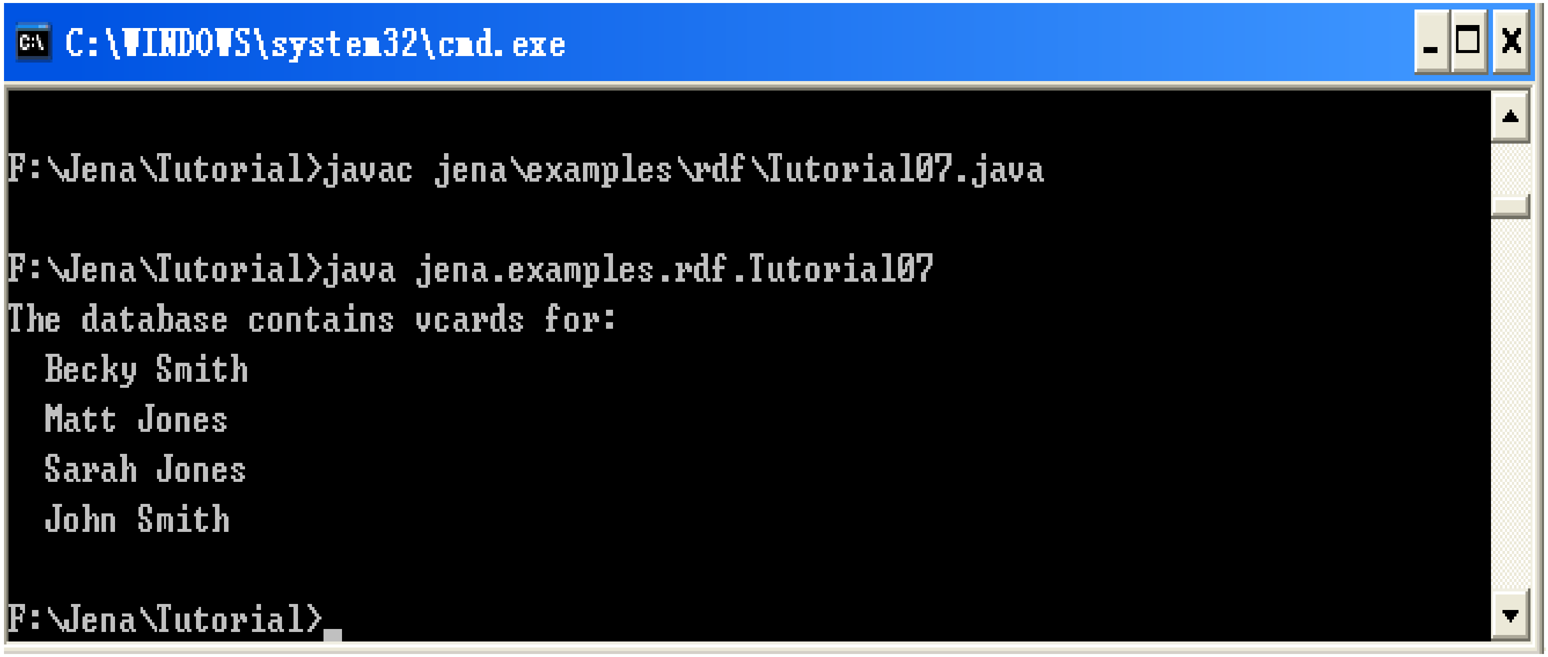
```
// list vcards
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
while (iter.hasNext()) {
    Resource r = iter.nextResource();
    ...
}
```

Tutorial07-Query a Model I

- Select subject, predicate and object
  - Selector selector = new SimpleSelector(subject, predicate, object):
  - It will select all the statements with a subject that matches subject, a predicate that matches predicate and an object that matches object.
  - If a null is supplied in any of the positions, it matches anything.
    - Selector selector = new SimpleSelector(null, null, null)
  - Example:
    - Selector selector = new SimpleSelector(null, VCARD.FN, null): will select all the statements with VCARD.FN as their predicate, whatever the subject or object.

Tutorial07-Query a Model I

- Store Tutorial07.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-1.rdf in C:\Jena\Tutorial\ (it has to be in this folder)
- Compile and Run



```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial07.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial07
The database contains vcards for:
  Becky Smith
  Matt Jones
  Sarah Jones
  John Smith

F:\Jena\Tutorial>_
```

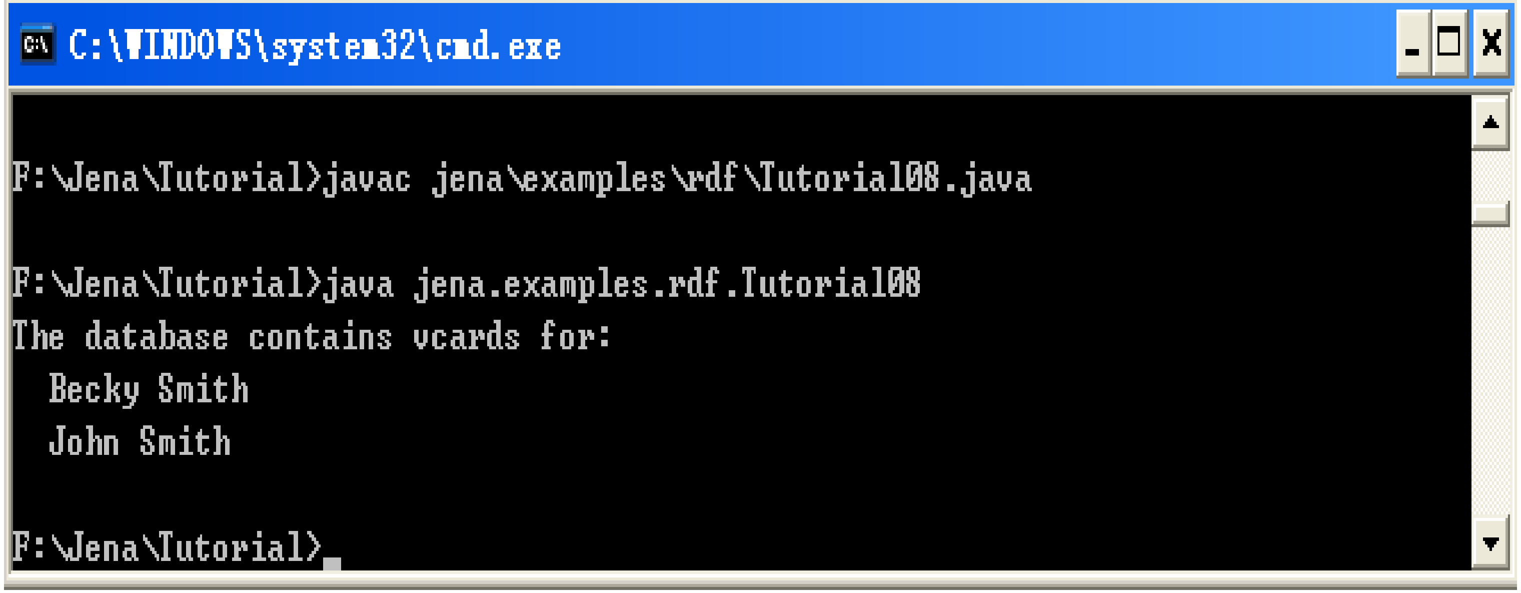
Tutorial08-Query a Model II

- Lets see some finer control over the selected statements

```
// select all the resources with a VCARD.FN property
// whose value ends with "Smith"
StmtIterator iter = model.listStatements(
    new SimpleSelector(null, VCARD.FN, (RDFNode) null) {
        public boolean selects(Statement s)
            {return s.getString().endsWith("Smith");}
    });
```

Tutorial08-Query a Model II

- Store Tutorial08.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-1.rdf in C:\Jena\Tutorial\ (it has to be in this folder)
- Compile and Run

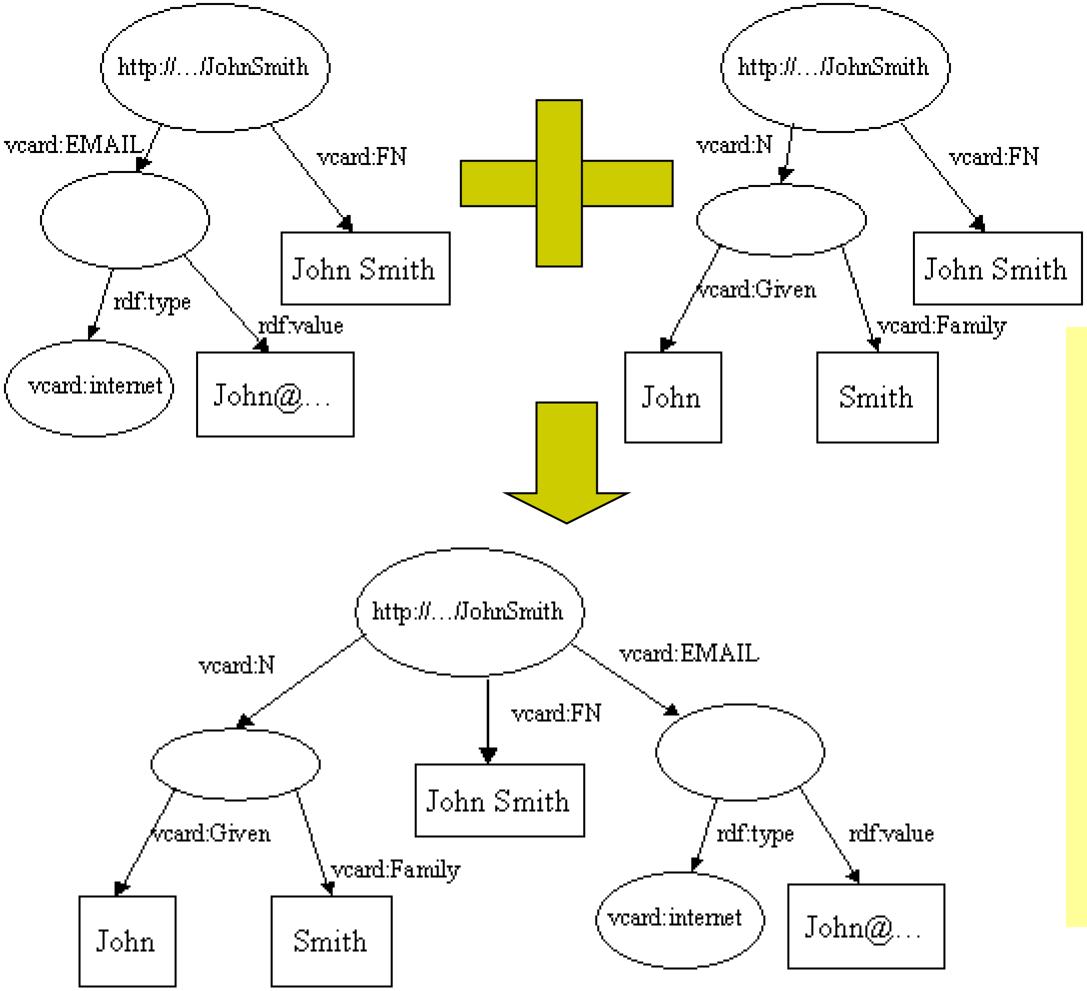


```
F:\Jena\Tutorial>javac jena\examples\rdf\Tutorial08.java

F:\Jena\Tutorial>java jena.examples.rdf.Tutorial08
The database contains vcards for:
  Becky Smith
  John Smith

F:\Jena\Tutorial>_
```

Tutorial09-Operations on Models

- Jena provides three operations for manipulating models:
  - Union - .union(Model): creates a new model containing all the statements in this model together with all of those in another given model.
    - It can merge data from different data sources
  - Intersection - .intersection(Model): create a new model containing all the statements which are in both this model and another
  - Difference - .difference(Model): create a new model containing all the statements in this model which are not in another.

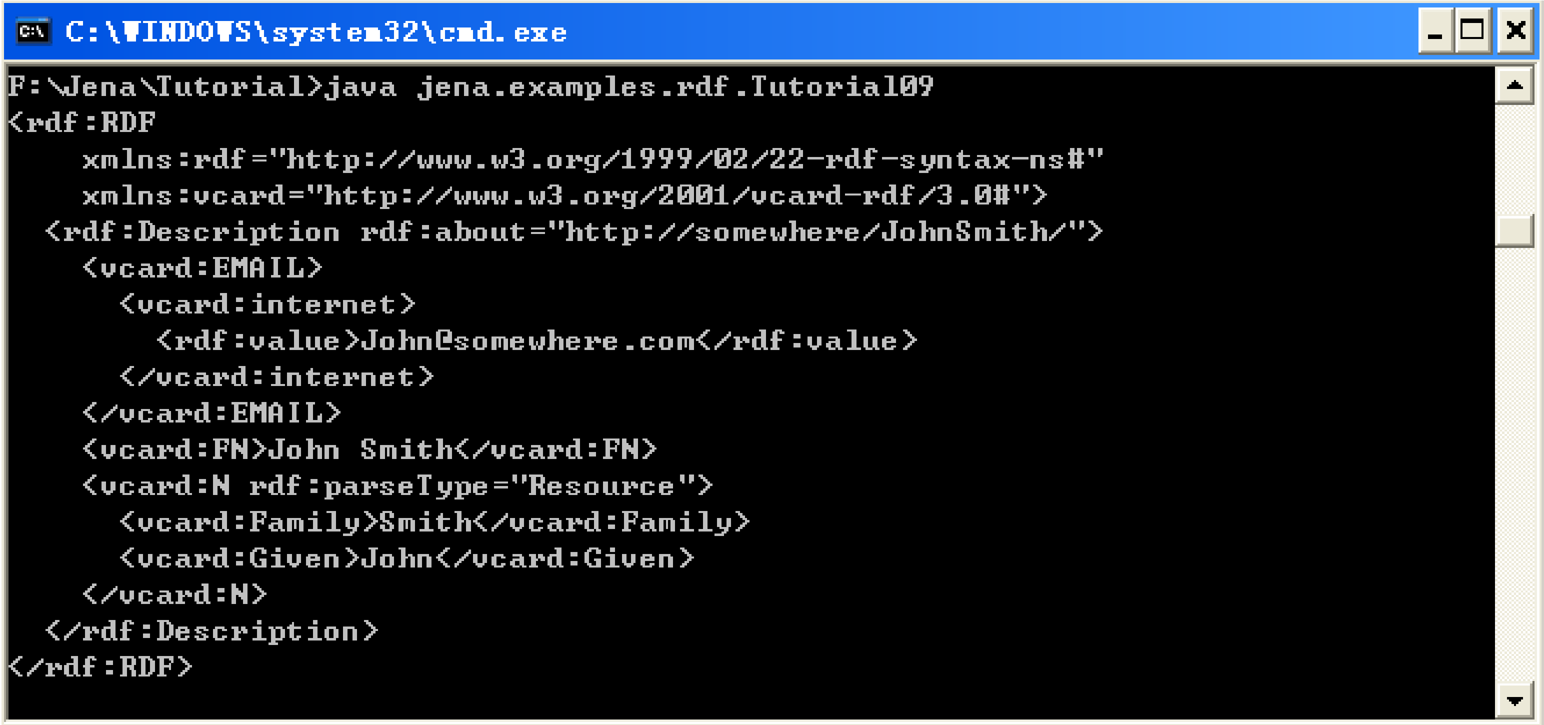
Tutorial09-Operations on Models



```
// read the RDF/XML files
model1.read(new
InputStreamReader(in1), "");
model2.read(new
InputStreamReader(in2), "");

// merge the Models
Model model =
model1.union(model2);

// print the Model as RDF/XML
model.write(system.out,
"RDF/XML-ABBREV");
```

Tutorial09-Operations on Models

- Store Tutorial09.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-3.rdf and vc-db-4.rdf in C:\Jena\Tutorial\
- Compile and Run

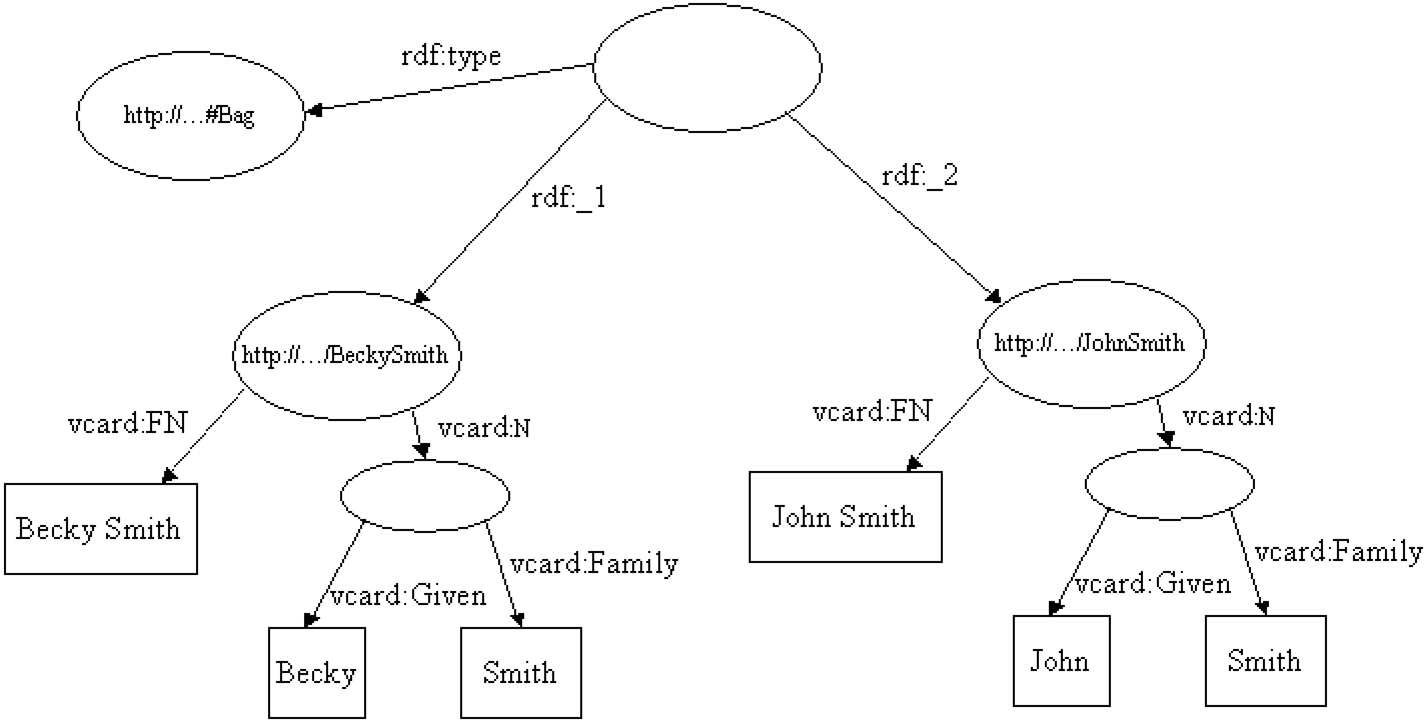


```
F:\Jena\Tutorial>java jena.examples.rdf.Tutorial09
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about="http://somewhere/JohnSmith/">
    <vcard:EMAIL>
      <vcard:internet>
        <rdf:value>John@somewhere.com</rdf:value>
      </vcard:internet>
    </vcard:EMAIL>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:parseType="Resource">
      <vcard:Family>Smith</vcard:Family>
      <vcard:Given>John</vcard:Given>
    </vcard:N>
  </rdf:Description>
</rdf:RDF>
```

Tutorial10 - Containers

- RDF provides containers to represent collections of things. There are three kinds of containers:
  - A BAG is an unordered collection
  - An ALT is an unordered collection for which only one selection can be made.
  - A SEQ is an ordered collection
- A container is represented by a resource, which has an rdf:type property whose value can be: rdf:Bag, rdf:Alt or rdf:Seq.
- The first number of the container is the value of the container's rdf:_1 property, rdf:_2,…rdf:_nn

Tutorial10 - Containers



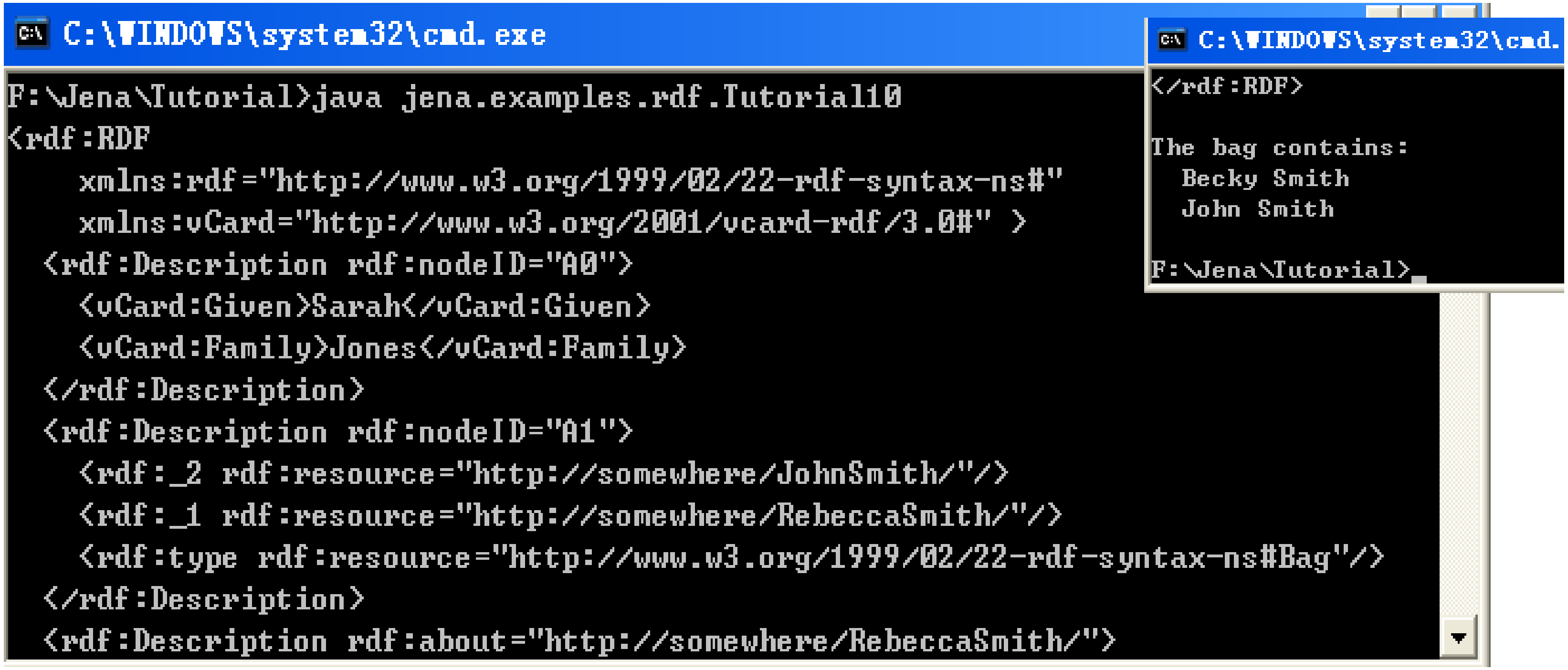
Tutorial10 - Containers

- Lets create a bag container

```
// create a bag
Bag smiths = model.createBag();

// select all the resources with a VCARD.FN property
// whose value ends with "Smith"
StmtIterator iter = model.listStatements(
    new SimpleSelector(null, VCARD.FN, (RDFNode) null) {
        public boolean selects(Statement s) {
                return s.getString().endsWith("Smith");
        }
    });
// add the Smith's to the bag
while (iter.hasNext()) {
    smiths.add(iter.nextStatement().getSubject());
}
```

