

```

//1.A simple hello world program

import com.hp.hpl.jena.rdf.model.Model;

import com.hp.hpl.jena.datatypes.xsd.XSDDatatype;

import com.hp.hpl.jena.rdf.model.ModelFactory;

import com.hp.hpl.jena.rdf.model.Property;

import com.hp.hpl.jena.rdf.model.Resource;

public class HelloRDFWorld{

    public static void main(String[] args){

        Model m =ModelFactory.createDefaultModel();

        String NS="http://example.com/test/";

        Resource r=m.createResource(NS+"r");

        Property p=m.createProperty(NS+ "p");

        r.addProperty(p,"hello world",XSDDatatype.XSDstring);

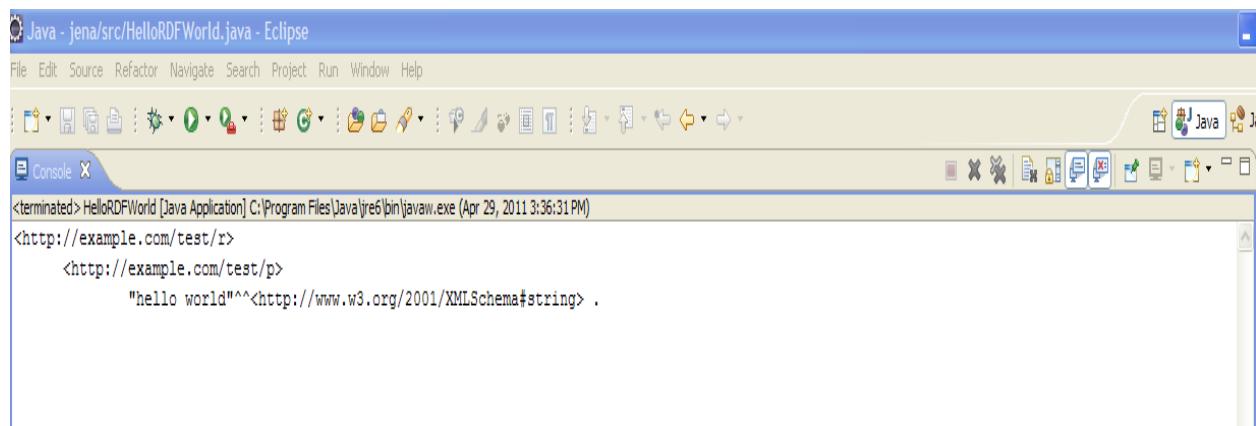
        m.write(System.out,"Turtle");

    }

}