Tutorial10 - Containers

- Store Tutorial10.java in C:\Jena\Tutorial\jena\examples\rdf
- Store vc-db-3.rdf and vc-db-4.rdf in C:\Jena\Tutorial\
- Compile and Run



```
F:\Jena\Tutorial>java jena.examples.rdf.Tutorial10
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description rdf:nodeID="A0">
    <vCard:Given>Sarah</vCard:Given>
    <vCard:Family>Jones</vCard:Family>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <rdf:_2 rdf:resource="http://somewhere/JohnSmith/"/>
    <rdf:_1 rdf:resource="http://somewhere/RebeccaSmith/"/>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere/RebeccaSmith/">
```

```
</rdf:RDF>

The bag contains:
  Becky Smith
  John Smith

F:\Jena\Tutorial>_
```

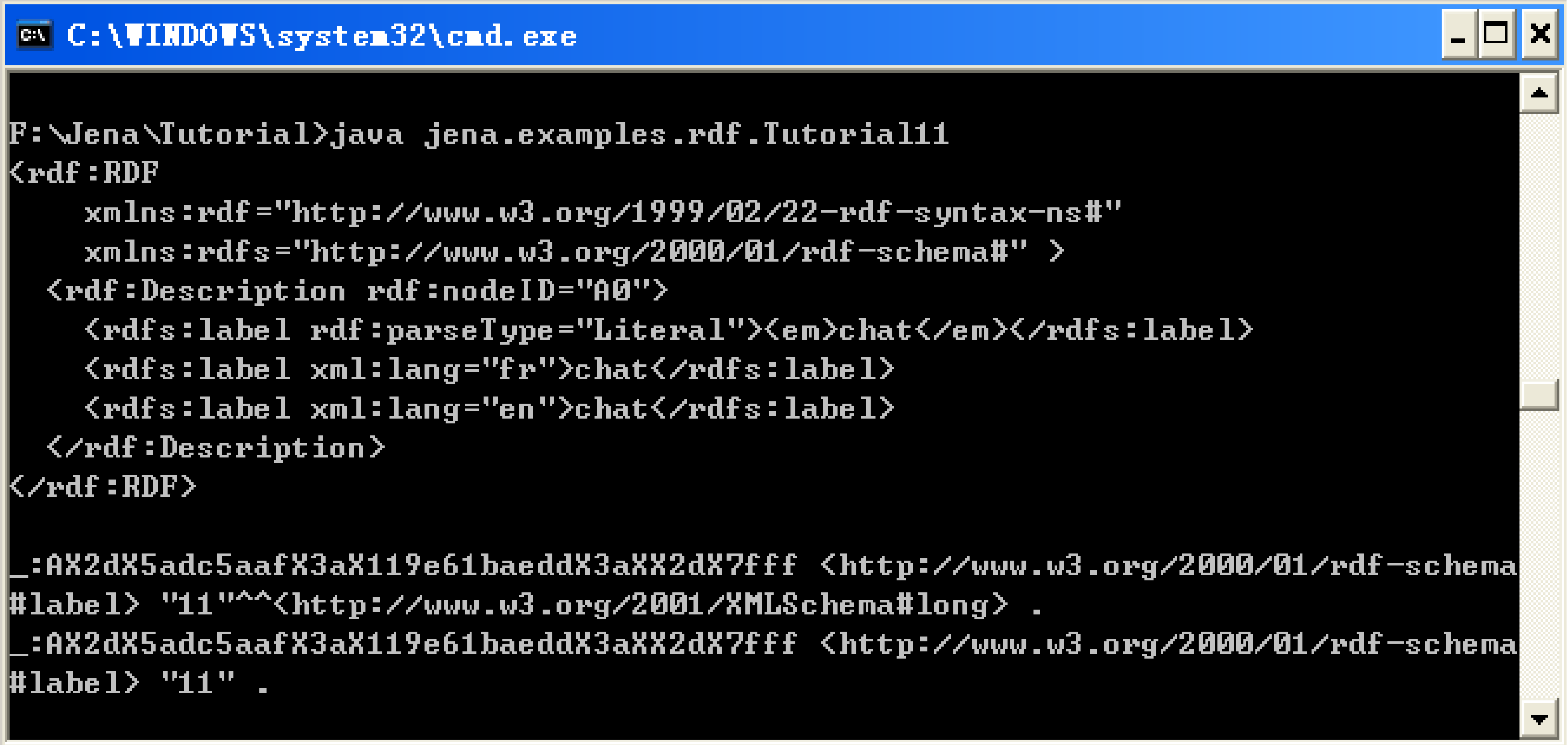
Tutorial11-Literals and Datatypes

- model.createLiteral(): create literal

```
// create the resource
Resource r = model.createResource();

// add the property
r.addProperty(RDFS.label, model.createLiteral("chat", "en"))
 .addProperty(RDFS.label, model.createLiteral("chat", "fr"))
 .addProperty(RDFS.label,
model.createLiteral("<em>chat</em>", true));

// write out the Model
model.write(system.out);
```

Tutorial11-Literals and Datatypes

- Store Tutorial11.java in C:\Jena\Tutorial\jena\examples\rdf
- Compile and Run



```
F:\Jena\Tutorial>java jena.examples.rdf.Tutorial11
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:nodeID="A0">
    <rdfs:label rdf:parseType="Literal"><em>chat</em></rdfs:label>
    <rdfs:label xml:lang="fr">chat</rdfs:label>
    <rdfs:label xml:lang="en">chat</rdfs:label>
  </rdf:Description>
</rdf:RDF>

_:AX2dX5adc5aafX3aX119e61baeddX3aXX2dX7fff <http://www.w3.org/2000/01/rdf-schema
#label> "11"^^<http://www.w3.org/2001/XMLSchema#long> .
_:AX2dX5adc5aafX3aX119e61baeddX3aXX2dX7fff <http://www.w3.org/2000/01/rdf-schema
#label> "11" .
```

Jena 2 Ontology API

http://www.srdc.metu.edu.tr/webpage/documents/jena_doc/ontology/index.html

General concepts

- Jena allows a programmer to specify, in an open, meaningful way the concepts and relationships that collectively characterise some domain.
  - E.g. red wine, grape varieties, vintage years, wineries --- wine domain classes
  - E.g. wineries produce wines, wines have a year of production – wine domain relationships
- The advantage of an ontology is that it is an explicit, first-class description, it can be published and reused for different purposes.
  - A winery may uses the wine ontology to link production schedule to the stock system
  - A wine recommendation program may use the wine ontology to recommend wines for different menus.

Ontology languages and Jena Ontology API

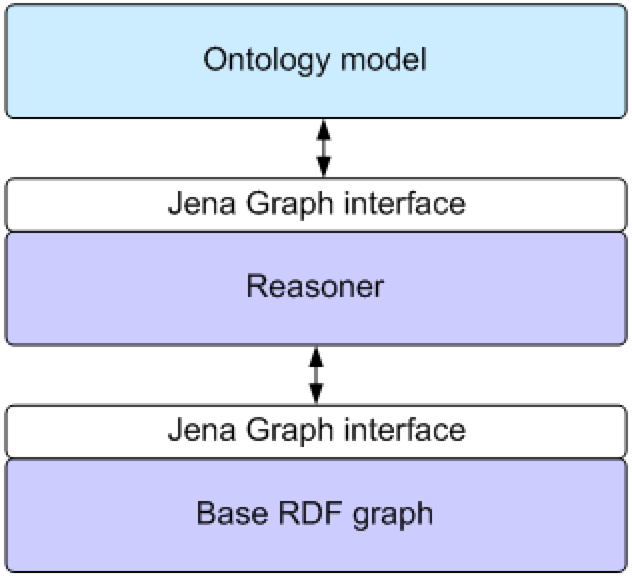
- Jena 1 tightly bound Java classes to the specific ontology languages,
- While Jena 2 ontology API is language-neutral.
  - Each language has its profile, which lists the permitted constructs and the URI’s of the classes and properties.
    - URI object property for
      - OWL →OWL:ObjectProperty
      - RDFS →null (as RDFS does not define object properties)
      - DAML →daml:ObjectProperty

Ontology Model

- Ontology Model allows to access to the statements in a collection of RDF data.
- The profile is bound to an ontology model, which extends this by adding supports to handle:
  - Classes (in a class hierarchy): OntClass has listSuperClasses() method
  - Properties (in a property hierarchy):
  - Individuals:
- Worth emphasizing:
  - No information is stored in the OntClass object itself.
  - When listSuperClass() method is called, the information is retrieved from the underlying RDF statements.

Ontology Model

- The statements that the ontology Java objects see:
  - The asserted statements in the underlying RDF graph
  - The statement inferred by the reasoner being used.
- Each module works with the Graph Interface which allows us:
  - To build models with no or different reasoners without changing ontology model
  - The RDF graph can be in memory store, persistent store without affecting the ontology model.



RDF-level polymorphism and Jena

- An ontology class: (relative)URI#DigitalCamera

```
<rdfs:Class rdf:ID=“DigitalCamera”></rdfs:Class>
```

- An OWL Restriction (subclass of rdfs:Class): #DigitalCamera

```
<rdfs:Class rdf:ID=“DigitalCamera”>
      <rdf:type owl:Restriction />  </rdfs:Class>
```

- #DigitalCamera is a class and a property

```
<rdfs:Class rdf:ID=“DigitalCamera”>
      <rdf:type owl:ObjectProperty />  </rdfs:Class>
```

How to change them in run-time? Jena 2
provide solution – as()

RDF-level polymorphism and Jena

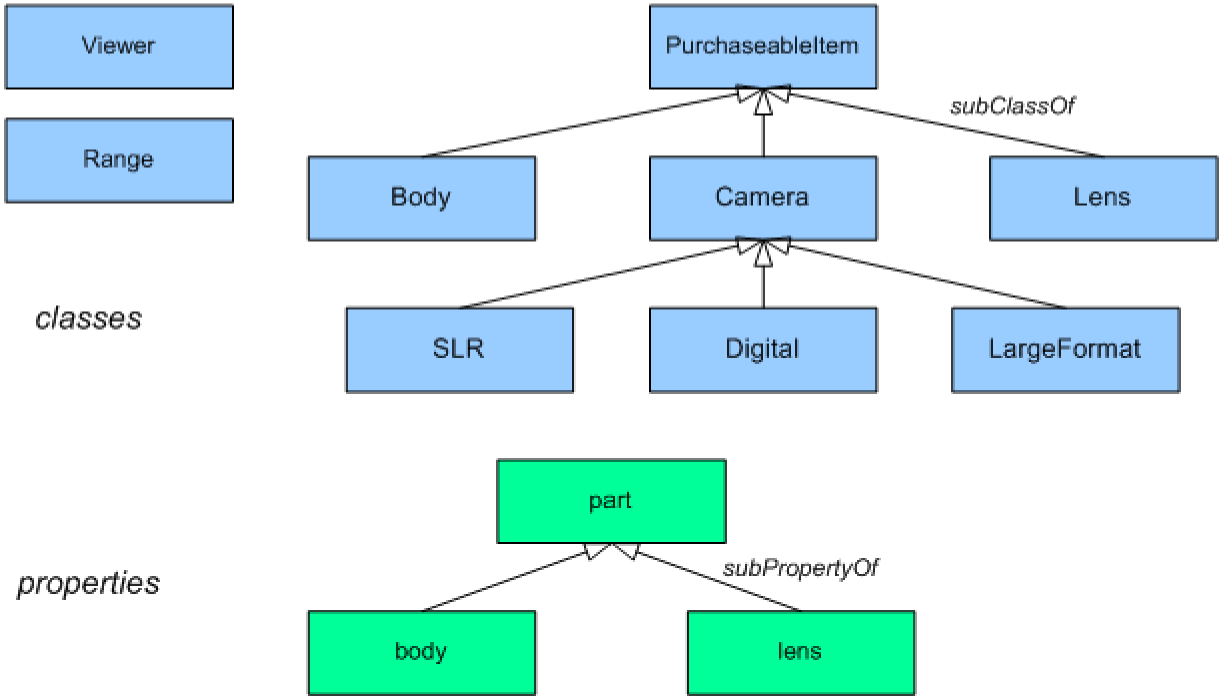
- Jena 2 accepts this basic characteristic of polymorphism at the RDF level by considering that the Java abstraction (OntClass, Restriction, DatatypeProperty, etc.) is just a view or facet of the resource.
- as() method

```
Resource r = myModel.getResource(myNS + “DigitalCamera” );
OntClass cls = (OntClass) r.as( OntClass.class );
Restriction rest = (Restriction) cls.as( Restriction.class );
```

This RDF-level polymorphism is used extensively in the Jena ontology API to allow maximum flexibility in handling ontology data

Example: the camera ontology

- This is an example drawn from a Roger Costello's camera ontology which deals with the domain of still-picture cameras.



Creating ontology models

- An ontology model is an extension of the Jena RDF model that provides extra capabilities for handling ontology data sources.
- Ontology models are created through the Jena ModelFactory.

```
OntModel m = ModelFactory.createOntologyModel();
```

- OntModelSpec is used to configure a ontology model, such as:
  - The ontology language in use, the reasoner, and the means of handling compound documents
  - OntModelSpec.OWL_MEM: a ontology model using the OWL FULL profile, an in-memory storage model, and no reasoner.
  - OntModelSpec.OWL_MEM_RDFS_INF: a ontology model same as above + using RDFS rule-based reasoner (which include entailments from subclass and sub-property hierarchies, and domain and range constraints, but not entailments from the disjointness of classes

Creating ontology models

- To create a model with a given specification,

```
OntModel m =
ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM, null );
```

- To create a custom model specification, try to copy the existing specification and then updates it as necessary:

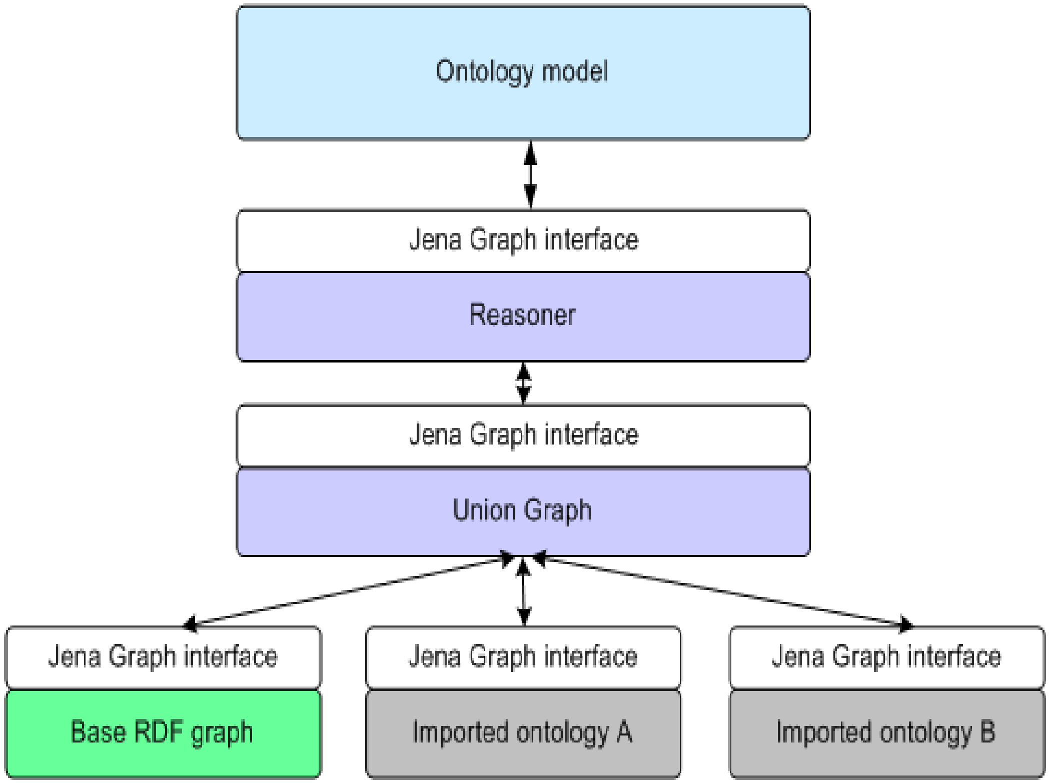
```
OntModelSpec s = new OntModelSpec( OntModelSpec.OWL_MEM );
s.setDocumentManager( myDocMgr );
OntModel m = ModelFactory.createOntologyModel( s, null );
```

URI of the ontology language

Ontology language	URI
RDFS	http://www.w3.org/2000/01/rdf-schema#
DAML+OIL	http://www.daml.org/2001/03/daml+oil#
OWL Full	http://www.w3.org/2002/07/owl#
OWL DL	http://www.w3.org/TR/owl-features/#term_OWLDL
OWL Lite	http://www.w3.org/TR/owl-features/#term_OWLLite

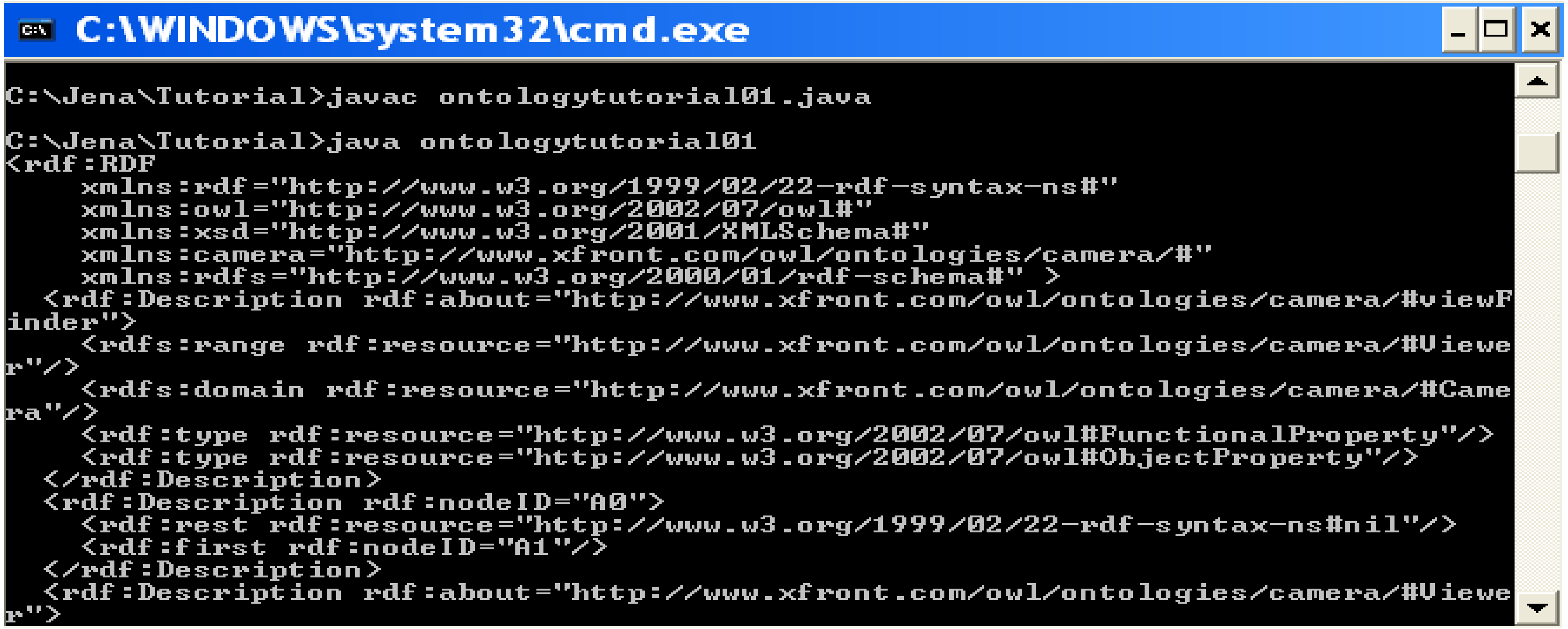
Handling ontology documents and imports

- Using the “read” method to load an ontology document into an ontology model
- The ontology “DocumentManager” assist to handle ontology import.
- It is important to hold each import as a separate graph structure so that we can know where a statement came from.



Ontology Tutorial 01

- Read ontology and output ontology
    - Read camera.owl and output it



```
C:\Jena\Tutorial>javac ontologytutorial01.java

C:\Jena\Tutorial>java ontologytutorial01
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:camera="http://www.xfront.com/owl/ontologies/camera/#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://www.xfront.com/owl/ontologies/camera/#viewF
inder">
    <rdfs:range rdf:resource="http://www.xfront.com/owl/ontologies/camera/#Viewe
r"/>
    <rdfs:domain rdf:resource="http://www.xfront.com/owl/ontologies/camera/#Came
ra"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A0">
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    <rdf:first rdf:nodeID="A1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.xfront.com/owl/ontologies/camera/#Viewe
r">
```

Ontology Tutorial 01

```java
import java.util.List;
import java.io.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;

public class ontologytutorial01 extends Object {
    static final String inputFileName = "camera.owl";

    public static void main (String args[]) {
          // Create an empty in-memory ontology model
          OntDocumentManager mgr = new OntDocumentManager();
          OntModelSpec s = new OntModelSpec( OntModelSpec.RDFS_MEM );
          s.setDocumentManager( mgr );
          OntModel m = ModelFactory.createOntologyModel( s, null );

          // use the FileManager to open the ontology from the filesystem
          InputStream in = FileManager.get().open(inputFileName);
          if (in == null) {
          throw new IllegalArgumentException( "File: " + inputFileName + " not found"); }

          // read the ontology file
          m.read( in, "" );

          // write it to standard out (RDF/XML)
          m.write(System.out); }
}
```

Ontologytutorial01.java

ontology document manager

- Each ontology model has an associated document manager that assists with the processing and handling of ontology documents.
- There is one global document manager that is used by default by ontology models.

```
OntDocumentManager mgr = new OntDocumentManager();
// set the mgr’s properties now ...
OntModelSpec s = new OntModelSpec( OntModelSpec.RDFS_MEM );
s.setDocumentManager( mgr );
OntModel m = ModelFactory.createOntologyModel( s, null );
```

Document manager policy

- The document manager has a large number of configurable options. There are two ways to setting them:
  - Using Java code to set them
  - Using document manager to load values for the parameters from ont-policy.rdf

```
<DocumentManagerPolicy>
  <!-- policy for controlling the document manager's behaviour -->
  <processImports
       rdf:datatype="&xsd;boolean">true</processImports>
  <cacheModels rdf:datatype="&xsd;Boolean">true</cacheModels>
</DocumentManagerPolicy>
```

ModelMaker

- The ModelMaker is a simple interface that allows different kinds of models (in memory, from file, in persistent database, etc.)
- For database, this may include passing the database user-name and password and other connection parameters.
- New model makers can be created via ModelFactory.
- The default specificaiton in OntModelSpec that begin MEM_ use in-memory model

Controlling imports processing

- To load an ontology without building the imports closure, call the method setProcessImports(false)
- To ignore certain URI’s when loading the imported documents, call the method addIgnoreImport(String uri)
- To solve the firewall problem of importing online ontologies, the ontology manager allows a local copy of such imported ontologies

```
<OntologySpec>
  <!-- local version of the RDFS vocabulary -->
  <publicURI rdf:resource="http://www.w3.org/2000/01/rdf-schema" />
  <altURL rdf:resource="file:vocabularies/rdf-schema.rdf" />
  <language rdf:resource="http://www.w3.org/2000/01/rdf-schema" />
  <prefix rdf:datatype="&xsd;string">rdfs</prefix>
</OntologySpec>
```

Specifying prefixes

- A model keeps a table of URI prefixes that can be used to render relative URIs
- The ontology model’s prefix table can be initialized by the document manger, to prevent such,
  - use the property useDeclaredNsPrefixes in the policy file (with value “false”), or
  - call the method setUseDeclaredPrefixes on the ontology object.

Caching models

- Suppose two ontologies, A and B both import ontology C. It would be nice not to have to read C twice when loading A and B.
- The document manager supports this use case by optionally caching C’s model.
- To turn model catching on or off,
  - use the policy property cacheModels, or
  - call the method setCacheModels(Boolean caching).
- The default is caching on.
- Model cache can be cleared at any time by calling clearCache().

```
OntModel m = ModelFactory.createOntologyModel();
OntDocumentManager dm = m.getDocumentManager();
dm.addAltEntry( "http://www.xfront.com/owl/ontologies/camera/",
                "file:" + JENA + "doc/user-manual/ontology/data/camera.owl" );
m.read( "http://www.xfront.com/owl/ontologies/camera/" );
```

The generic ontology type: OntResource

- All the classes in the ontology API that represent ontology values have OntResource as a common super-class.
- This makes OntResource a good place to put shared functionality for all such classes.
- The Java interface OntResource extends Jena’s RDF Resource interface.

Common Attributes of OntResource

Attribute	Meaning
versionInfo	A string documenting the version or history of this resource
comment	A general comment associated with this value
label	A human-readable label
seeAlso	Another web location to consult for more information about this resource
isDefinedBy	A specialisation of seeAlso that is intended to supply a definition of this resource
sameAs	Denotes another resource that this resource is equivalent to
differentFrom	Denotes another resource that is distinct from this resource (by definition)

Methods for Attributes of OntResource

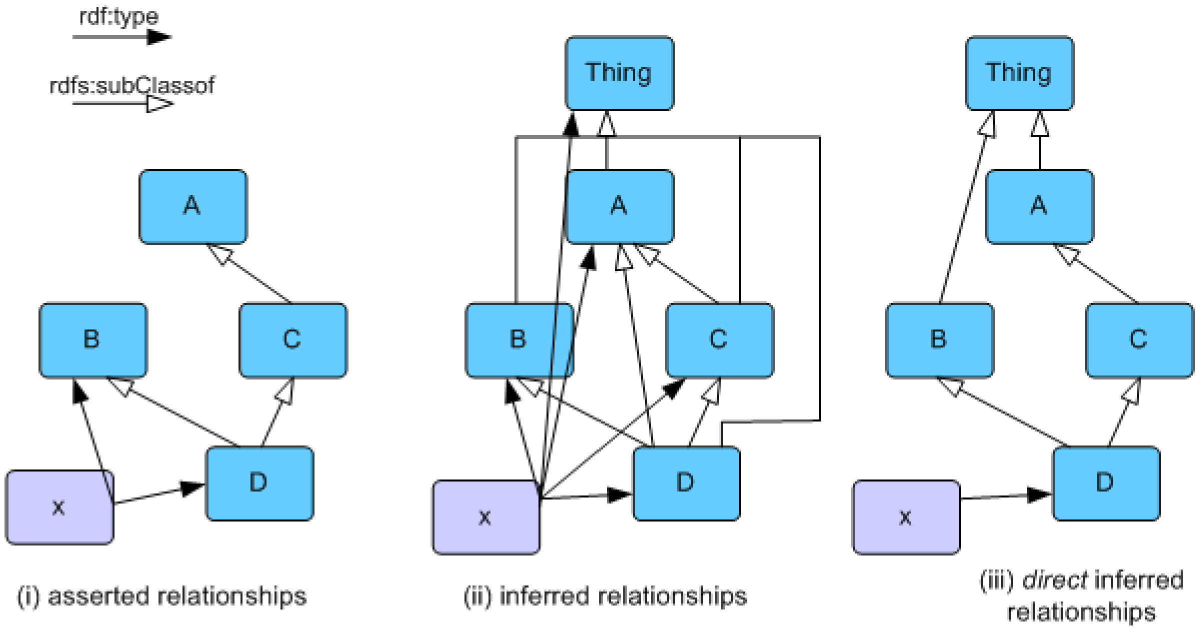
Method	Effect
add<property>	Add an additional value for the given property
set<property>	Remove any existing values for the property, then add the given value
list<property>	Return an iterator ranging over the values of the property
get<property>	Return the value for the given property, if the resource has one. If not, return null. If it has more than one value, an arbitrary selection is made.
has<property>	Return true if there is at least one value for the given property. Depending on the name of the property, this is sometimes is<property>
remove<property>	Removes a given value from the values of the property on this resource. Has no effect if the resource does not have that value.

OntResource other methods

- To find out how many values a resource has for a given property: getCardinality(Property p)
- Delete a resource: remove()
- Set the value of a given property: addPropertyValue(Property p, RDFNode value)
- Get the value of a given property: getPropertyValue(Property p)
- List the RDF types of a resource: listRDFTypes()
  - E.g., class B is the subclass of class A, resource x rdf:type is B,
    - Without reasoner, x’s RDF types is B
    - Reasoners with subclass hierarchy, x’s RDF types are B and A,
    - Complete reasoners, x’s RDF types are B, A, owl:Thing, rdf:Resource

rdf:type inference

- listRDFTypes()   // assumes not-direct
- listRDFTypes(Boolean direct) //if true, show only direct relationships



(i) asserted relationships

(ii) inferred relationships

(iii) *direct* inferred relationships

Handling ontology components: basic class expressions

- A simple class is represented in Jena as an OntClass object, which is the a facet of an RDF resource
- Get an ontology class

```
String camNS = "http://www.xfront.com/owl/ontologies/camera/#";
Resource r = m.getResource( camNS + "Camera" );
OntClass camera = (OntClass) r.as( OntClass.class );

Or
OntClass camera = m.getOntClass( camNS + "Camera" );
```

- Create a new ontology class

```
OntClass pinCamera = m.createClass( camNS + "PinholeCamera" );
```

- Create an anonymous class

```
OntClass anonClass = m.createClass();
```

Handling ontology components: basic class expressions

- The collection of methods for class are:
  - set, add, get, test, list and remove values
- Similar methods of class can be used to:
  - subClass, superClass, equivalentClass, disjointWith

Ontology Tutorial 02

- List the subclasses of class Camera

```
OntClass camera = m.getOntClass( camNS + "Camera" );
for (Iterator i = camera.listSubClasses(); i.hasNext(); ) {
  OntClass c = (OntClass) i.next();
  System.out.print( c.getLocalName() + " " );
}
```

Ontology Tutorial 02

```
import java.util.List;
import java.io.*;

import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;

public class ontologytutorial02 extends Object {

    static final String inputFileName = "camera.owl";
    static String camNS = "http://www.xfront.com/owl/ontologies/camera/#";

    public static void main (String args[]) {

        // Create an empty in-memory ontology model
        OntDocumentManager mgr = new OntDocumentManager();
        OntModelSpec s = new OntModelSpec( OntModelSpec.OWL_MEM );
        s.setDocumentManager( mgr );
        OntModel m = ModelFactory.createOntologyModel( s, null );
```

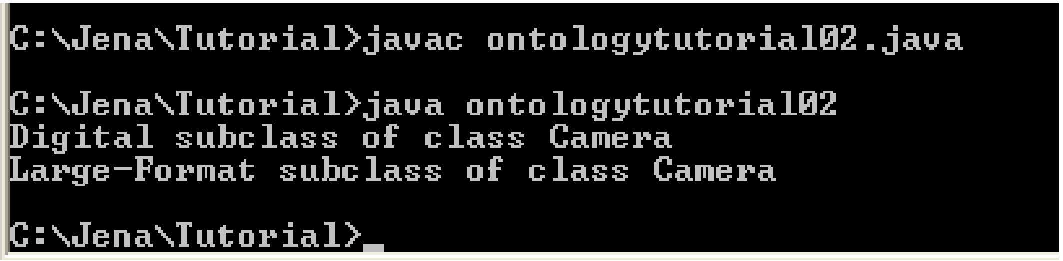
Ontology Tutorial 02

```
         // use the FileManager to open the ontology from the filesystem
         InputStream in = FileManager.get().open(inputFileName);
         if (in == null) {
       throw new IllegalArgumentException( "File: " + inputFileName + " not 
found");
        }

    // read the ontology file
    m.read( in, "" );

    // list the subclass of class Camera
    OntClass camera = m.getOntClass( camNS + "Camera" );
         for (ExtendedIterator i = camera.listSubClasses(); i.hasNext(); ) {
                 OntClass c = (OntClass) i.next();
                 System.out.println( c.getLocalName() + " subclass of class 
Camera " );
                          }
    }
}
```

Ontologytutorial02.java

Ontology Tutorial 02

- OntModelSpec s = new OntModelSpec(OntModelSpec.OWL_MEM);



```
C:\Jena\Tutorial>javac ontologytutorial02.java

C:\Jena\Tutorial>java ontologytutorial02
Digital subclass of class Camera
Large-Format subclass of class Camera

C:\Jena\Tutorial>_
```

Handling ontology components: properties

- The class for representing ontology properties in Jena is OntProperty.
- It can add, set, get, list, has and remove methods.

OntProperty

Attribute	Meaning
subProperty	A sub property of this property; i.e. a property which is declared to be a subPropertyOf this property. If p is a sub property of q, and we know that A p B is true, we can infer that A q B is also true.
superProperty	A super property of this property, i.e. a property that this property is a subPropertyOf
domain	Denotes the class or classes that form the domain of this property. Multiple domain values are interpreted as a conjunction. The domain denotes the class of value the property maps from.
range	Denotes the class or classes that form the range of this property. Multiple range values are interpreted as a conjunction. The range denotes the class of values the property maps to.
equivalentProperty	Denotes a property that is the same as this property.
inverse	Denotes a property that is the inverse of this property. Thus if q is the inverse of p, and we know that A q B, then we can infer that B p A.

Create property

- In camera ontology, the property body is a sub-property of part, and has domain Camera and range Body. We can create such property as:

```
OntModel newM = ModelFactory.createOntologyModel();
OntClass Camera = newM.createClass( camNS + "Camera" );
OntClass Body = newM.createClass( camNS + "Body" );

ObjectProperty part = newM.createObjectProperty( camNS + "part" );
ObjectProperty body = newM.createObjectProperty( camNS + "body" );

body.addSuperProperty( part );
body.addDomain( Camera );
body.addRange( Body );
```

More properties

- Use as() to change an object property facet to different kinds of property facet.
  - FunctionalProperty: for a given individual in the domain, the range value will be the same.
  - InverseFunctionalProperty: for a given range element, the domain value is unique
  - TransitiveProperty: if p is transitive, and we know A p B and also B p C, then A p C (e.g., hasBrother)
  - SymmetricProperty: if p is symmetric, and we know A p B, then B p A

```
public TransitiveProperty asTransitiveProperty();
public FunctionalProperty asFunctionalProperty();
public SymmetricProperty asSymmetricPropery();
public InverseFunctionalProperty asInverseFunctionalProperty();
```

Handling ontology components: more complex class expressions

- There are a number of additional class expressions that allow richer and more expressive descriptions of concepts, such as
  - Restriction class expression:
    - has value, all values from, some values from, cardinality, min cardinality, max cardinality,
  - Boolean expression:
    - and, or, not – intersection, union, and complement
  - List expression
    - Seq, Alt and Bag
  - Enumerated classes

Examples

```
OntClass c = m.createClass( Ns + "C" );
ObjectProperty p = m.createObjectProperty( Ns + "p" );

// use a null URI to create an anonymous restriction
AllValuesFromRestriction rst =
m.createAllValuesFromRestriction( null, p, c );
```

Restriction class

```
OntModel m = ModelFactory.createOntModel();
OntClass c0 = m.createClass( Ns + "c0" );
OntClass c1 = m.createClass( Ns + "c1" );
OntClass c2 = m.createClass( Ns + "c2" );

RDFList cs = m.createList( new RDFNode[] {c0, c1, c2} );
```

RDF List

Ontology Tutorial 03

- Create Ontology Camera
  - Here we show how to create the complex class “SLR”

```
<owl:Class rdf:ID="SLR">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Camera"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#viewfinder"/>
      <owl:hasValue rdf:resource="#ThroughTheLens"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Ontology Tutorial 03

```
import java.io.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;

public class CreateOntology extends Object {

    public static void main (String args[]) throws Exception{

        String camNS = "http://www.xfront.com/owl/ontologies/camera/#";
        String xmlbase = "http://www.xfront.com/owl/ontologies/camera/";

        // create an Ontology model
        OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

        Resource NAMESPACE = m.createResource( camNS );
        m.setNsPrefix( "camera", camNS);

        RDFWriter rdfw=m.getWriter("RDF/XML-ABBREV");
                    rdfw.setProperty("xmlbase", xmlbase);

          // class Camera
          OntClass Camera = m.createClass( camNS + "Camera" );
```

Ontology Tutorial 03

```
          // create the throughTheLens window instance
          OntClass Window = m.createClass( camNS + "Window" );
          Individual throughTheLens = m.createIndividual( camNS + "ThroughTheLens", Window );

          // create the viewfinder property
          ObjectProperty viewfinder = m.createObjectProperty( camNS + "viewfinder"    );

          // now the anonymous hasValue restriction
          HasValueRestriction viewThroughLens =
                      m.createHasValueRestriction( null, viewfinder, throughTheLens );

          // finally create the intersection class to define SLR
          IntersectionClass SLR = m.createIntersectionClass( camNS + "SLR",
                      m.createList( new RDFNode[] {viewThroughLens, Camera} ) );

        // now write the model in XML form to a file
        FileOutputStream camera_File = new FileOutputStream("C:/Jena/Tutorial/camera1.owl");
        //OutputStream out = (OutputStream) camera_File;

        m.write(camera_File, "RDF/XML-ABBREV", xmlbase);
    }
}
```

C:/Jena/Tutorial/CreateOntology.java

Ontology Tutorial 03

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:camera="http://www.xfront.com/owl/ontologies/camera/#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Class rdf:ID="Window"/>
  <owl:Class rdf:ID="SLR">
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:hasValue>
          <camera:Window rdf:ID="ThroughTheLens"/>
        </owl:hasValue>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="viewfinder"/>
        </owl:onProperty>
      </owl:Restriction>
      <owl:Class rdf:ID="Camera"/>
    </owl:intersectionOf>
  </owl:Class>
</rdf:RDF>
```

C:/Jena/Tutorial/camera1.owl

Instances (individuals)

- The method createIndividual(Resource cls) creates an anonymous individual belonging to the given class.

```
OntClass c = m.createClass( Ns + "C" );
Individual inst = m.createIndividual( Ns + "foo", c );
```

Ontology meta-data

- The metadata about the ontology itself is attached to an instance of class Ontology.
- It normally contains:
  - Version, Author, Comment, import

```
Ontology ont = m.getOntology( baseURI );
ont.addProperty( DC.creator, "John Smith" );
```

Ontology inference

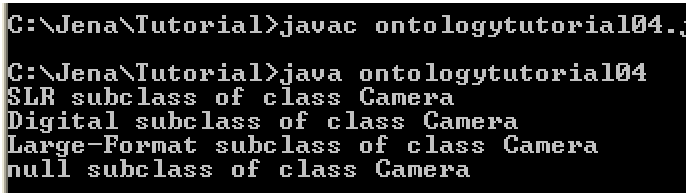
- Ontology inference by Jena is handled by Ontology Model.
- Jena framework also aligns with other reasoners, such as Pellet.

Inference and storage for ontology model

OntModelSpec	Language	Storage	Reasoner
OWL_MEM	OWL full	in-memory	none
OWL_MEM_TRANS_INF	OWL full	in-memory	transitive class-hierarchy inference
OWL_MEM_RULE_INF	OWL full	in-memory	rule-based reasoner with OWL rules
OWL_MEM_MICRO_RULE_INF	OWL full	in-memory	optimised rule-based reasoner with OWL rules
OWL_MEM_MINI_RULE_INF	OWL full	in-memory	rule-based reasoner with subset of OWL rules
OWL_DL_MEM	OWL DL	in-memory	none
OWL_DL_MEM_RDFS_INF	OWL DL	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_DL_MEM_TRANS_INF	OWL DL	in-memory	transitive class-hierarchy inference
OWL_DL_MEM_RULE_INF	OWL DL	in-memory	rule-based reasoner with OWL rules
OWL_LITE_MEM	OWL Lite	in-memory	none
OWL_LITE_MEM_TRANS_INF	OWL Lite	in-memory	transitive class-hierarchy inference
OWL_LITE_MEM_RDFS_INF	OWL Lite	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_LITE_MEM_RULES_INF	OWL Lite	in-memory	rule-based reasoner with OWL rules
RDFS_MEM	RDFS	in-memory	none
RDFS_MEM_TRANS_INF	RDFS	in-memory	transitive class-hierarchy inference
RDFS_MEM_RDFS_INF	RDFS	in-memory	rule reasoner with RDFS-level entailment-rules

Ontology Tutorial 04

- Test different inference models
- Based on ontology tutorial 02: List the subclass of Camera
  - OntModelSpec.OWL_MEM_RULE_INF
    - SLR, Digital, Large-Format, null
  - OntModelSpec.OWL_DL_MEM_RDFS_INF
    - Digital, Large-Format
  - OntModelSpec.RDFS_MEM_RDFS_INF
    - Digital, Large-Format



```
C:\Jena\Tutorial>javac ontologytutorial04.j

C:\Jena\Tutorial>java ontologytutorial04
SLR subclass of class Camera
Digital subclass of class Camera
Large-Format subclass of class Camera
null subclass of class Camera
```

Future versions of Jena will contain means of selectively ignoring such correct but unhelpful entailments

C:/Jena/Tutorial/ontologytutorial04.java

Jena schemagen

- Convert .rdf or .owl to java class.
- Command line:
  - java jena.schemagen -i <input> [-a <namespaceURI>] [-o <output file>] [-c <config uri>] [-e <encoding>] ...
  - Example: convert camera.owl to java class
    - C:\Jena\Tutorial>java jena.schemagen -i camera.owl –a http://www.xfront.com/owl/ontologies/camera/#

Jena schemagen

```
import java.io.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;

public class CreateOntology extends Object {

    public static void main (String args[]) throws Exception{

        String camNS = "http://localhost/owl/ontologies/camera/#";
        String xmlbase = camNS;

        // create an Ontology model
        OntModel m = ModelFactory.createOntologyModel(ProfileRegistry.OWL_LANG);

        Resource NAMESPACE = m.createResource( camNS );

        RDFWriter rdfw=m.getWriter("RDF/XML-ABBREV");
                    rdfw.setProperty("xmlbase", xmlbase);
                    rdfw.setProperty("relativeURIs", "");

          // class Camera
          OntClass Camera = m.createClass( camNS + "Camera" );
```

Jena schemagen

```
          // create the throughTheLens window instance
          OntClass Window = m.createClass( camNS + "Window" );
          Individual throughTheLens = m.createIndividual( camNS + "ThroughTheLens",
Window );

                    // create the viewfinder property
                    ObjectProperty viewfinder = m.createObjectProperty( camNS +
"viewfinder"    );

                    // now the anonymous hasValue restriction
                    HasValueRestriction viewThroughLens =
                      m.createHasValueRestriction( null, viewfinder, throughTheLens );

                    // finally create the intersection class to define SLR
                    IntersectionClass SLR = m.createIntersectionClass( camNS + "SLR",
                      m.createList( new RDFNode[] {viewThroughLens, Camera} ) );

        // now write the model in XML form to a file
        FileOutputStream camera_File = new FileOutputStream("C:/camera.owl");
        //OutputStream out = (OutputStream) camera_File;

        m.write(camera_File, "RDF/XML-ABBREV", xmlbase);

    }

}
```

CameraSchemagen.java

Working with persistent ontologies

- A common way to work with ontology data is to load the ontology and instances at run-time for a set of source documents.
- Limitation: it requires the documents to be parsed each time the application is run.
- Jena provides an implementation of the RDF model interface to store the ontology and instances persistently in a database.
- Suit for large size ontology