```

Output:



```

//2.Program to output an rdf based on the resources and properties created.

import com.hp.hpl.jena.rdf.model.*;
import java.io.PrintWriter;
public class triple extends Object {
public static void main (String args[]) {
    String sURI =      "http://tinman.cs.gsu.edu#";
    String sRelated = "fullname";
    String sCourse = "course";
try {
// Create an empty graph
    Model model = ModelFactory.createDefaultModel();
    model.setNsPrefix( "student", "http://tinman.cs.gsu.edu#" );

// Create the resource
Resource postcon = model.createResource(sURI+"Ranjani");
Resource postcon1 = model.createResource(sURI+"Priyanka");
Resource postcon2 = model.createResource(sURI+"Srujana");

// Create the predicate (property)

//fullname
Property fullname = model.createProperty(sURI, sRelated);
Property fullname1 = model.createProperty(sURI, sRelated);
Property fullname2 = model.createProperty(sURI, sRelated);

//subject
Property course = model.createProperty(sURI, sCourse);
Property course1 = model.createProperty(sURI, sCourse);
Property course2 = model.createProperty(sURI, sCourse);

// Add the properties with associated values (objects)

//property for fullname

postcon.addProperty(fullname,
"Ranjani Sankaran");
postcon1.addProperty(fullname1,
"Priyanka Chebrolu");
postcon2.addProperty(fullname2,
"Srujana George");

//property for course
postcon.addProperty(course,
"Database and the Web");
postcon1.addProperty(course1,
"Human Computer Interaction");
postcon2.addProperty(course2,
"Network Security");

//Print RDF/XML of model to system output
model.write(new PrintWriter(System.out));

} catch (Exception e) {
System.out.println("Failed: " + e);
}

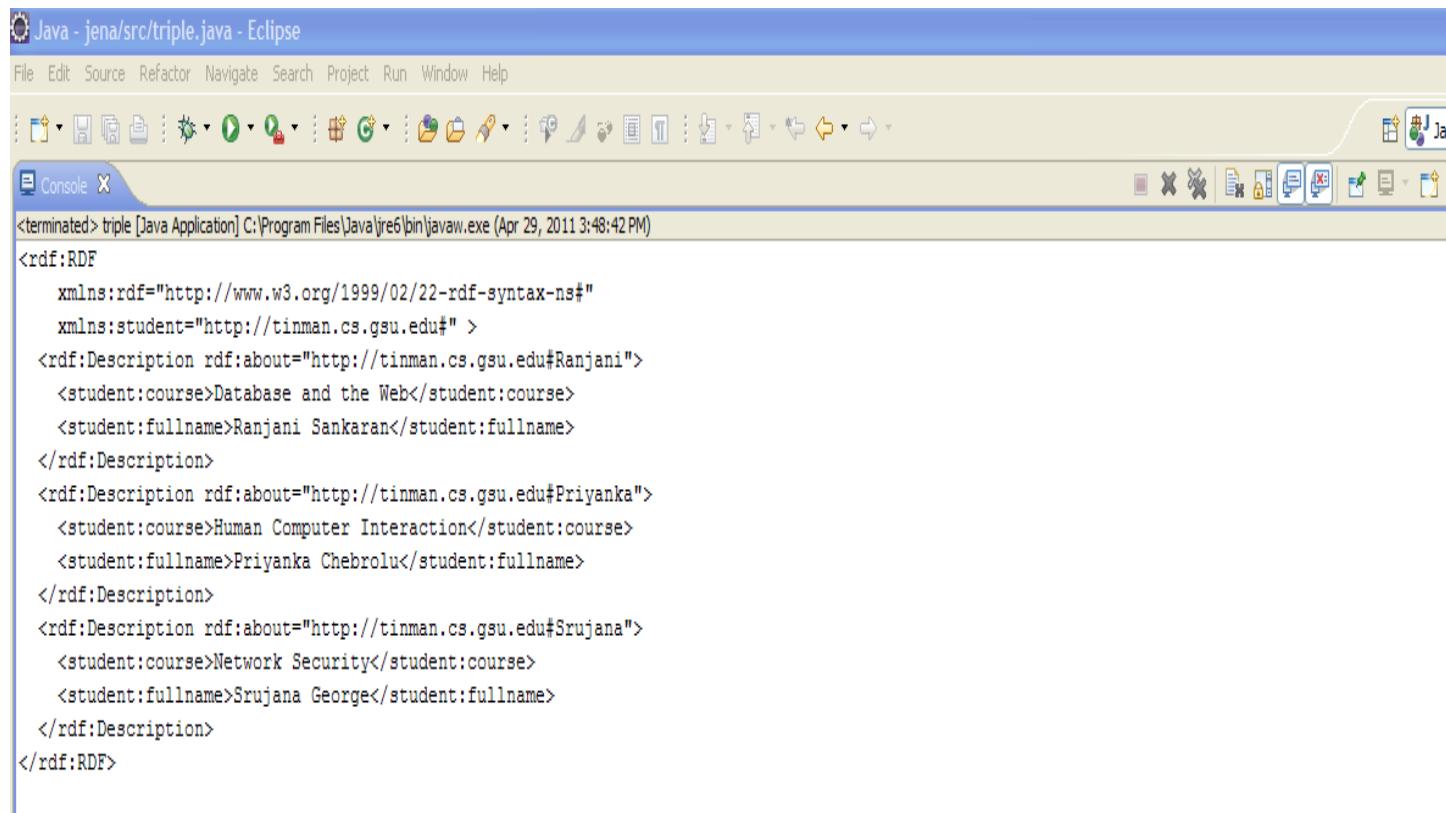
```

```
}
```

```
}
```

```
}
```