Importing and persisting models

- Persistent model
  - Java can persist its models on the filesystem (model.write() and model.read()), or in a relational database, the database engines currently supported are PostgreSQL, Oracle, and MySQL.
- Persistent models are created in the same way for any database system:
  - Load the JDBC driver, this enables Jena to communicate with the database instance.
  - Create a database connection, this creates a Java object for a database connection.
  - Create a ModelMaker for the database
  - Create a Model for existing or new data

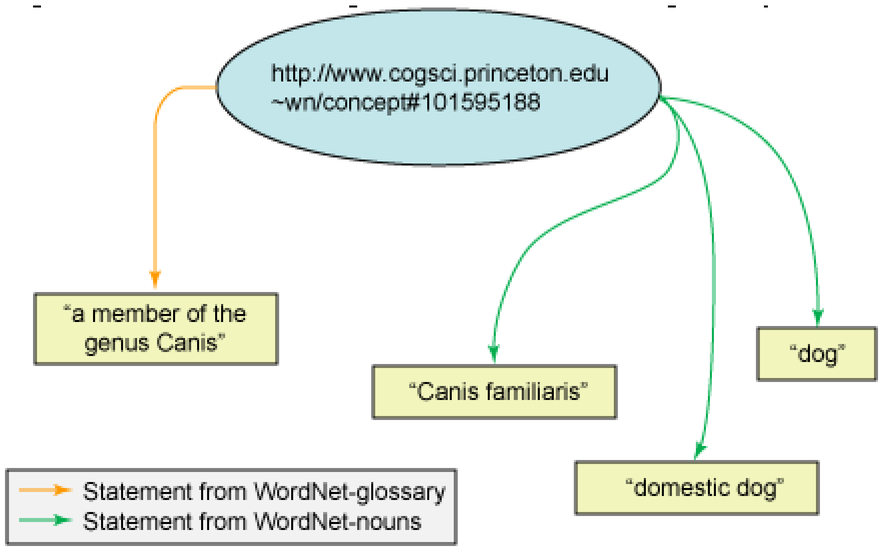
Connecting Jena with MySQL

- Steps in Java code

```
String className = "com.mysql.jdbc.Driver";         // path of driver class
Class.forName (className);                          // Load the Driver
String DB_URL =     "jdbc:mysql://localhost/jena";  // URL of database
String DB_USER =   "????";                          // database user id
String DB_PASSWD = "????";                          // database password
String DB =        "MySQL";                         // database type

// Create database connection
IDBConnection conn = new DBConnection ( DB_URL, DB_USER, DB_PASSWD, DB );
ModelMaker maker = ModelFactory.createModelRDBMaker(conn) ;

// create or open the default model
Model model = maker.createDefaultModel();

// Close the database connection
conn.close();
```

Ontology tutorial 05

- Import WordNet 1.6 RDF database to MySQL
  - Taking the form of several separate RDF documents, importing them into a single Jena model and merge their statements.


http://www.cogsci.princeton.edu
~wn/concept#101595188
"a member of the genus Canis"
"Canis familiaris"
"dog"
"domestic dog"
Statement from WordNet-glossary
Statement from WordNet-nouns

C:Jena/Tutorial/familytree/wordnet_glossary-20010201.rdf

C:Jena/Tutorial/familytree/wordnet_hyponyms-20010201.rdf

C:Jena/Tutorial/familytree/wordnet_nouns-20010201.rdf

Connecting Jena with MySQL

- Set classpath for mysql connector (JDBC driver)
  - set classpath=%classpath%;…\mysql-connector-java-5.1.6-bin.jar
  - javac -cp ;…\mysql-connector-java-5.1.6-bin.jar Yourclass.java
  - Or go to environmental variables to add the classpath.
- Go to MySQL
  - open and stop mysql server
    - C:\mysql\bin>net start mysql
    - C:\mysql\bin>net stop mysql
  - open and stop mysql client
    - C:\mysql\bin>mysql -uroot -p111
    - mysql>exit
  - Create an empty database
    - create database wordnet; in mysql client →mysql>create database wordnet;
    - Show existing mysql databases: mysql>show databases;

Loading Wordnet rdf files to MySQL

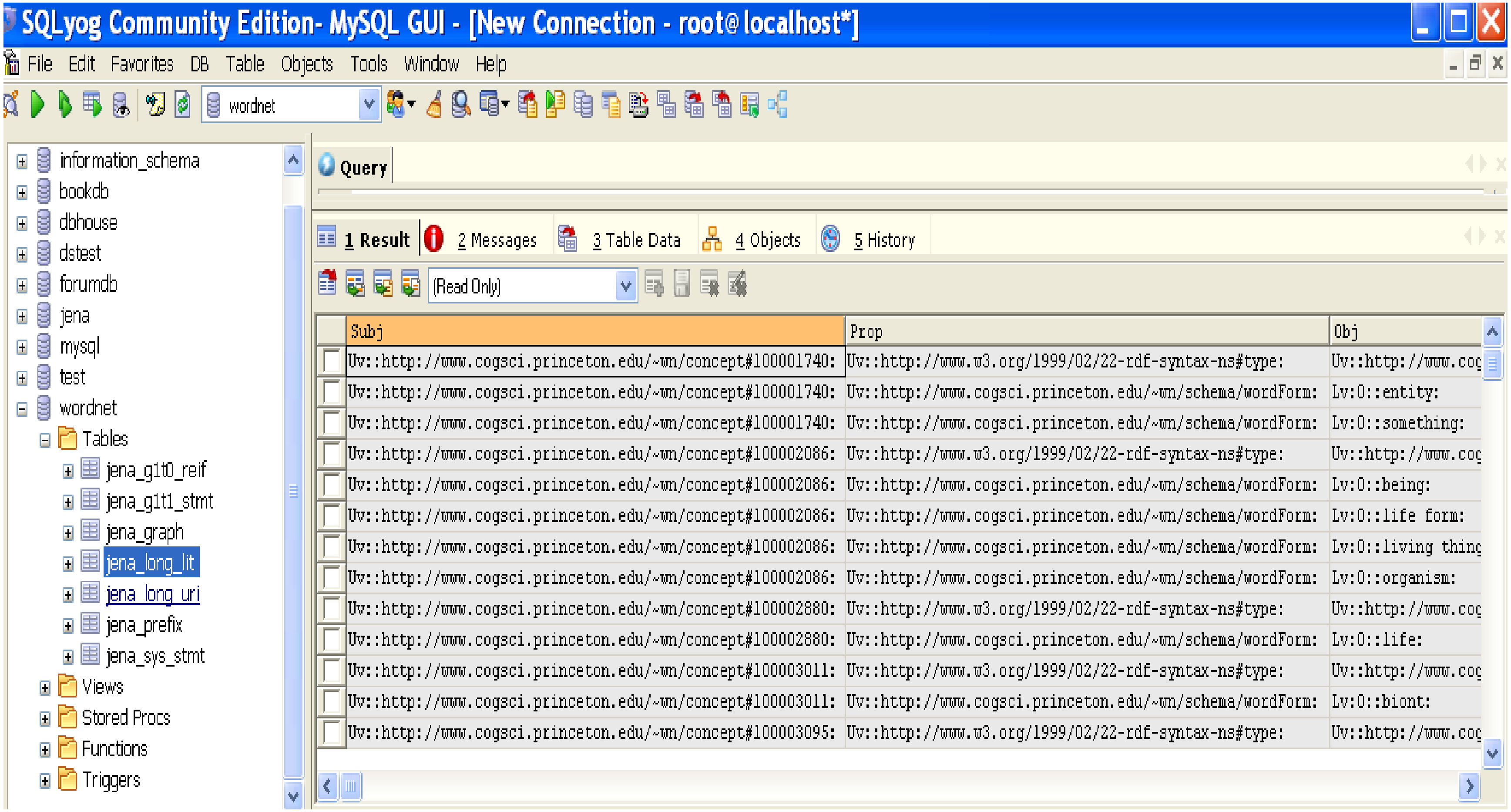
```
import java.io.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.rdf.model.*;

public class ImportWordnet {
  /** MySQL driver classname */
  private static final String mysqlDriver = "com.mysql.jdbc.Driver";
  /** URL of database to use */
  private static final String DB_URL = "jdbc:mysql://localhost/wordnet";
  private static final String DB_TYPE = "MySQL";
  /** User credentials */
  private static final String DB_USER = "root";
  private static final String DB_PASSWORD = "111";
  /** Name of the Jena model to create */
  private static final String MODEL_NAME = "wordnet";
  /** Locations of wordnet graphs to load */
  private static String WN_NOUNS    = "wordnet_nouns-20010201.rdf";
  private static String WN_GLOSSARY = "wordnet_glossary-20010201.rdf";
  private static String WN_HYPONYMS = "wordnet_hyponyms-20010201.rdf";

public static void main(String args[]) {
    try {
      // Instantiate database driver
      Class.forName(mysqlDriver);
    } catch (ClassNotFoundException e) {
      System.err.println("MySQL driver class not found");
      System.exit(-1);}
```

Loading Wordnet rdf files to MySQL

```
    // Get a connection to the db
    DBConnection connection = new DBConnection(DB_URL, DB_USER, DB_PASSWORD, DB_TYPE);
    // Get a ModelMaker for database-backend models
    ModelMaker maker = ModelFactory.createModelRDBMaker(connection);
    // Create a new model named "wordnet".
    Model wordnetModel = maker.createModel(MODEL_NAME,true);
    try {
      wordnetModel.begin();
      readFileIntoModel(WN_NOUNS, wordnetModel);
      readFileIntoModel(WN_GLOSSARY, wordnetModel);
      readFileIntoModel(WN_HYPONYMS, wordnetModel);
      // Commit the transaction
      wordnetModel.commit();
    } catch (FileNotFoundException e) {System.err.println(e.toString());
    } finally { try {
        // Close the database connection
        connection.close(); } catch (java.sql.SQLException e) {}  }
  }

private static void readFileIntoModel(String filename, Model model)
    throws FileNotFoundException {
      // Use the class loader to find the input file
      InputStream in = ImportWordnet.class.getClassLoader().getResourceAsStream(filename);
      if (in == null) {
        throw new FileNotFoundException("File not found on classpath: "+ filename);    }
      // Read the triples from the file into the model
      model.read(in,null);    }
}
```

C:/Jena/Tutorial/familytree/
ImportWordnet.java

Ontology tutorial 05



Subj	Prop	Obj
Uv::http://www.cogsci.princeton.edu/~wn/concept#100001740:	Uv::http://www.w3.org/1999/02/22-rdf-syntax-ns#type:	Uv::http://www.cog
Uv::http://www.cogsci.princeton.edu/~wn/concept#100001740:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::entity:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100001740:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::something:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002086:	Uv::http://www.w3.org/1999/02/22-rdf-syntax-ns#type:	Uv::http://www.cog
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002086:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::being:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002086:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::life form:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002086:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::living thing
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002086:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::organism:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002880:	Uv::http://www.w3.org/1999/02/22-rdf-syntax-ns#type:	Uv::http://www.cog
Uv::http://www.cogsci.princeton.edu/~wn/concept#100002880:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::life:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100003011:	Uv::http://www.w3.org/1999/02/22-rdf-syntax-ns#type:	Uv::http://www.cog
Uv::http://www.cogsci.princeton.edu/~wn/concept#100003011:	Uv::http://www.cogsci.princeton.edu/~wn/schema/wordForm:	Lv:0::biont:
Uv::http://www.cogsci.princeton.edu/~wn/concept#100003095:	Uv::http://www.w3.org/1999/02/22-rdf-syntax-ns#type:	Uv::http://www.cog

Jena 2 Database Interface

- The schema for storing RDF statements in a relational database is the triple store.
  - Each RDF statement is stored as a single row in a three column ‘statement’ table with subject, predicate and object as the column, and a fourth column to indicate if the object is a literal or a URI.
- Jena2 uses a denormalized triple store approach to achieve some efficiency:
  - A statement table, two types:
    - One for asserted statements
    - One for reified statements
  - a literal table
    - short literal is stored in statement table and long literal is stored in literal table
  - a resource table
    - long URIs are stored in the resource table

Tables

- Statement Tables
  - Asserted Statement Table (Jena_GiTj_Stmt)
    - Subj, Prop, Obj, GraphId (Identifier of graph (model) that contains the asserted statement)
  - Reified Statement Table (Jena_GiTj_Stmt)
    - Subj, Prop, Obj, GraphId, Stmt (identifier (URI) of reified statement), HasType (“true” if the graph contains the statement)
- System table (Jena_Sys_Stmt)
  - Subj, Prop, Obj, GraphId (always 0, representing the system graph)
- Long Literals Table (Jena_Long_Lit)
  - Id (ID for long literal, referenced from the statement tables), Head (first n characters of long literal), ChkSum (checksum of tail of long literal), Tail (remaining of long literal)

Tables

- Long Resources Table (Jena_Long_URI)
  - Id (identifier of long URI, referenced from the statement tables), Head (first n characters of long URI), ChkSum (Checksum of tail of long URI), Tail (remainder of long URI)
- Prefixes Table (Jena_Prefix)
  - Id (identifier for prefix, referenced from the statement tables), Head (first n characters of prefix), ChkSum (Checksum of tail of long prefix), Tail (remainder of long prefix)
- Graph Table (Jena_Graph)
  - Id (unique identifier for graph), Name (Graph name)

Jena Sparql

http://jena.sourceforge.net/ARQ/Tutorial/index.html

Jena SPARQL

- SPARQL queries RDF graphs (a set of triples):
  - RDF graphs – models (in Jena)
  - RDF triples – statements (in Jena)
- It is the triples that SPARQL cares, not the serialization.
  - The serialization is just a way to write the triples down
  - Here we use Turtle

vc-db-1.rdf in Turtle

```
@prefix vCard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://somewhere/MattJones/>  vCard:FN   "Matt Jones" .
<http://somewhere/MattJones/>  vCard:N    _:b0 .
_:b0  vCard:Family "Jones" .
_:b0  vCard:Given  "Matthew" .

<http://somewhere/RebeccaSmith/> vCard:FN    "Becky Smith" .
<http://somewhere/RebeccaSmith/> vCard:N     _:b1 .
_:b1 vCard:Family "Smith" .
_:b1 vCard:Given  "Rebecca" .

<http://somewhere/JohnSmith/>    vCard:FN    "John Smith" .
<http://somewhere/JohnSmith/>    vCard:N     _:b2 .
_:b2 vCard:Family "Smith" .
_:b2 vCard:Given  "John"  .

<http://somewhere/SarahJones/>   vCard:FN    "Sarah Jones" .
<http://somewhere/SarahJones/>   vCard:N     _:b3 .
_:b3 vCard:Family  "Jones" .
_:b3 vCard:Given   "Sarah" .
```

ARQ

- ARQ is a query engine for Jena that supports the SPARQL RDF Query language.
- ARQ Features:
  - Multiple query languages
    - SPARQL (.rq – file extension)
    - RDQL (.rdql)
    - ARQ, the engine’s own language (.arq)
  - Multiple query engines
    - General purpose engine
    - Remote access engines
    - Rewriter to SQL

Install ARQ

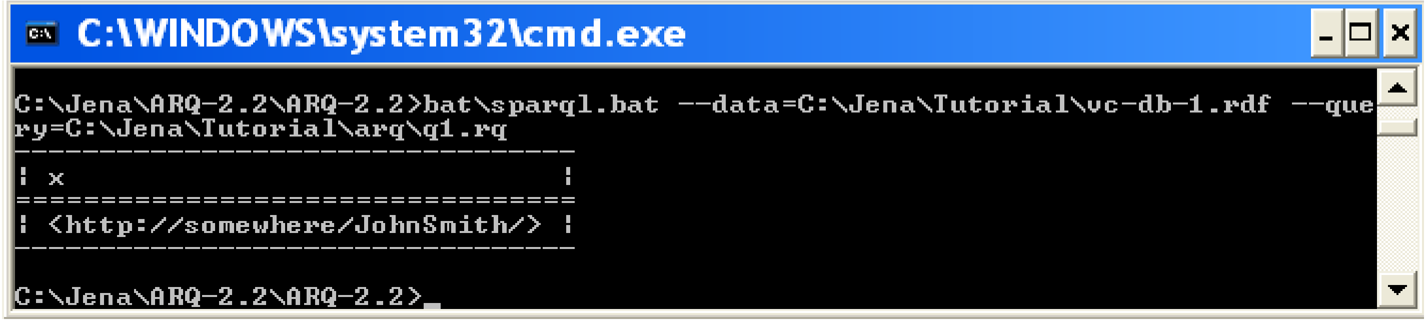
- Download ARQ from: http://jena.sourceforge.net/ARQ/download.html
- Unpack the zip: it unpacks into a directory (e.g. C:\Jena\ARQ-2.2\ARQ-2.2)
- Set classpath
  - Put every file in \lib on your classpath.
    - Using setcp.bat (e.g. the location of setcp.bat is C:\Jena)
    - C:\Jena>setcp.bat C:\Jena\ARQ-2.2\ARQ-2.2\lib
- Set ARQROOT environment variable to the path of the ARQ distribution
  - Go to C:\Jena\ARQ-2.2\ARQ-2.2 (ARQ location)
  - C:\Jena\ARQ-2.2\ARQ-2.2>set ARQROOT= C:\Jena\ARQ-2.2\ARQ-2.2

Query 1

- Data file: C:\Jena\Tutorial\vc-db-1.rdf
- Query file: C:\Jena\Tutorial\arq\q1.rq

```
SELECT ?x
WHERE
 { ?x <http://www.w3.org/2001/vcard-rdf/3.0#FN> "John Smith" }
```

- Execute query q1.rq
  - C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data= C:\Jena\Tutorial\vc-db-1.rdf --query= C:\Jena\Tutorial\arq\q1.rq

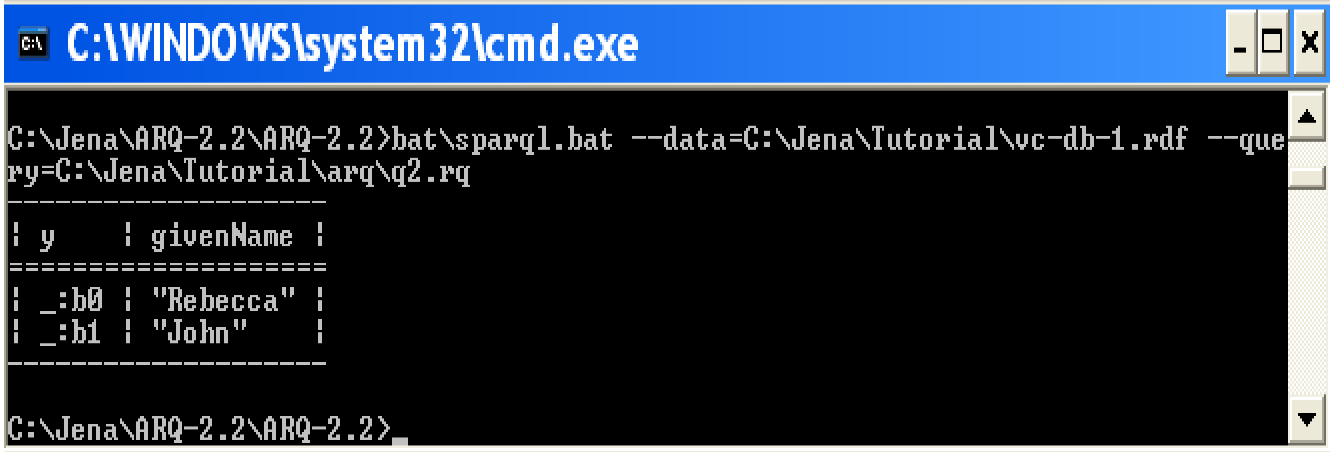


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=C:\Jena\Tutorial\vc-db-1.rdf --que
ry=C:\Jena\Tutorial\arq\q1.rq
---------------------------------
| x                             |
=================================
| <http://somewhere/JohnSmith/> |
---------------------------------

C:\Jena\ARQ-2.2\ARQ-2.2>_
```

Query 2

- Data file: C:\Jena\Tutorial\vc-db-1.rdf
- Query file: C:\Jena\Tutorial\arq\q2.rq

```
PREFIX vcard:      <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?y ?givenName
WHERE { ?y vcard:Family "Smith" .
         ?y vcard:Given  ?givenName . }
```

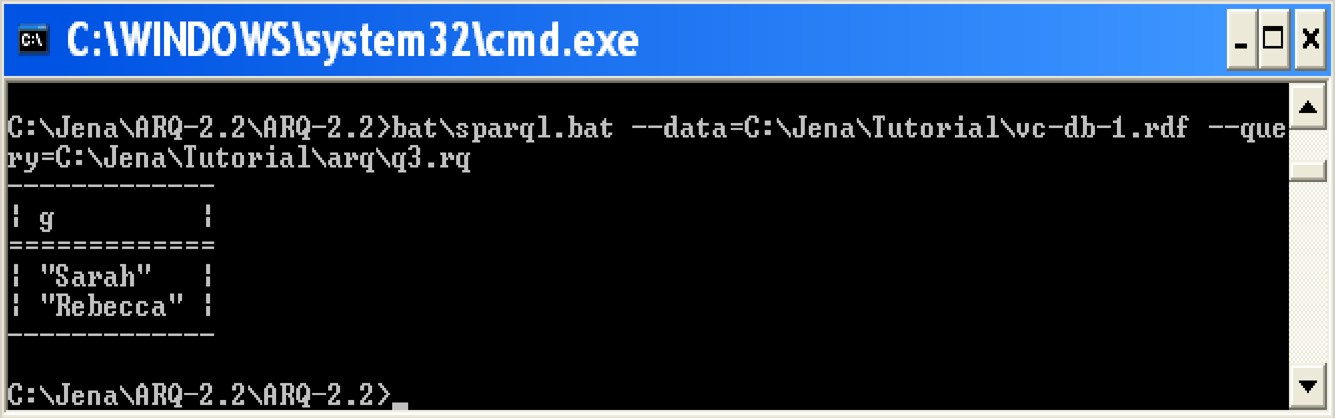


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=C:\Jena\Tutorial\vc-db-1.rdf --que
ry=C:\Jena\Tutorial\arq\q2.rq
---------------------
| y     | givenName |
=====================
| _:b0  | "Rebecca" |
| _:b1  | "John"    |
---------------------

C:\Jena\ARQ-2.2\ARQ-2.2>_
```

Query 3 - Filter

- Data file: C:\Jena\Tutorial\vc-db-1.rdf
- Query file: C:\Jena\Tutorial\arq\q3.rq

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?g
WHERE
{ ?y vcard:Given ?g .
  FILTER regex(?g, "r", "i") }
```

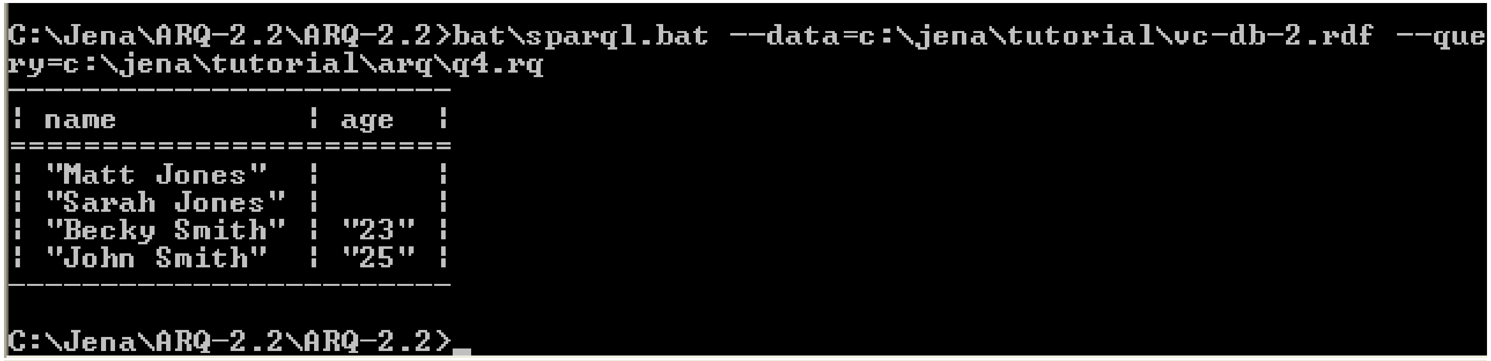


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=C:\Jena\Tutorial\vc-db-1.rdf --que
ry=C:\Jena\Tutorial\arq\q3.rq
-------------
| g         |
=============
| "Sarah"   |
| "Rebecca" |
-------------

C:\Jena\ARQ-2.2\ARQ-2.2>_
```

Query 4 - OPTIONAL

- Data file: C:\Jena\Tutorial\vc-db-2.rdf
- Query file: C:\Jena\Tutorial\arq\q4.rq

```
PREFIX info:    <http://somewhere/peopleInfo#>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{    ?person vcard:FN  ?name .
    OPTIONAL { ?person info:age ?age }  }
```

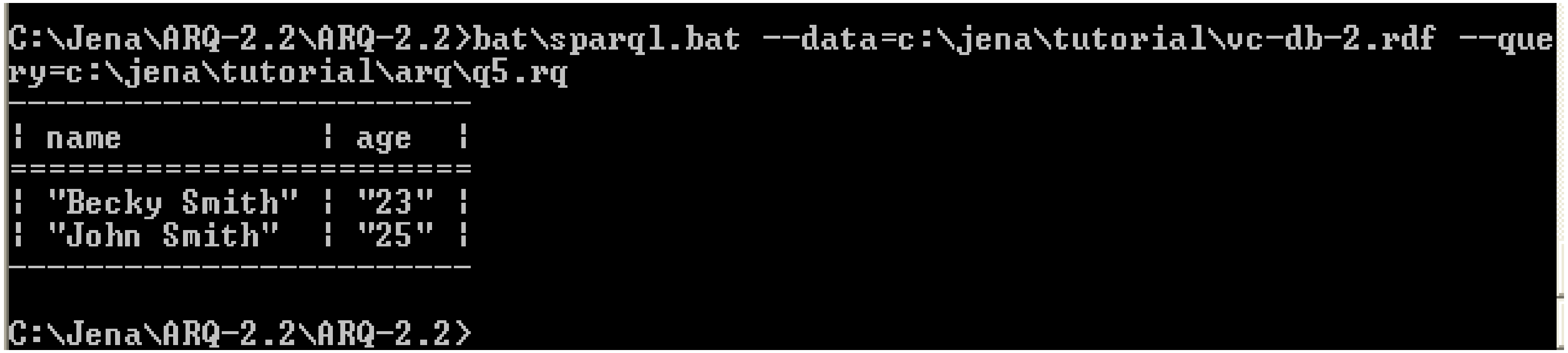


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=c:\jena\tutorial\vc-db-2.rdf --que
ry=c:\jena\tutorial\arq\q4.rq
------------------------
| name           | age  |
========================
| "Matt Jones"   |      |
| "Sarah Jones"  |      |
| "Becky Smith"  | "23" |
| "John Smith"   | "25" |
------------------------

C:\Jena\ARQ-2.2\ARQ-2.2>_
```

Query 5

- Data file: C:\Jena\Tutorial\vc-db-2.rdf
- Query file: C:\Jena\Tutorial\arq\q5.rq

```
PREFIX info:    <http://somewhere/peopleInfo#>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{    ?person vcard:FN  ?name .
     ?person info:age ?age .   }
```

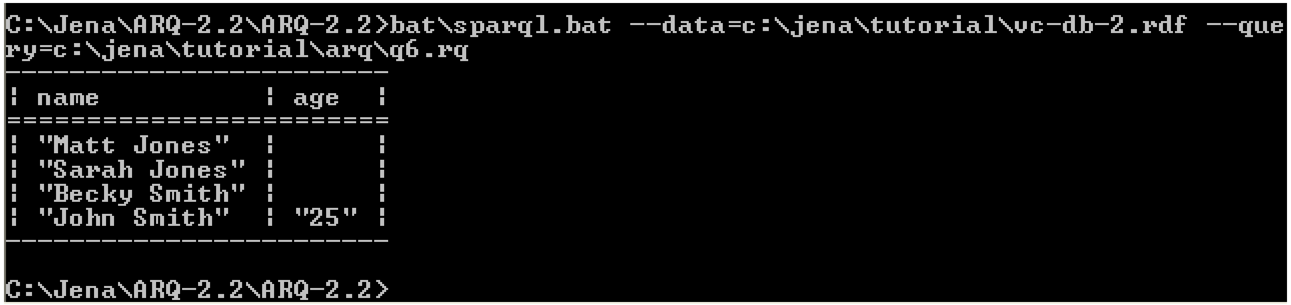


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=c:\jena\tutorial\vc-db-2.rdf --que
ry=c:\jena\tutorial\arq\q5.rq
------------------------
¦ name          ¦ age  ¦
========================
¦ "Becky Smith" ¦ "23" ¦
¦ "John Smith"  ¦ "25" ¦
------------------------

C:\Jena\ARQ-2.2\ARQ-2.2>
```

Query 6 – Optional and Filter

- Data file: C:\Jena\Tutorial\vc-db-2.rdf
- Query file: C:\Jena\Tutorial\arq\q6.rq

```
PREFIX info:        <http://somewhere/peopleInfo#>
PREFIX vcard:      <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
    ?person vcard:FN  ?name .
    OPTIONAL { ?person info:age ?age . FILTER ( ?age > “24” ) }
}
```



```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=c:\jena\tutorial\vc-db-2.rdf --que
ry=c:\jena\tutorial\arq\q6.rq
------------------------
| name          | age  |
========================
| "Matt Jones"  |      |
| "Sarah Jones" |      |
| "Becky Smith" |      |
| "John Smith"  | "25" |
------------------------

C:\Jena\ARQ-2.2\ARQ-2.2>
```

Query 7 - Union

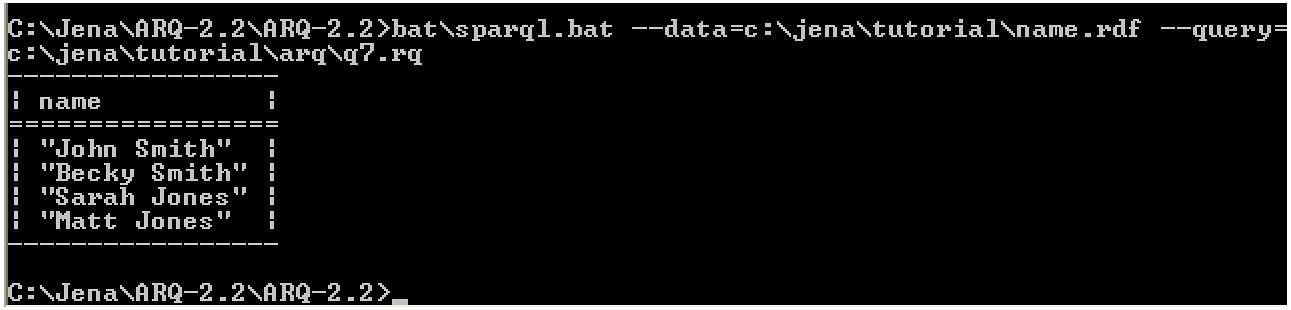
- Data: C:\Jena\Tutorial\name.rdf

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:a foaf:name   "Matt Jones" .

_:b foaf:name   "Sarah Jones" .

_:c vcard:FN    "Becky Smith" .

_:d vcard:FN    "John Smith" .
```

Query 7 - Union

- Data file: C:\Jena\Tutorial\name.rdf
- Query file: C:\Jena\Tutorial\arq\q7.rq

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name
WHERE
{
   { [] foaf:name ?name } UNION { [] vCard:FN ?name }
}
```



```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=c:\jena\tutorial\name.rdf --query=
c:\jena\tutorial\arq\q7.rq
-----------------
| name          |
=================
| "John Smith"  |
| "Becky Smith" |
| "Sarah Jones" |
| "Matt Jones"  |
-----------------

C:\Jena\ARQ-2.2\ARQ-2.2>_
```

Query 8 – Named Graphs

Default graph:
ds-dft.ttl

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<ds-ng-1.ttl> dc:date "2005-07-14T03:18:56+0100"^^xsd:dateTime .
<ds-ng-2.ttl> dc:date "2005-09-22T05:53:05+0100"^^xsd:dateTime .
```

Named graph:
ds-ng-1.ttl

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .

[] dc:title "Harry Potter and the Philospher's Stone" .
[] dc:title "Harry Potter and the Chamber of Secrets" .
```

Named graph:
ds-ng-2.ttl

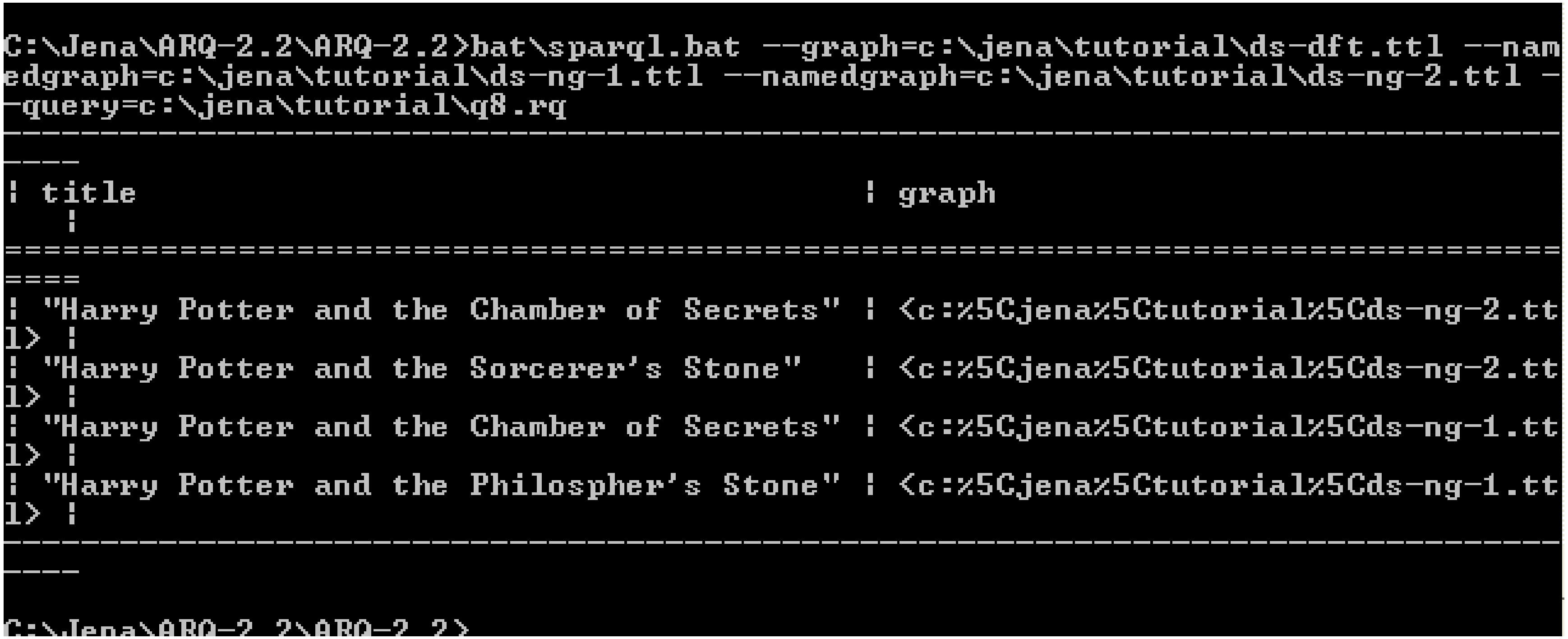
```
@prefix dc: <http://purl.org/dc/elements/1.1/> .

[] dc:title "Harry Potter and the Sorcerer's Stone" .
[] dc:title "Harry Potter and the Chamber of Secrets" .
```

Query 8 – Named Graphs

- Data file: C:\Jena\Tutorial\
- Query file: C:\Jena\Tutorial\arq\q8.rq

```
PREFIX  xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX  dc:     <http://purl.org/dc/elements/1.1/>
PREFIX :    <.>

SELECT ?title ?graph
FROM                      <ds-dft.ttl>
FROM NAMED                <ds-ng-1.ttl>
FROM NAMED                <ds-ng-2.ttl>
WHERE {
  GRAPH ?graph {
         ?x dc:title ?title . }
}
```

Query 8 – Named Graphs

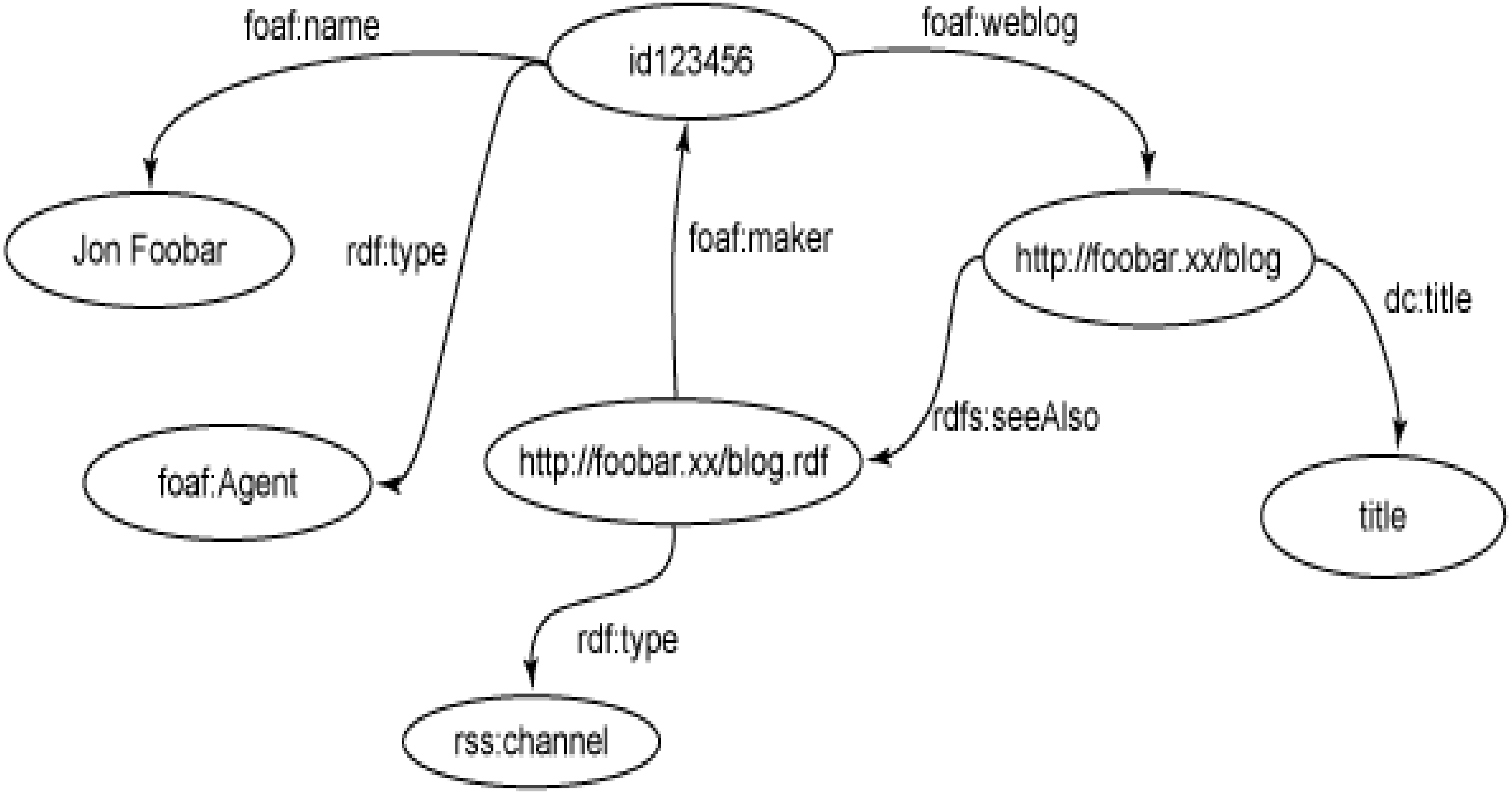


```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --graph=c:\jena\tutorial\ds-dft.ttl --nam
edgraph=c:\jena\tutorial\ds-ng-1.ttl --namedgraph=c:\jena\tutorial\ds-ng-2.ttl -
-query=c:\jena\tutorial\q8.rq
--------------------------------------------------------------------------------
----
¦ title                                     ¦ graph
   ¦
================================================================================
====
¦ "Harry Potter and the Chamber of Secrets" ¦ <c:%5Cjena%5Ctutorial%5Cds-ng-2.tt
l> ¦
¦ "Harry Potter and the Sorcerer's Stone"   ¦ <c:%5Cjena%5Ctutorial%5Cds-ng-2.tt
l> ¦
¦ "Harry Potter and the Chamber of Secrets" ¦ <c:%5Cjena%5Ctutorial%5Cds-ng-1.tt
l> ¦
¦ "Harry Potter and the Philospher's Stone" ¦ <c:%5Cjena%5Ctutorial%5Cds-ng-1.tt
l> ¦
--------------------------------------------------------------------------------
----

C:\Jena\ARQ-2.2\ARQ-2.2>
```

Executing SPARQL queries via Jena API

- SPARQL queries are created and executed with Jena via classes in the com.hp.hpl.jena.query package.
- Using QueryFactory is the simplest approach.
  - Create() methods are used to read a textual query from a file or from a String.
  - Create() returns a query object with a parsed query
- Create an instance of QueryExecution to perform a different type of query
  - Call QueryExecutionFactory.create(query, model)
  - Because the data for the query is provided programmatically, the query does not need a FROM clause.
- ResultSet allows you to iterate over QuerySolution providing access to each bound variable’s value.