**Output:**



The screenshot shows the Eclipse IDE interface with the title bar "Java - jena/src/triple.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. The main area is a "Console" window titled "Console X". The console output is as follows:

```
<terminated> triple [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 3:48:42 PM)
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:student="http://tinman.cs.gsu.edu#" >
  <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Ranjani">
    <student:course>Database and the Web</student:course>
    <student:fullname>Ranjani Sankaran</student:fullname>
  </rdf:Description>
  <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Priyanka">
    <student:course>Human Computer Interaction</student:course>
    <student:fullname>Priyanka Chebrolu</student:fullname>
  </rdf:Description>
  <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Srujana">
    <student:course>Network Security</student:course>
    <student:fullname>Srujana George</student:fullname>
  </rdf:Description>
</rdf:RDF>
```

//3. Creates model and appends resources, properties to it. Iterates through the statements.

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

/**
 * A small family tree held in a Jena Model
 */
public class Student {

    // Namespace declarations
    static final String familyUri = "http://tinman.cs.gsu.edu#";

    // Jena model representing the family
    private Model model;

    /**
     * Creates a model and populates it with family members and their
     * relationships
     */
    private Student() {
        // Create an empty Model
        model = ModelFactory.createDefaultModel();
        model.setNsPrefix("family", familyUri);
        addFamilyMembers();
    }

    /**
     * Adds the family members to the model
     */
    void addFamilyMembers() {
        // Add the father
        Person father = new Person("John");
        father.setGender("Male");
        father.setBirthYear(1950);
        father.setDeathYear(0);
        father.setSpouse(null);
        father.setChildren(null);
        model.add(father);
        System.out.println("Added father: " + father);

        // Add the mother
        Person mother = new Person("Jane");
        mother.setGender("Female");
        mother.setBirthYear(1955);
        mother.setDeathYear(0);
        mother.setSpouse(father);
        mother.setChildren(null);
        model.add(mother);
        System.out.println("Added mother: " + mother);

        // Add the son
        Person son = new Person("Mike");
        son.setGender("Male");
        son.setBirthYear(1980);
        son.setDeathYear(0);
        son.setSpouse(null);
        son.setChildren(null);
        model.add(son);
        System.out.println("Added son: " + son);

        // Add the daughter
        Person daughter = new Person("Sarah");
        daughter.setGender("Female");
        daughter.setBirthYear(1985);
        daughter.setDeathYear(0);
        daughter.setSpouse(null);
        daughter.setChildren(null);
        model.add(daughter);
        System.out.println("Added daughter: " + daughter);

        // Add the spouse relationship
        father.setSpouse(mother);
        model.add(father);
        model.add(mother);
        System.out.println("Added spouse relationship");

        // Add the children relationships
        father.setChildren(son);
        father.setChildren(daughter);
        model.add(father);
        System.out.println("Added children relationships");
    }

    /**
     * Prints the family tree to the console
     */
    void printFamilyTree() {
        System.out.println("Family Tree:");
        System.out.println("Father: " + father);
        System.out.println("Mother: " + mother);
        System.out.println("Son: " + son);
        System.out.println("Daughter: " + daughter);
        System.out.println("Spouse relationship: " + father.getSpouse());
        System.out.println("Children relationships: " + father.getChildren());
    }
}
```

```
model = ModelFactory.createDefaultModel();

model.setNsPrefix( "student", "http://tinman.cs.gsu.edu#" );

String fullName = "fullname";
String College = "collegename";
String Major = "major";
String Course = "course";

// Create the types of Property we need to describe relationships
// in the model

Property fullname = model.createProperty(familyUri,fullName);
Property college = model.createProperty(familyUri,College);
Property major = model.createProperty(familyUri,Major);
Property course = model.createProperty(familyUri,Course);

// Create resources representing the people in our model

Resource ranjani = model.createResource(familyUri+"Ranjani");
Resource priyanka = model.createResource(familyUri+"Priyanka");
Resource srujana = model.createResource(familyUri+"Srujana");
Resource asif = model.createResource(familyUri+"Asif");
Resource harshal = model.createResource(familyUri+"Harshal");
```

```
// Add properties to describing the relationships between them  
  
ranjani.addProperty(fullname,"Ranjani Sankaran");  
  
priyanka.addProperty(fullname,"Chebrolu Krishna Priyanka");  
  
srujana.addProperty(fullname,"Srujana Gorge");  
  
asif.addProperty(fullname,"Mohammad Asif");  
  
harshal.addProperty(fullname,"Harshal Jhaveri");  
  
  
// Statements can also be directly created ...  
  
Statement statement1 = model.createStatement(ranjani,college,"gsu");  
  
Statement statement2 = model.createStatement(priyanka,college,"gsu");  
  
Statement statement3 = model.createStatement(srujana,college,"gsu");  
  
Statement statement4 = model.createStatement(asif,college,"gsu");  
  
Statement statement5 = model.createStatement(harshal,college,"gsu");  
  
  
// ... then added to the model:  
  
model.add(statement1);  
  
model.add(statement2);  
  
model.add(statement3);  
  
model.add(statement4);  
  
model.add(statement5);  
  
  
// Arrays of Statements can also be added to a Model:  
  
Statement statements[] = new Statement[5];
```

```
statements[0] = model.createStatement(ranjani,major,"csc");

statements[1] = model.createStatement(priyanka,major,"csc");

statements[2] = model.createStatement(srujana,major,"csc");

statements[3] = model.createStatement(asif,major,"csc");

statements[4] = model.createStatement(harshal,major,"csc");

model.add(statements);

// A List of Statements can also be added

List list = new ArrayList();

list.add(model.createStatement(ranjani,course,"DB and the Web"));

list.add(model.createStatement(priyanka,course,"Human Computer Interaction"));

list.add(model.createStatement(srujana,course,"Data Mining"));

list.add(model.createStatement(asif,course,"Network Security"));

list.add(model.createStatement(harshal,course,"Design and Analysis of Algorithms"));

model.add(list);

model.write(new PrintWriter(System.out));