Bloggers.rdf

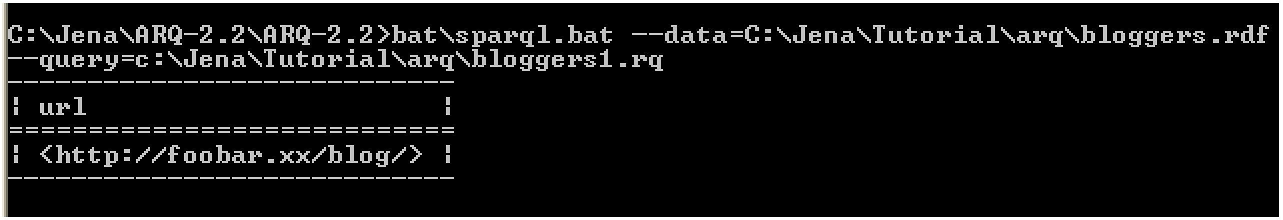


Query bloggers

- Data file: C:\Jena\Tutorial\arq\bloggers.rdf
- Query file: C:\Jena\Tutorial\arq\bloggers1.rq

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?url
FROM     <bloggers.rdf>
WHERE  {
    ?contributor foaf:name "Jon Foobar" .
    ?contributor foaf:weblog ?url .
}
```

- Execute query bloggers1.rq
  - C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data= C:\Jena\Tutorial\arq\bloggers.rdf - -query= C:\Jena\Tutorial\arq\bloggers1.rq



```
C:\Jena\ARQ-2.2\ARQ-2.2>bat\sparql.bat --data=C:\Jena\Tutorial\arq\bloggers.rdf
--query=c:\Jena\Tutorial\arq\bloggers1.rq
-----------------------------
| url                       |
=============================
| <http://foobar.xx/blog/>  |
-----------------------------
```

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.query.* ;
import com.hp.hpl.jena.query.ARQ;
import com.hp.hpl.jena.sparql.*;

import java.io.*;

public class Bloggers extends Object {

  static final String inputFileName = "bloggers.rdf";

  public static void main (String args[]) {

  // Create an empty in-memory model
  Model model = ModelFactory.createDefaultModel();

  // use the FileManager to open the bloggers RDF graph from the filesystem
  InputStream in = FileManager.get().open(inputFileName);
   if (in == null) {
    throw new IllegalArgumentException( "File: " + inputFileName + " not found");
        }

  // read the RDF/XML file
  model.read( in, "" );
```

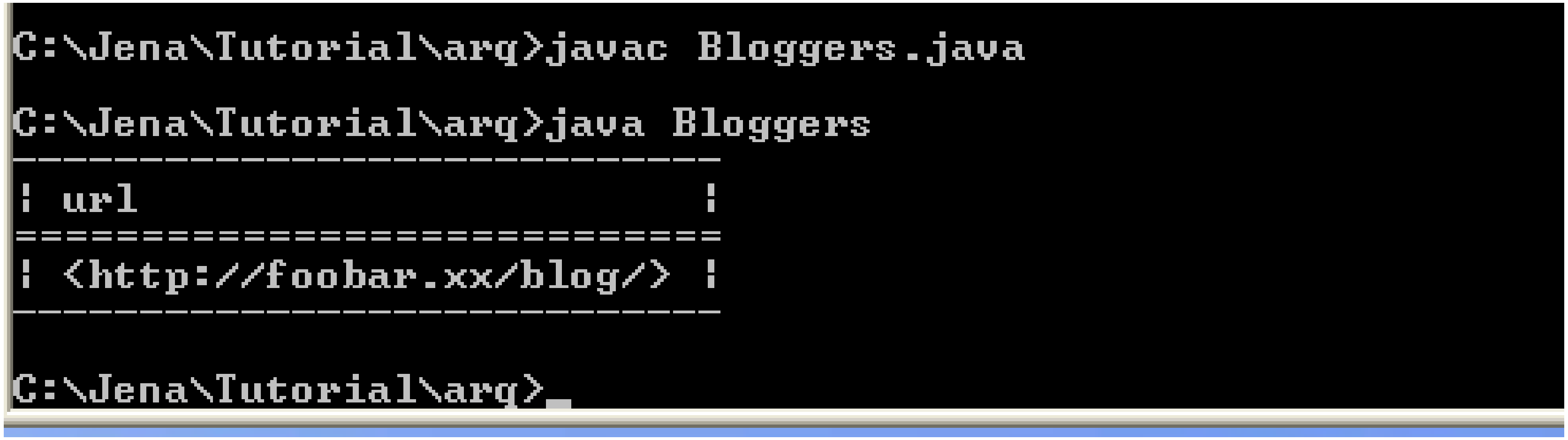
Bloggers.java

```
// Create a new query
String queryString =
        "PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
        "SELECT ?url " +
        "WHERE {" +
        "      ?contributor foaf:name \"Jon Foobar\" . " +
        "      ?contributor foaf:weblog ?url . " +
        "      }";

Query query = QueryFactory.create(queryString);

// Execute the query and obtain results
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();

// Output query results
ResultSetFormatter.out(System.out, results, query);

// Important - free up resources used running the query
qe.close();
}

}
```

Bloggers.java

Executing SPARQL queries via Jena API

- Store bloggers.java in C:\Jena\Tutorial\arq
- Compile
- Run



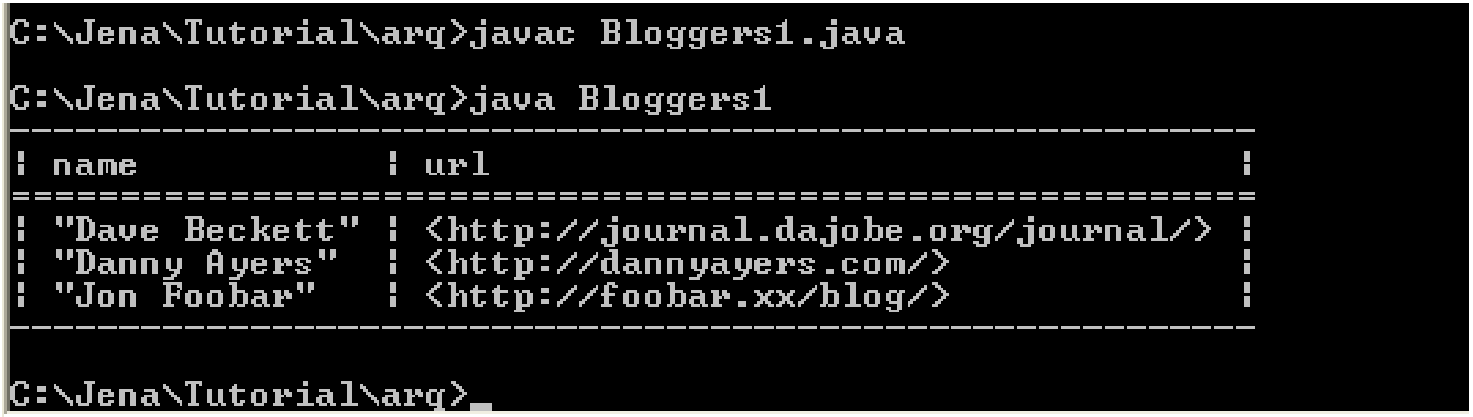
```
C:\Jena\Tutorial\arq>javac Bloggers.java

C:\Jena\Tutorial\arq>java Bloggers
-----------------------------
| url                       |
=============================
| <http://foobar.xx/blog/> |
-----------------------------

C:\Jena\Tutorial\arq>_
```

Executing SPARQL queries via Jena API

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?url
FROM      <bloggers.rdf>
WHERE  {
    ?x foaf:name ?name .
    ?x foaf:weblog ?url .
}
```

Bloggers1.java

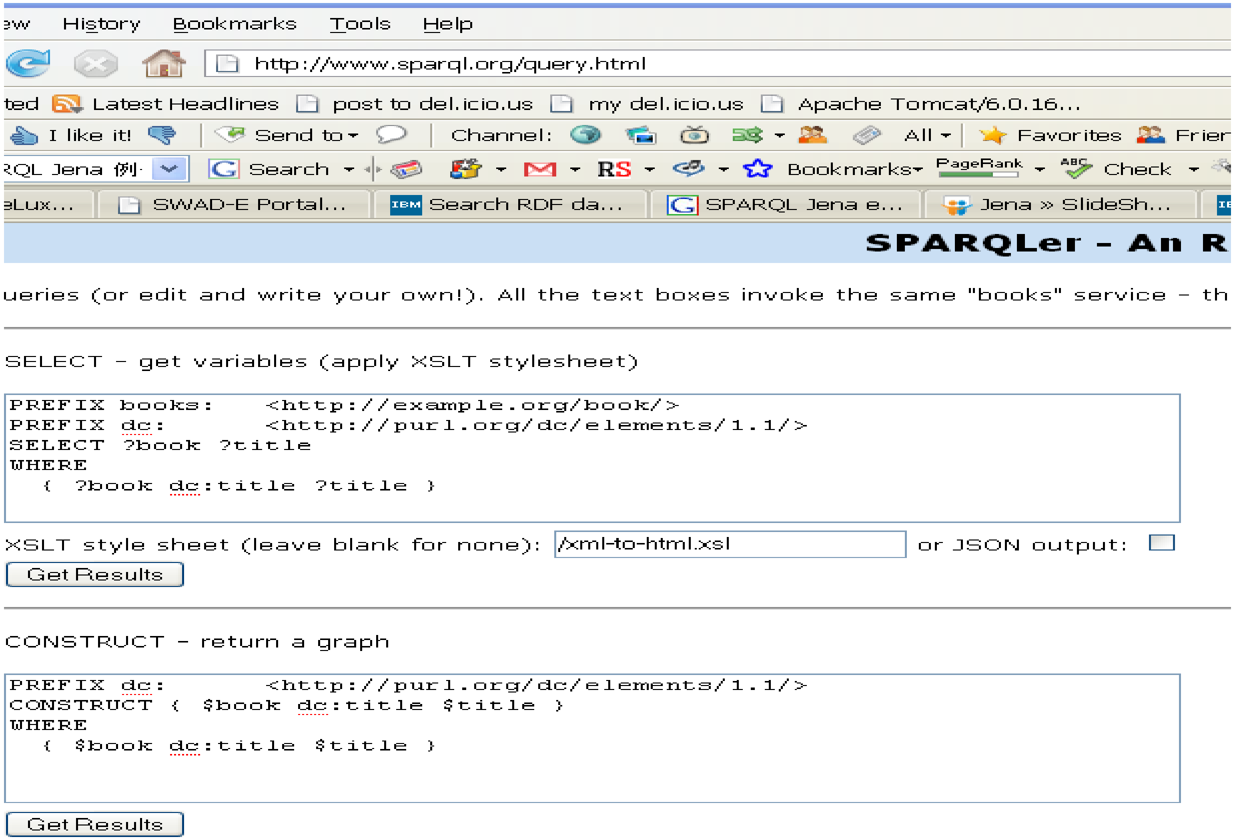
- Store bloggers1.java in C:\Jena\Tutorial\arq
- Compile and run



```
C:\Jena\Tutorial\arq>javac Bloggers1.java

C:\Jena\Tutorial\arq>java Bloggers1
-----------------------------------------------------------
¦ name              ¦ url                                  ¦
===========================================================
¦ "Dave Beckett"  ¦ <http://journal.dajobe.org/journal/> ¦
¦ "Danny Ayers"   ¦ <http://dannyayers.com/>            ¦
¦ "Jon Foobar"    ¦ <http://foobar.xx/blog/>            ¦
-----------------------------------------------------------

C:\Jena\Tutorial\arq>_
```

SPARQLer – An RDF Query Demo

- http://www.sparql.org/query.html


ew History Bookmarks Tools Help
http://www.sparql.org/query.html
ted Latest Headlines post to del.icio.us my del.icio.us Apache Tomcat/6.0.16...
I like it! Send to Channel: All Favorites Frier
RQL Jena 例 Search RS Bookmarks PageRank Check
eLux... SWAD-E Portal... Search RDF da... SPARQL Jena e... Jena » SlideSh...
SPARQLer - An R

ueries (or edit and write your own!). All the text boxes invoke the same "books" service - th

SELECT - get variables (apply XSLT stylesheet)

```
PREFIX books:   <http://example.org/book/>
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
SELECT ?book ?title
WHERE
  { ?book dc:title ?title }
```

XSLT style sheet (leave blank for none): /xml-to-html.xsl or JSON output: □

Get Results

CONSTRUCT - return a graph

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
CONSTRUCT { $book dc:title $title }
WHERE
  { $book dc:title $title }
```

Get Results

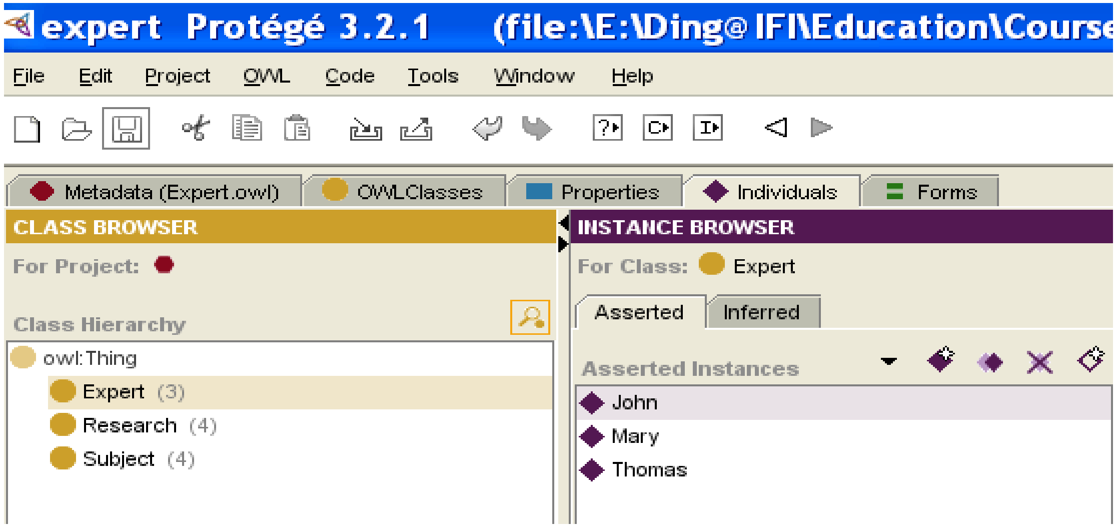
Jena Examples

Expert

FamilyTree

Expert

- Using Protege to build up expert.owl:
  - Three classes: Expert, Research, Subject
  - Properties: hasResearch: expert can have many researches; research is associated with many subjects
  - Create some instances.
  - Output this ontology as expert.owl (for example: C:\Jena\Tutorial\expert\expert.owl)



ontologyDB

- Introduce some methods to handle store, read ontology in/from persistent database (here takes MySQL as example):
  - connectDB
  - createDBModelFromFile
  - getModelFromDB
  - getModelSpec

ontologyDB

```
import java.util.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;

public class ontologyDB {

/* link database */
public static IDBConnection connectDB(String DB_URL, String DB_USER, String
DB_PASSWD, String DB_NAME) {
        return new DBConnection(DB_URL, DB_USER, DB_PASSWD, DB_NAME);}

/* Read ontology from filesystem and store it into database */
public static OntModel createDBModelFromFile(IDBConnection con, String name,
String filePath) {
        ModelMaker maker = ModelFactory.createModelRDBMaker(con);
        Model base = maker.createModel(name);
        OntModel newmodel =
        ModelFactory.createOntologyModel( getModelSpec(maker), base );
        newmodel.read(filePath);
        return newmodel; }
```

ontologyDB

```
/* Get ontology from database */
public static OntModel getModelFromDB(IDBConnection con, String name) {
         ModelMaker maker = ModelFactory.createModelRDBMaker(con);
         Model base = maker.getModel(name);
         OntModel newmodel =
         ModelFactory.createOntologyModel( getModelSpec(maker), base);
         return newmodel; }

public static OntModelSpec getModelSpec(ModelMaker maker) {
         OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM);
         spec.setImportModelMaker(maker);
         return spec;}
}
```

C:\Jena\Tutorial\expert\ontologyDB.java

Main.java

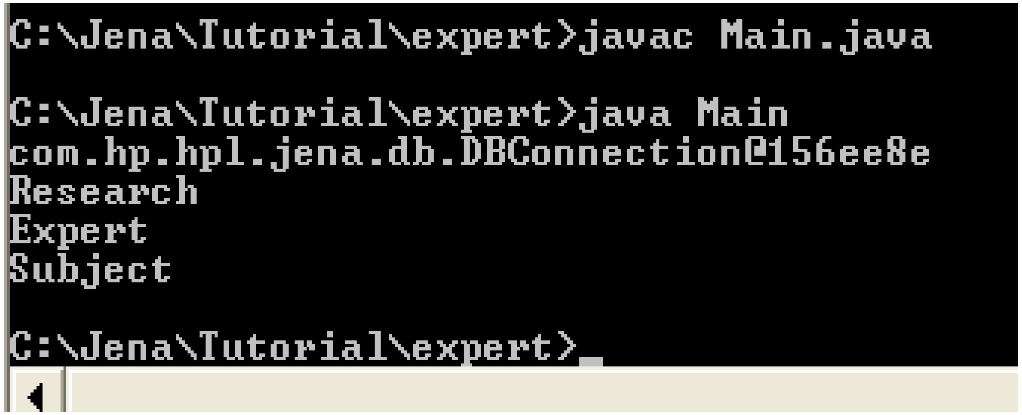
- Store expert.owl into MySQL database,
- Read expert.owl from MySQL database,
- List the classes of expert.owl

C:\Jena\Tutorial\expert\Main.java

```
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.ModelMaker;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.db.*;

public class Main  {
    public static final String DB_URL = "jdbc:mysql://localhost/expert";
    public static final String DB_USER = "root";
    public static final String DB_PASSWD = "111";
    public static final String DB = "MySQL";
    public static final String DB_DRIVER = "com.mysql.jdbc.Driver";

  public static void main (String args[]) {
    try {
        Class.forName("com.mysql.jdbc.Driver");
         } catch (ClassNotFoundException e) {
                 e.printStackTrace(); }
    String filePath = "file:C://Jena//Tutorial//expert//expert.owl";
    IDBConnection con = ontologyDB.connectDB(DB_URL,DB_USER, DB_PASSWD, DB);
    System.out.println(con);

    ontologyDB.createDBModelFromFile(con, "expert", filePath);
    OntModel model = ontologyDB.getModelFromDB(con, "expert");

   for (ExtendedIterator i = model.listClasses(); i.hasNext();) {
         OntClass c = (OntClass) i.next();
         System.out.println(c.getLocalName());     }        }
}
```

Expert

- Create expert database in MySQL
  - Mysql>create database expert;
- Parse ontologyDB.java and Main.java
- Run Main.java



```
C:\Jena\Tutorial\expert>javac Main.java

C:\Jena\Tutorial\expert>java Main
com.hp.hpl.jena.db.DBConnection@156ee8e
Research
Expert
Subject

C:\Jena\Tutorial\expert>_
```

Notes:

Make sure all classes are owl classes because OWL_MEM:

```
<owl:Class rdf:ID="Research"/>
  <owl:Class rdf:ID="Expert"/>
  <owl:Class rdf:ID="Subject"/>
```

Expert - Class

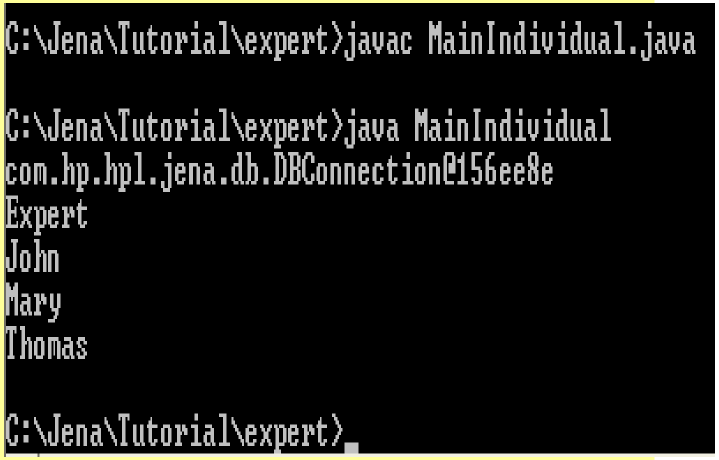
- Main.java
  - Not using method to output class of the ontology
- Main1.java
  - Using defined SimpleReadOntology method to output class of the ontology

```
public static void SimpleReadOntology(OntModel model) {
    for (ExtendedIterator i = model.listClasses(); i.hasNext();) {
        OntClass c = (OntClass) i.next();
        System.out.println(c.getLocalName());
       }
}
```

Expert - individual

- MainIndividual.java
  - C:\Jena\Tutorial\expert\MainIndividual.java, defined prix is default namespace from expert.owl, using getInstance method.
  - But you have to compile ontologyDB.java, as all the Main* files are calling the methods defined in ontologyDB.class

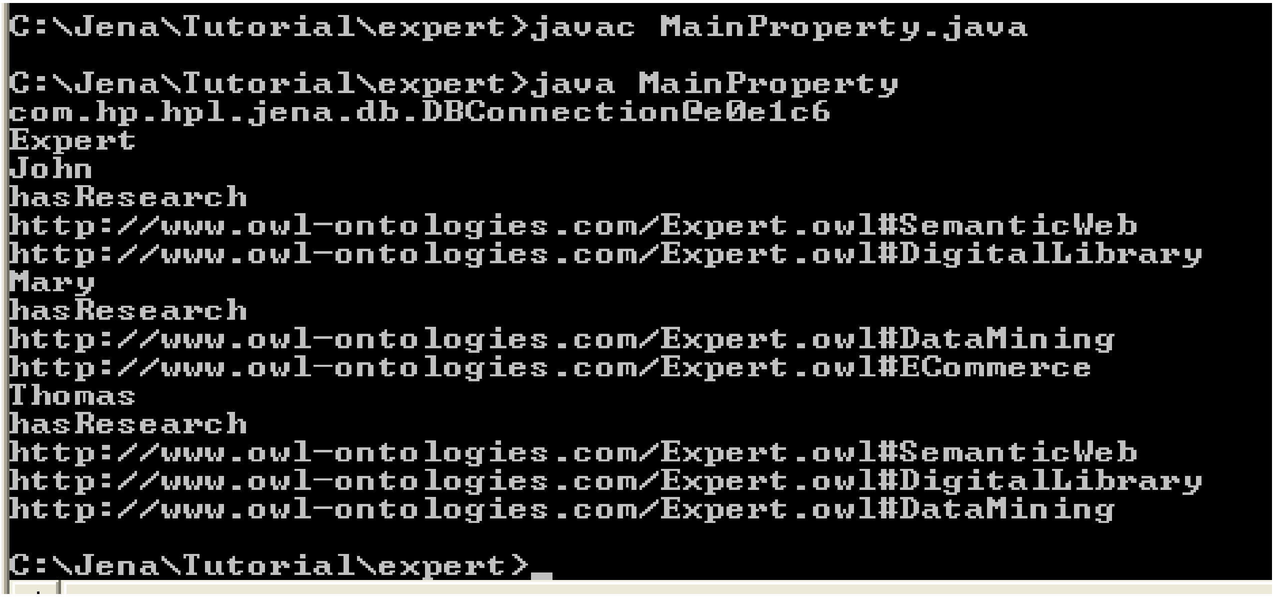
```
public static void getInstance(OntModel model){
    String prix = "http://www.owl-ontologies.com/Expert.owl#";
    /*get Expert class from the onotlogy*/
    OntClass expert = model.getOntClass(prix + "Expert");
    //print out the name of the Expert class
    System.out.println(expert.getLocalName());

    //get instances of the Expert class
    ExtendedIterator it = expert.listInstances();
    //print out the instances of the Expert class
    while(it.hasNext()){
        Individual oi = (Individual)it.next();
        System.out.println(oi.getLocalName());
    }
}
```



```
C:\Jena\Tutorial\expert>javac MainIndividual.java

C:\Jena\Tutorial\expert>java MainIndividual
com.hp.hpl.jena.db.DBConnection@156ee8e
Expert
John
Mary
Thomas

C:\Jena\Tutorial\expert>_
```

Expert - property

- MainProperty.java
  - C:\Jena\Tutorial\expert\MainProperty.java
  - Using getProperty method
  - Compile ontologyDB.java first



```
C:\Jena\Tutorial\expert>javac MainProperty.java

C:\Jena\Tutorial\expert>java MainProperty
com.hp.hpl.jena.db.DBConnection@e0e1c6
Expert
John
hasResearch
http://www.owl-ontologies.com/Expert.owl#SemanticWeb
http://www.owl-ontologies.com/Expert.owl#DigitalLibrary
Mary
hasResearch
http://www.owl-ontologies.com/Expert.owl#DataMining
http://www.owl-ontologies.com/Expert.owl#ECommerce
Thomas
hasResearch
http://www.owl-ontologies.com/Expert.owl#SemanticWeb
http://www.owl-ontologies.com/Expert.owl#DigitalLibrary
http://www.owl-ontologies.com/Expert.owl#DataMining

C:\Jena\Tutorial\expert>_
```