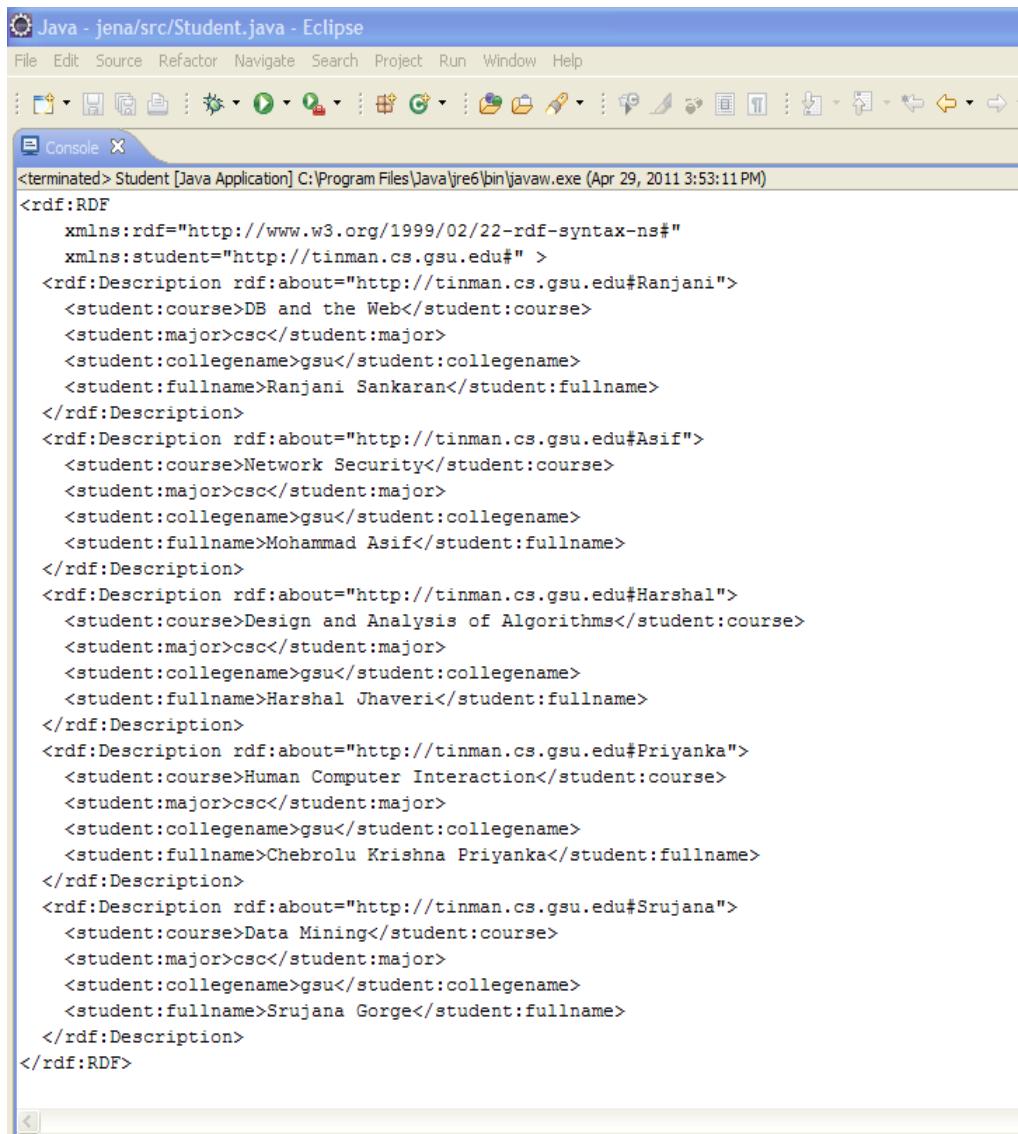
}

/**
 * Creates a FamilyModel and dumps the content of its RDF representation
 */

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

```
// Create a model representing the family  
Student thestudent = new Student();  
  
// Dump out a String representation of the model  
  
}  
}
```

Output:



```
<terminated> Student [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 3:53:11 PM)
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:student="http://tinman.cs.gsu.edu#"
    <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Ranjanji">
        <student:course>DB and the Web</student:course>
        <student:major>csc</student:major>
        <student:collegename>gsu</student:collegename>
        <student:fullname>Ranjanji Sankaran</student:fullname>
    </rdf:Description>
    <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Asif">
        <student:course>Network Security</student:course>
        <student:major>csc</student:major>
        <student:collegename>gsu</student:collegename>
        <student:fullname>Mohammad Asif</student:fullname>
    </rdf:Description>
    <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Harshal">
        <student:course>Design and Analysis of Algorithms</student:course>
        <student:major>csc</student:major>
        <student:collegename>gsu</student:collegename>
        <student:fullname>Harshal Jhaveri</student:fullname>
    </rdf:Description>
    <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Priyanka">
        <student:course>Human Computer Interaction</student:course>
        <student:major>csc</student:major>
        <student:collegename>gsu</student:collegename>
        <student:fullname>Chebrolu Krishna Priyanka</student:fullname>
    </rdf:Description>
    <rdf:Description rdf:about="http://tinman.cs.gsu.edu#Srujana">
        <student:course>Data Mining</student:course>
        <student:major>csc</student:major>
        <student:collegename>gsu</student:collegename>
        <student:fullname>Srujana Gorge</student:fullname>
    </rdf:Description>
</rdf:RDF>
```

```
4. //*****
//List students who have taken a course
ResIterator students_course = model.listSubjectsWithProperty(course);

// Because subjects of statements are Resources, the method returned a
ResIterator
while (students_course.hasNext()) {

    // ResIterator has a typed nextResource() method
    Resource person = students_course.nextResource();

    // Print the URI of the resource
    System.out.println("The list of students who have taken
courses"+person.getURI());
}
```

```

    // Can also find all the courses by getting the objects of all "course"
    statements
    // Objects of statements could be Resources or literals, so the Iterator
    returned
    // contains RDFNodes
    NodeIterator courses = model.listObjectsOfProperty(course);
    System.out.println("****LIST OF COURSES***");
    while (courses.hasNext()) {

        System.out.println(courses.nextNode().toString());
    }

    // To find all the courses taken by a student, the model itself can be
    queried
    NodeIterator moreStudents1 = model.listObjectsOfProperty(priyanka,
    course);
    System.out.println("****LIST OF COURSES TAKEN BY PRIYANKA***");
    while (moreStudents1.hasNext()) {

        System.out.println(moreStudents1.nextNode().toString());
    }

    // But it's more elegant to ask the Resource directly
    // This method yields an iterator over Statements
    StmtIterator moreStudents2 = priyanka.listProperties(course);

    System.out.println("****LIST OF COURSES TAKEN BY PRIYANKA (USING DIRECT
METHOD)***");
    while (moreStudents2.hasNext()) {

        System.out.println(moreStudents2.next().toString());
    }

    //*****USING SELECTORS TO QUERY*****
    // Find the exact statement "priyanka has taken DB and the Web"

    StmtIterator moreStudents5 = model.listStatements(priyanka, course, "DB and
    the Web");
    System.out.println("****priyanka has taken DB and the Web****");

    while (moreStudents5.hasNext()) {

        System.out.println(moreStudents5.next().toString());
    }

    // Find all statements with ranjani as the subject and "csc" as the
    object

    StmtIterator moreStudents6 = model.listStatements(ranjani, null, "csc");
    System.out.println("****ranjani as the subject and \"csc\" as the
    object****");