Expert – getProperty method

```
public static void getProperty(OntModel model) {
  String NS = "http://www.owl-ontologies.com/Expert.owl#";
  /* get the Expert class */
  OntClass expert = model.getOntClass(NS + "Expert");
  // print out the name of the Expert class
  System.out.println(expert.getLocalName());

  // get the instances of the Expert class
  ExtendedIterator it = expert.listInstances();
  // print out the instances of the Expert class
  while (it.hasNext()) {
   Individual oi = (Individual) it.next();
   System.out.println(oi.getLocalName());

   //get the properties of the instances of the Expert class
   for (ExtendedIterator ipp = expert.listDeclaredProperties(); ipp.hasNext();) {
        OntProperty p = (OntProperty) ipp.next();
       //print out property name and its values
       System.out.println( p.getLocalName());

        for (ExtendedIterator ivv = oi.listPropertyValues(p); ivv.hasNext();) {
           String valuename = ivv.next().toString();
           System.out.println(valuename); }         }
    }
}
```

Query Expert

- Using SPARQL and Jena Reasoner
- Add familiar_with property (domain: Expert, range: Subject) to expert.owl.
    - There is no instance for this property.
    - We will use reasoner to find its inferred instances
- Query: list the experts and their familiar_with subjects.
- Setting up rules for the reasoner:
    - ExpertA hasResearch ResearchB, ResearchB is associated with SubjectC → ExpertA is familiar_with SubjectC
- Using Sparql to query ExpertX (is familir_with) SubjectY
    - Select ?expert ?subject
    - Where { ?expert faimilar_with ?subject }

MainQuery.java

```java
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.query.* ;
import com.hp.hpl.jena.sparql.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.db.*;

public class MainQuery  {
    public static final String DB_URL = "jdbc:mysql://localhost/expert";
    public static final String DB_USER = "root";
    public static final String DB_PASSWD = "111";
    public static final String DB = "MySQL";
    public static final String DB_DRIVER = "com.mysql.jdbc.Driver";
  public static void main (String args[]) {
     try {
        Class.forName("com.mysql.jdbc.Driver");
          } catch (ClassNotFoundException e) {
          e.printStackTrace();     }
    String filePath = "file:C://Jena//Tutorial//expert//expert.owl";
    IDBConnection con = ontologyDB.connectDB(DB_URL,DB_USER, DB_PASSWD, DB);
    System.out.println(con);
    ontologyDB.createDBModelFromFile(con, "expert", filePath);
    OntModel model = ontologyDB.getModelFromDB(con, "expert");

    searchOnto(model);        }
```

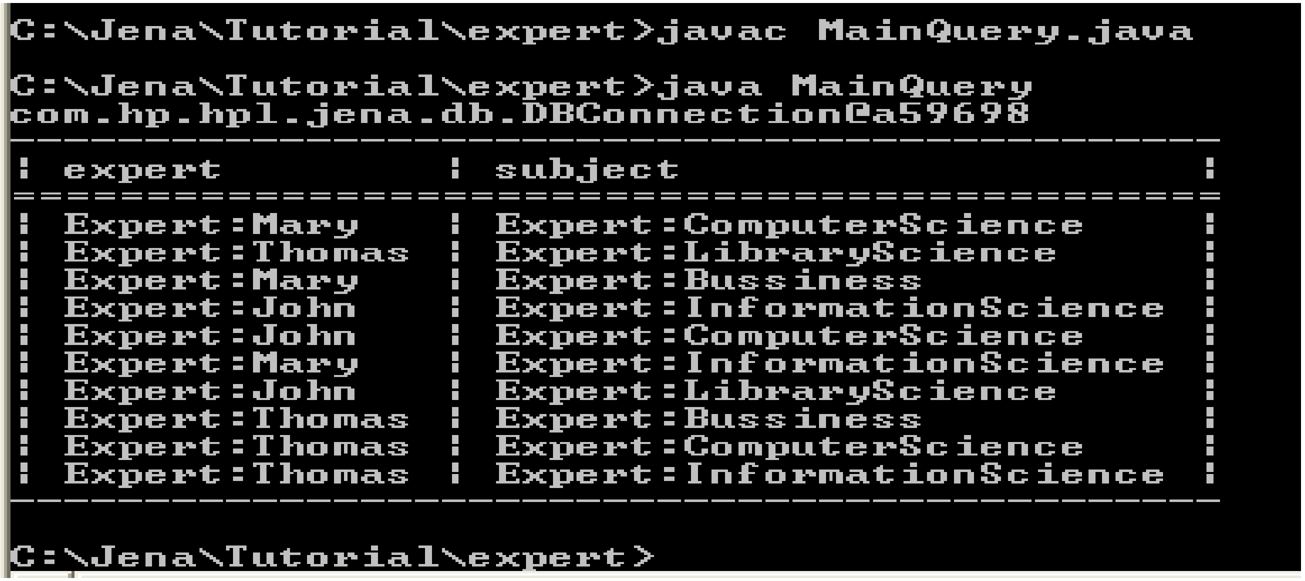
MainQuery.java

```
public static void searchOnto(OntModel model){
     /*Setting up rules*/
     String rule = "[rule1:(?x http://www.owl-ontologies.com/Expert.owl#hasResearch ?y) " +
         "(?y http://www.owl-ontologies.com/Expert.owl#associate ?z) " +
         "->(?x http://www.owl-ontologies.com/Expert.owl#familiar_with ?z)]";

     /*query String*/
     String queryString = "PREFIX Expert:<http://www.owl-ontologies.com/Expert.owl#> " +
     "SELECT ?expert ?subject " +
     "WHERE {?expert Expert:familiar_with ?subject} ";

     /*set up reasoner*/
     Reasoner reasoner2 = new GenericRuleReasoner(Rule.parseRules(rule));

     InfModel inf = ModelFactory.createInfModel(reasoner2, model);
     Query query = QueryFactory.create(queryString);

     QueryExecution qe = QueryExecutionFactory.create(query, inf);
     ResultSet results = qe.execSelect();

     /*output result*/
     ResultSetFormatter.out(System.out, results, query);
     qe.close();    }
}
```

C:C:\Jena\Tutorial\expert\MainQuery.java

Query Expert

- MainQuery.java
  - C:\Jena\Tutorial\expert\MainQuery.java
  - Using searchOnto method
  - Compile ontologyDB.java first



```
C:\Jena\Tutorial\expert>javac MainQuery.java

C:\Jena\Tutorial\expert>java MainQuery
com.hp.hpl.jena.db.DBConnection@a59698
---------------------------------------------
| expert         | subject                  |
=============================================
| Expert:Mary    | Expert:ComputerScience   |
| Expert:Thomas  | Expert:LibraryScience    |
| Expert:Mary    | Expert:Bussiness         |
| Expert:John    | Expert:InformationScience |
| Expert:John    | Expert:ComputerScience   |
| Expert:Mary    | Expert:InformationScience |
| Expert:John    | Expert:LibraryScience    |
| Expert:Thomas  | Expert:Bussiness         |
| Expert:Thomas  | Expert:ComputerScience   |
| Expert:Thomas  | Expert:InformationScience |
---------------------------------------------

C:\Jena\Tutorial\expert>
```

Family Tree

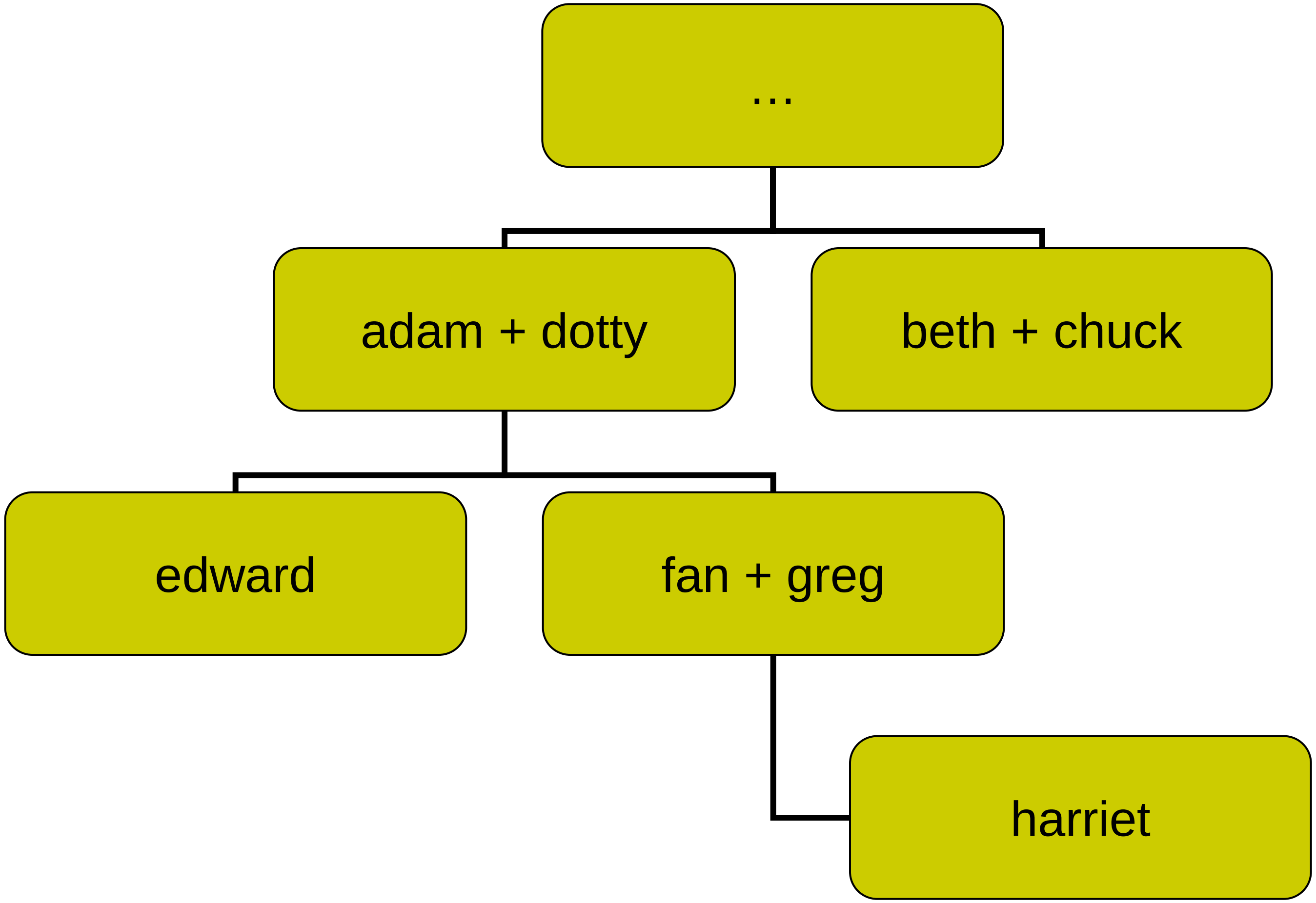
Family Tree

- The example shows:
  - How to create and populate RDF models
  - How to persist them to a database,
  - How to query them programmatically using SPARQL query language
  - How to show Jena reasoning capabilities which can be used to infer knowledge about models from an ontology
- URL: http://www-128.ibm.com/developerworks/xml/library/j-jena/

Creating a simple RDF model

- Create a model from scratch and add RDF statements to it.
- Create a model to represent the relationships in a family using different relationship types, such as siblingOf, spouseOf, parentOf, childOf (more details about relationship ontology: http://vocab.org/relationship/)
- Define family members using URIs from a made-up namespace: http://family/. It is useful to declare them as Java constants.

Family Tree



Creating a simple RDF model

```
import java.util.*;
import java.io.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;

//A small family tree held in a Jena Model
public class FamilyModel {
  // Namespace declarations
  static final String familyUri = "http://family/";
  static final String relationshipUri = "http://purl.org/vocab/relationship/";

    public static void main(String args[]) {

// Create an empty Model
    Model model = ModelFactory.createDefaultModel(); . . .


//set namespace
    Resource NAMESPACE = model.createResource( relationshipUri );
         model.setNsPrefix( "rela", relationshipUri);

// Create the types of Property we need to describe relationships in the model
    Property childOf = model.createProperty(relationshipUri,"childOf"); . . .

// Create resources representing the people in our model
    Resource adam = model.createResource(familyUri+"adam");. . .
```

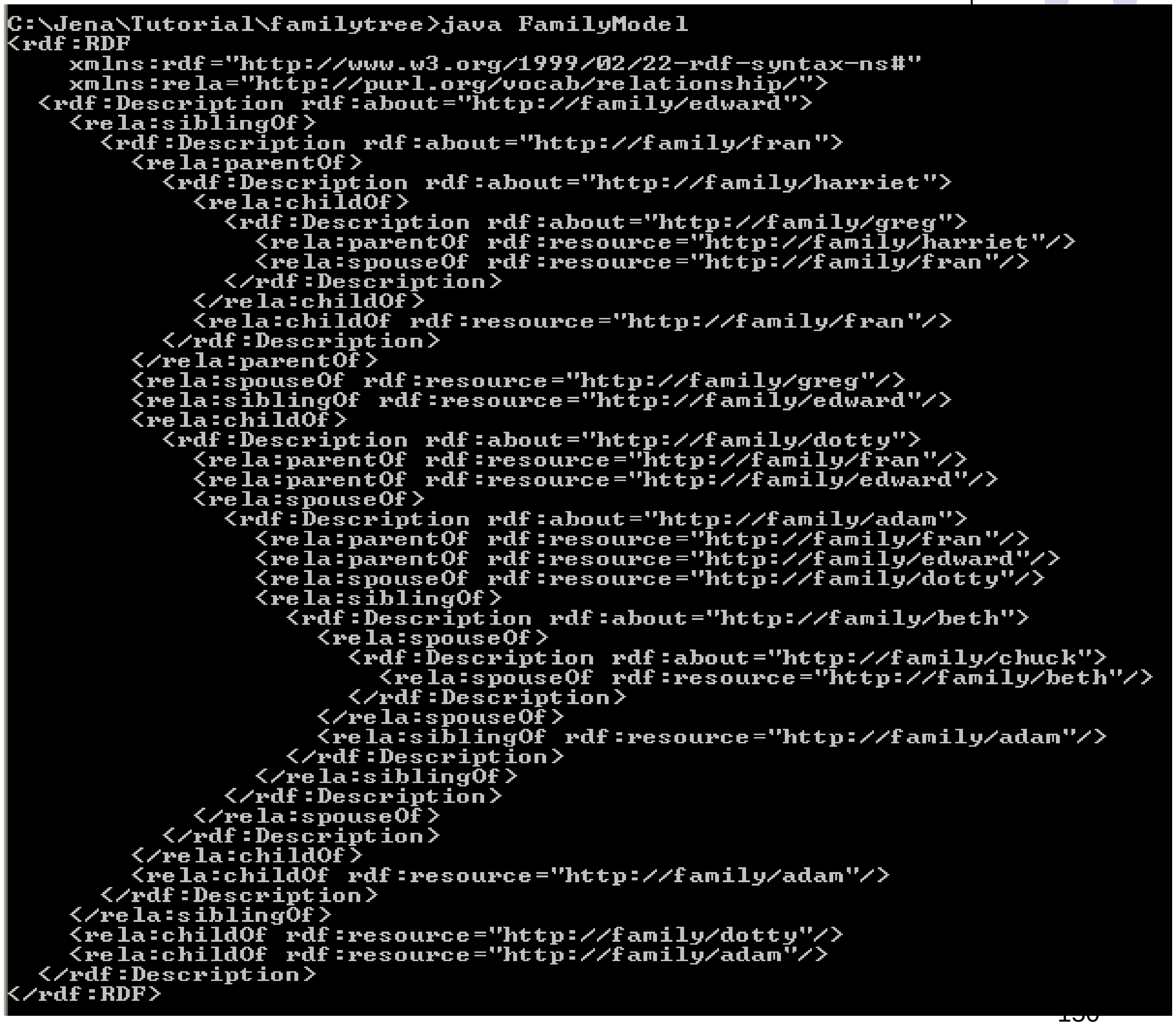
Creating a simple RDF model

```
// Add properties to describing the relationships between them
    adam.addProperty(siblingOf,beth); . . .

// Statements can also be directly created ...
    Statement statement1 = model.createStatement(edward,childOf,adam); . . .

// ... then added to the model:
    model.add(statement1); . . .

// Arrays of Statements can also be added to a Model:
    Statement statements[] = new Statement[5];
    statements[0] = model.createStatement(fran,childOf,adam); . . .

// A List of Statements can also be added
    List list = new ArrayList(); . . .

    list.add(model.createStatement(greg,spouseOf,fran)); . . .
    model.add(list);

    model.write(System.out, "RDF/XML-ABBREV");
          }
}
```

C:\Jena\Tutorial\familytree\
FamilyModel.java

Creating a simple RDF model

- Store FamilyModel.java in C:\Jena\Tutorial\familytree
- Compile and Run



```
C:\Jena\Tutorial\familytree>java FamilyModel
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rela="http://purl.org/vocab/relationship/">
  <rdf:Description rdf:about="http://family/edward">
    <rela:siblingOf>
      <rdf:Description rdf:about="http://family/fran">
        <rela:parentOf>
          <rdf:Description rdf:about="http://family/harriet">
            <rela:childOf>
              <rdf:Description rdf:about="http://family/greg">
                <rela:parentOf rdf:resource="http://family/harriet"/>
                <rela:spouseOf rdf:resource="http://family/fran"/>
              </rdf:Description>
            </rela:childOf>
            <rela:childOf rdf:resource="http://family/fran"/>
          </rdf:Description>
        </rela:parentOf>
        <rela:spouseOf rdf:resource="http://family/greg"/>
        <rela:siblingOf rdf:resource="http://family/edward"/>
        <rela:childOf>
          <rdf:Description rdf:about="http://family/dotty">
            <rela:parentOf rdf:resource="http://family/fran"/>
            <rela:parentOf rdf:resource="http://family/edward"/>
            <rela:spouseOf>
              <rdf:Description rdf:about="http://family/adam">
                <rela:parentOf rdf:resource="http://family/fran"/>
                <rela:parentOf rdf:resource="http://family/edward"/>
                <rela:spouseOf rdf:resource="http://family/dotty"/>
                <rela:siblingOf>
                  <rdf:Description rdf:about="http://family/beth">
                    <rela:spouseOf>
                      <rdf:Description rdf:about="http://family/chuck">
                        <rela:spouseOf rdf:resource="http://family/beth"/>
                      </rdf:Description>
                    </rela:spouseOf>
                    <rela:siblingOf rdf:resource="http://family/adam"/>
                  </rdf:Description>
                </rela:siblingOf>
              </rdf:Description>
            </rela:spouseOf>
          </rdf:Description>
        </rela:childOf>
        <rela:childOf rdf:resource="http://family/adam"/>
      </rdf:Description>
    </rela:siblingOf>
    <rela:childOf rdf:resource="http://family/dotty"/>
    <rela:childOf rdf:resource="http://family/adam"/>
  </rdf:Description>
</rdf:RDF>
```

Store this RDF model in file

- Store the RDF model in to C:\Jena\Tutorial\familytree\family.rdf
- Add following codes to FamilyModel.java

```
try{
  File file=new File("C:\\Jena\\Tutorial\\familytree\\family.rdf");
        FileOutputStream f1=new FileOutputStream(file);
        RDFWriter d = model.getWriter("RDF/XML-ABBREV");
                       d.write(model,f1,null);
        }catch(Exception e) {}
```

C:\Jena\Tutorial\familytree\FamilyModel01.java

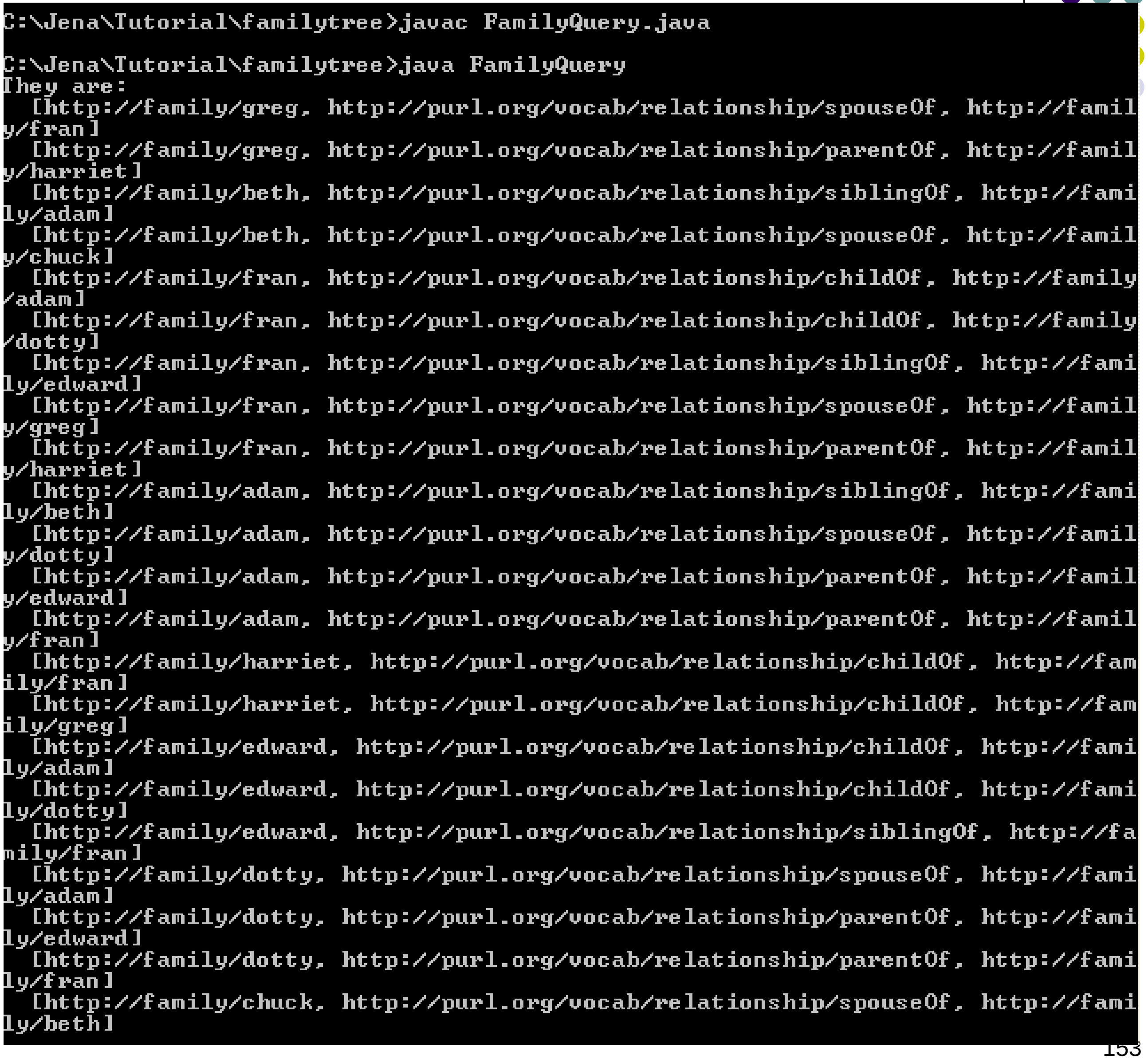
Query family tree - listStatement

- Query : Show me who has which-kind-of relation with whom.
  - listStatement (Subject s, Property p, RDFNode o)

C:\Jena\Tutorial\familytree\
FamilyQuery.java

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.vocabulary.*;
import java.io.*;

public class FamilyQuery {
  static final String inputFileName = "family.rdf";
  public static void main (String args[]) {
    // create an empty model
    Model model = ModelFactory.createDefaultModel();
    // use the FileManager to find the input file
    InputStream in = FileManager.get().open(inputFileName);
      if (in == null) {
         throw new IllegalArgumentException( "File: " + inputFileName + " not found"); }
    model.read( in, "");
    //query the statement:subject, property and object
    StmtIterator iter = model.listStatements(null, null, (RDFNode) null);
        if (iter.hasNext()) {System.out.println("They are:");
            while (iter.hasNext()) {System.out.println("  " + iter.nextStatement());  }
        } else {System.out.println("They are not in the database");    }
    }
}
```

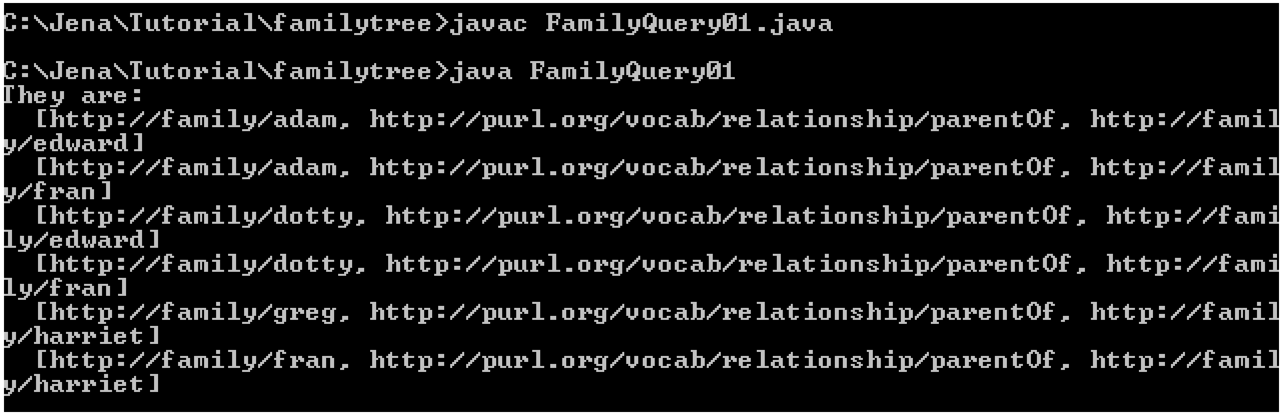
Run FamilyQuery.java



```
C:\Jena\Tutorial\familytree>javac FamilyQuery.java

C:\Jena\Tutorial\familytree>java FamilyQuery
They are:
  [http://family/greg, http://purl.org/vocab/relationship/spouseOf, http://famil
y/fran]
  [http://family/greg, http://purl.org/vocab/relationship/parentOf, http://famil
y/harriet]
  [http://family/beth, http://purl.org/vocab/relationship/siblingOf, http://fami
ly/adam]
  [http://family/beth, http://purl.org/vocab/relationship/spouseOf, http://famil
y/chuck]
  [http://family/fran, http://purl.org/vocab/relationship/childOf, http://family
/adam]
  [http://family/fran, http://purl.org/vocab/relationship/childOf, http://family
/dotty]
  [http://family/fran, http://purl.org/vocab/relationship/siblingOf, http://fami
ly/edward]
  [http://family/fran, http://purl.org/vocab/relationship/spouseOf, http://famil
y/greg]
  [http://family/fran, http://purl.org/vocab/relationship/parentOf, http://famil
y/harriet]
  [http://family/adam, http://purl.org/vocab/relationship/siblingOf, http://fami
ly/beth]
  [http://family/adam, http://purl.org/vocab/relationship/spouseOf, http://famil
y/dotty]
  [http://family/adam, http://purl.org/vocab/relationship/parentOf, http://famil
y/edward]
  [http://family/adam, http://purl.org/vocab/relationship/parentOf, http://famil
y/fran]
  [http://family/harriet, http://purl.org/vocab/relationship/childOf, http://fam
ily/fran]
  [http://family/harriet, http://purl.org/vocab/relationship/childOf, http://fam
ily/greg]
  [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fami
ly/adam]
  [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fami
ly/dotty]
  [http://family/edward, http://purl.org/vocab/relationship/siblingOf, http://fa
mily/fran]
  [http://family/dotty, http://purl.org/vocab/relationship/spouseOf, http://fami
ly/adam]
  [http://family/dotty, http://purl.org/vocab/relationship/parentOf, http://fami
ly/edward]
  [http://family/dotty, http://purl.org/vocab/relationship/parentOf, http://fami
ly/fran]
  [http://family/chuck, http://purl.org/vocab/relationship/spouseOf, http://fami
ly/beth]
```

Query family tree - listStatement

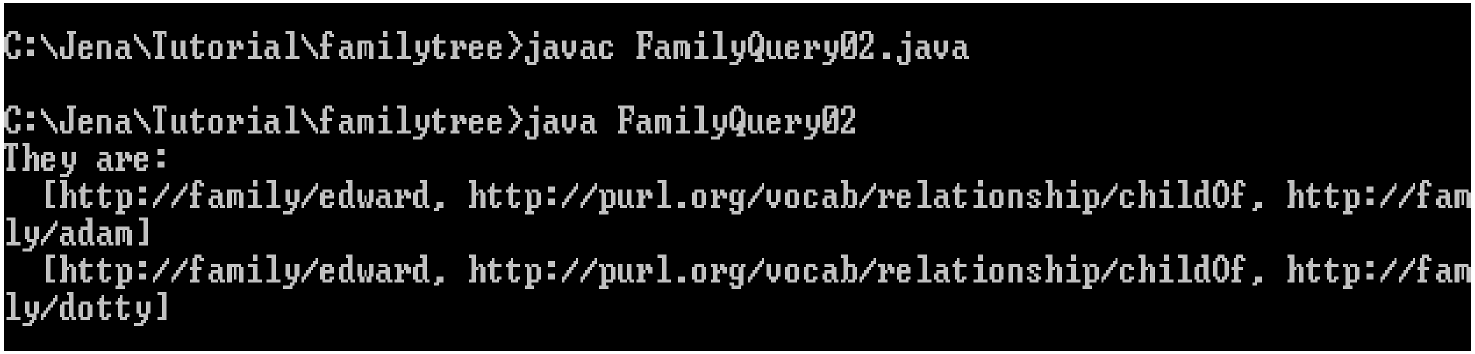
- Query01: show me who are the parent of whom
  - listStatements(null, model.getProperty("http://purl.org/vocab/relationship/parentOf"), (RDFNode) null)
  - C:\Jena\Tutorial\familytree\FamilyQuery01.java



```
C:\Jena\Tutorial\familytree>javac FamilyQuery01.java

C:\Jena\Tutorial\familytree>java FamilyQuery01
They are:
  [http://family/adam, http://purl.org/vocab/relationship/parentOf, http://famil
y/edward]
  [http://family/adam, http://purl.org/vocab/relationship/parentOf, http://famil
y/fran]
  [http://family/dotty, http://purl.org/vocab/relationship/parentOf, http://fami
ly/edward]
  [http://family/dotty, http://purl.org/vocab/relationship/parentOf, http://fami
ly/fran]
  [http://family/greg, http://purl.org/vocab/relationship/parentOf, http://famil
y/harriet]
  [http://family/fran, http://purl.org/vocab/relationship/parentOf, http://famil
y/harriet]
```

Query family tree - listStatement

- Query02: who are parent of edward
  - model.listStatements(model.getResource("http://family/edward"), model.getProperty("http://purl.org/vocab/relationship/childOf"), (RDFNode) null)
  - C:\Jena\Tutorial\familytree\FamilyQuery02.java



```
C:\Jena\Tutorial\familytree>javac FamilyQuery02.java

C:\Jena\Tutorial\familytree>java FamilyQuery02
They are:
  [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fam
ly/adam]
  [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fam
ly/dotty]
```

Query family tree - Sparql

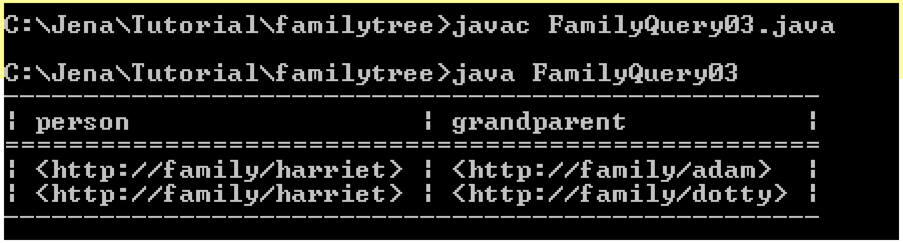
- Find grandparent?

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.query.* ;
import com.hp.hpl.jena.sparql.*;
import java.io.*;

public class FamilyQuery03 {

  static final String inputFileName = "family.rdf";
  public static void main (String args[]) {
    // create an empty model
    Model model = ModelFactory.createDefaultModel();
    // use the FileManager to find the input file
    InputStream in = FileManager.get().open(inputFileName);
    if (in == null) {
            throw new IllegalArgumentException( "File: " + inputFileName + " not found"); }
    model.read( in, "");
```

Query family tree - Sparql

```
String queryString = "PREFIX rela: <http://purl.org/vocab/relationship/> " +
                             "SELECT ?person ?grandparent " +
                             "WHERE {" +
                             "      ?grandparent rela:parentOf ?y . " +
                             "      ?y rela:parentOf ?person . " +
                             "      }";

    Query query = QueryFactory.create(queryString);

    // Execute the query and obtain results
    QueryExecution qe = QueryExecutionFactory.create(query, model);
    ResultSet results = qe.execSelect();

    // Output query results
    ResultSetFormatter.out(System.out, results, query);

    // Important - free up resources used running the query
    qe.close();
    }
}
```

C:\Jena\Tutorial\familytree\FamilyQuery03.java

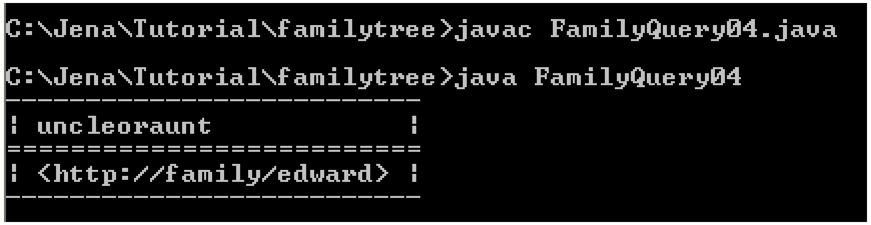


```
C:\Jena\Tutorial\familytree>javac FamilyQuery03.java

C:\Jena\Tutorial\familytree>java FamilyQuery03
----------------------------------------------------
¦ person                   ¦ grandparent          ¦
====================================================
¦ <http://family/harriet> ¦ <http://family/adam>  ¦
¦ <http://family/harriet> ¦ <http://family/dotty> ¦
----------------------------------------------------
```

Query family tree - Sparql

- Who is uncle of harriet?
- C:\Jena\Tutorial\familytree\FamilyQuery04.java

```
String queryString =
       "PREFIX rela: <http://purl.org/vocab/relationship/> " +
       "SELECT ?uncleoraunt " +
       "WHERE {" +
       "      <http://family/harriet> rela:childOf ?x . " +
       "      ?x rela:siblingOf ?uncleoraunt . " +
       "      }";
```



```
C:\Jena\Tutorial\familytree>javac FamilyQuery04.java

C:\Jena\Tutorial\familytree>java FamilyQuery04
--------------------------
| uncleoraunt            |
==========================
| <http://family/edward> |
--------------------------
```

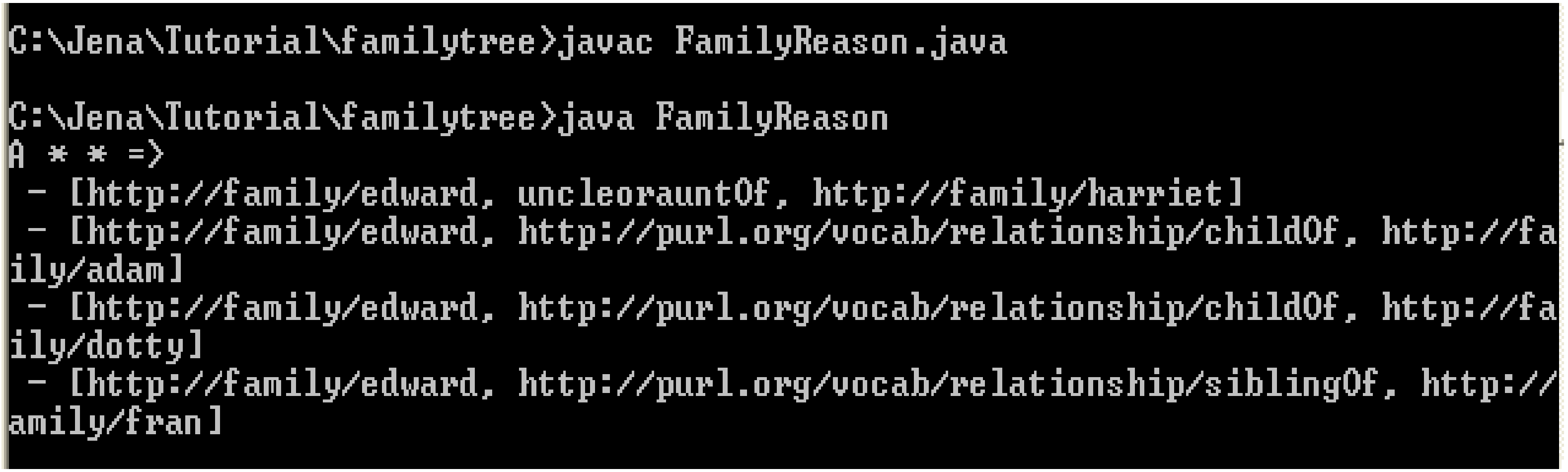
Reasoning family tree

- Who are niece or nephew of edwar?

C:\Jena\Tutorial\familytree\
FamilyReason.java

```
import java.io.*;
import java.util.Iterator;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.reasoner.rulesys.*;

public class FamilyReason {
  private static String fname = "family.rdf";
  private static String NS = "http://family/";
  public static void main(String args[]) {
    Model rawData = FileManager.get().loadModel(fname);
    String rules = "[r1: (?x http://purl.org/vocab/relationship/siblingOf ?y), (?y
          http://purl.org/vocab/relationship/parentOf ?z)" + "-> (?x uncleorauntOf ?z)]";

    Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
          InfModel inf = ModelFactory.createInfModel(reasoner, rawData);
          Resource A = inf.getResource(NS + "edward");
          System.out.println("A * * =>");
          Iterator list = inf.listStatements(A, null, (RDFNode)null);
          while (list.hasNext()) {
          System.out.println(" - " + list.next()); }
          }
}
```

Reasoning family tree



```
C:\Jena\Tutorial\familytree>javac FamilyReason.java

C:\Jena\Tutorial\familytree>java FamilyReason
A * * =>
 - [http://family/edward, uncleorauntOf, http://family/harriet]
 - [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fa
ily/adam]
 - [http://family/edward, http://purl.org/vocab/relationship/childOf, http://fa
ily/dotty]
 - [http://family/edward, http://purl.org/vocab/relationship/siblingOf, http://
amily/fran]
```

Reasoning family tree

- Using FamilyModel02.java to delete some statements (siblingOf, childOf, spouseOf,…) and store it in N-Triple in family1.nt.
- Who are the children of dotty?
- Using generic rule reasoning
- Using sparql to query
- C:\Jena\Tutorial\familytree\FamilyReason01.java

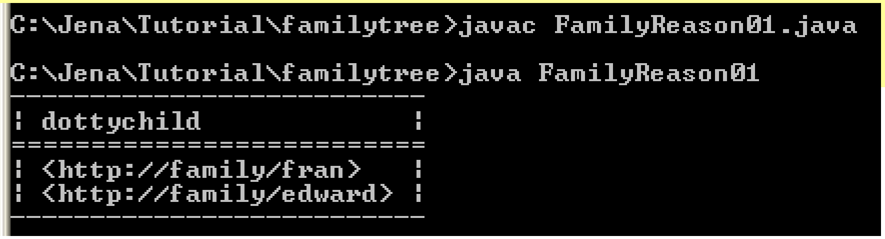
FamilyReason01.java

```
import java.io.*;
import java.util.Iterator;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.query.* ;
import com.hp.hpl.jena.sparql.*;

public class FamilyReason01 {
  private static String fname = "family1.nt";

  public static void main(String args[]) {
    Model rawData = FileManager.get().loadModel(fname);

    //setting up rules
    String rules = "[r1: (?x http://purl.org/vocab/relationship/parentOf ?y),
         (?x http://purl.org/vocab/relationship/spouseOf ?z)" +
                 "-> (?z http://purl.org/vocab/relationship/parentOf ?y)]";
```

FamilyReason01.java

C:\Jena\Tutorial\familytree\
FamilyReason01.java

```
   /*query String*/
   String queryString = "SELECT ?dottychild " +
     "WHERE { <http://family/dotty>
         <http://purl.org/vocab/relationship/parentOf> ?dottychild} ";

   Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
     InfModel inf = ModelFactory.createInfModel(reasoner, rawData);
     Query query = QueryFactory.create(queryString);
     QueryExecution qe = QueryExecutionFactory.create(query, inf);
     ResultSet results = qe.execSelect();

    /*output result*/
    ResultSetFormatter.out(System.out, results, query);
    qe.close();
         }
}
```



```
C:\Jena\Tutorial\familytree>javac FamilyReason01.java

C:\Jena\Tutorial\familytree>java FamilyReason01
--------------------------
| dottychild             |
==========================
| <http://family/fran>   |
| <http://family/edward> |
--------------------------
```

Reasoning family tree

- Using multiple rules to do complex reasoning.
- Dataset: C:\Jena\Tutorial\familytree\family2.nt

```
<http://family/harriet> <http://purl.org/vocab/relationship/childOf> <http://family/fran> .
<http://family/fran> <http://purl.org/vocab/relationship/spouseOf> <http://family/greg> .
<http://family/fran> <http://purl.org/vocab/relationship/childOf> <http://family/adam> .
<http://family/adam> <http://purl.org/vocab/relationship/spouseOf> <http://family/dotty> .
<http://family/adam> <http://purl.org/vocab/relationship/siblingOf> <http://family/beth> .
<http://family/beth> <http://purl.org/vocab/relationship/spouseOf> <http://family/chuck> .
<http://family/edward> <http://purl.org/vocab/relationship/siblingOf> <http://family/fran> .
<http://family/edward> <http://purl.org/vocab/relationship/childOf> <http://family/adam> .
```

FamilyReason02.java

- Multiple rules:
  - R1: ?x parentOf ?y, ?y parentOf ?z ->?x grandparentOf ?z
  - R2: ?x parentOf ?y, ?x spouseOf ?z ->?z parentOf ?y
  - R3: ?x childOf ?y -> ?y parentOf ?x
- Query: who can be grandparents?

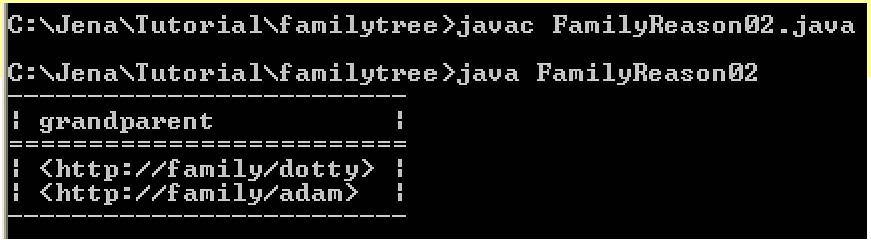
FamilyReason02.java

```
import java.io.*;
import java.util.Iterator;
import com.hp.hpl.jena.util.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.reasoner.*;
import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.query.* ;
import com.hp.hpl.jena.sparql.*;

public class FamilyReason02 {

  private static String fname = "family2.nt";

  public static void main(String args[]) {

    Model rawData = FileManager.get().loadModel(fname);

    //setting up rules
    String rules = "[r1: (?x http://purl.org/vocab/relationship/parentOf ?y),
                    (?y http://purl.org/vocab/relationship/parentOf ?z)" +
                      "-> (?x grandparentOf ?z)]" +
                   "[r2: (?x http://purl.org/vocab/relationship/parentOf ?y),
                    (?x http://purl.org/vocab/relationship/spouseOf ?z)" +
                      "-> (?z http://purl.org/vocab/relationship/parentOf ?y)]" +
                    "[r3: (?x http://purl.org/vocab/relationship/childOf ?y)" +
                      "-> (?y http://purl.org/vocab/relationship/parentOf ?x)]";
```

FamilyReason02.java

```
/*query String*/
    String queryString = "SELECT ?grandparent " +
     "WHERE { ?grandparent <http://purl.org/vocab/relationship/parentOf> ?x ." +
                    " ?x <http://purl.org/vocab/relationship/parentOf> ?y . } ";

    Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
    InfModel inf = ModelFactory.createInfModel(reasoner, rawData);

    Query query = QueryFactory.create(queryString);

    QueryExecution qe = QueryExecutionFactory.create(query, inf);
    ResultSet results = qe.execSelect();

    /*output result*/
    ResultSetFormatter.out(System.out, results, query);
    qe.close();
          }
}
```

C:\Jena\Tutorial\familytree\FamilyReason02.java



```
C:\Jena\Tutorial\familytree>javac FamilyReason02.java

C:\Jena\Tutorial\familytree>java FamilyReason02
-------------------------
| grandparent           |
=========================
| <http://family/dotty> |
| <http://family/adam>  |
-------------------------
```