```

```

while (moreStudents6.hasNext()) {

    System.out.println(moreStudents6.next().toString());
}

```

Output:

```

The list of students who have taken courseshttp://tinman.cs.gsu.edu#Ranjani
The list of students who have taken courseshttp://tinman.cs.gsu.edu#Asif
The list of students who have taken courseshttp://tinman.cs.gsu.edu#Priyanka
The list of students who have taken courseshttp://tinman.cs.gsu.edu#Harshal
The list of students who have taken courseshttp://tinman.cs.gsu.edu#Srujana
****LIST OF COURSES***
Human Computer Interaction
Design and Analysis of Algorithms
Network Security
DB and the Web
Data Mining
****LIST OF COURSES TAKEN BY PRIYANKA***
Human Computer Interaction
DB and the Web
****LIST OF COURSES TAKEN BY PRIYANKA (USING DIRECT METHOD) ***
[http://tinman.cs.gsu.edu#Priyanka, http://tinman.cs.gsu.edu#course, "DB and the Web"]
[http://tinman.cs.gsu.edu#Priyanka, http://tinman.cs.gsu.edu#course, "Human Computer Interaction"]
***priyanka has taken DB and the Web****
[http://tinman.cs.gsu.edu#Priyanka, http://tinman.cs.gsu.edu#course, "DB and the Web"]
***ranjani as the subject and "csc" as the object****
[http://tinman.cs.gsu.edu#Ranjani, http://tinman.cs.gsu.edu#major, "csc"]

```

**//5.Creating model with and without reasoner and executing a simple sparql query**

```

import com.hp.hpl.jena.graph.query.Query;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.rdf.model.ModelFactory;

public class ImportWordnet {

    public static void main (String args[]) {

```

```

String SOURCE = "http://www.opentox.org/api/1.1";
String NS = SOURCE + "#";
//create a model using reasoner
OntModel model1 = ModelFactory.createOntologyModel(
OntModelSpec/owl_mem_micro_rule_inf);
//create a model which doesn't use a reasoner
OntModel model2 = ModelFactory.createOntologyModel(
OntModelSpec/owl_mem);

// read the RDF/XML file
model1.read( SOURCE, "RDF/XML" );
model2.read( SOURCE, "RDF/XML" );
//prints out the RDF/XML structure
// qe.close();
System.out.println(" ");

// Create a new query
String queryString =
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> "+
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> "+
"select ?uri "+
"where { "+
"?uri rdfs:subClassOf <http://www.opentox.org/api/1.1#Feature> "+
"} \n ";
com.hp.hpl.jena.query.Query query = QueryFactory.create(queryString);

System.out.println("-----");
System.out.println("Query Result Sheet");
System.out.println("-----");

System.out.println("Direct&Indirect Descendants (model1)");
System.out.println("-----");

// Execute the query and obtain results
QueryExecution qe = QueryExecutionFactory.create(query, model1);
com.hp.hpl.jena.query.ResultSet results = qe.execSelect();

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

qe.close();

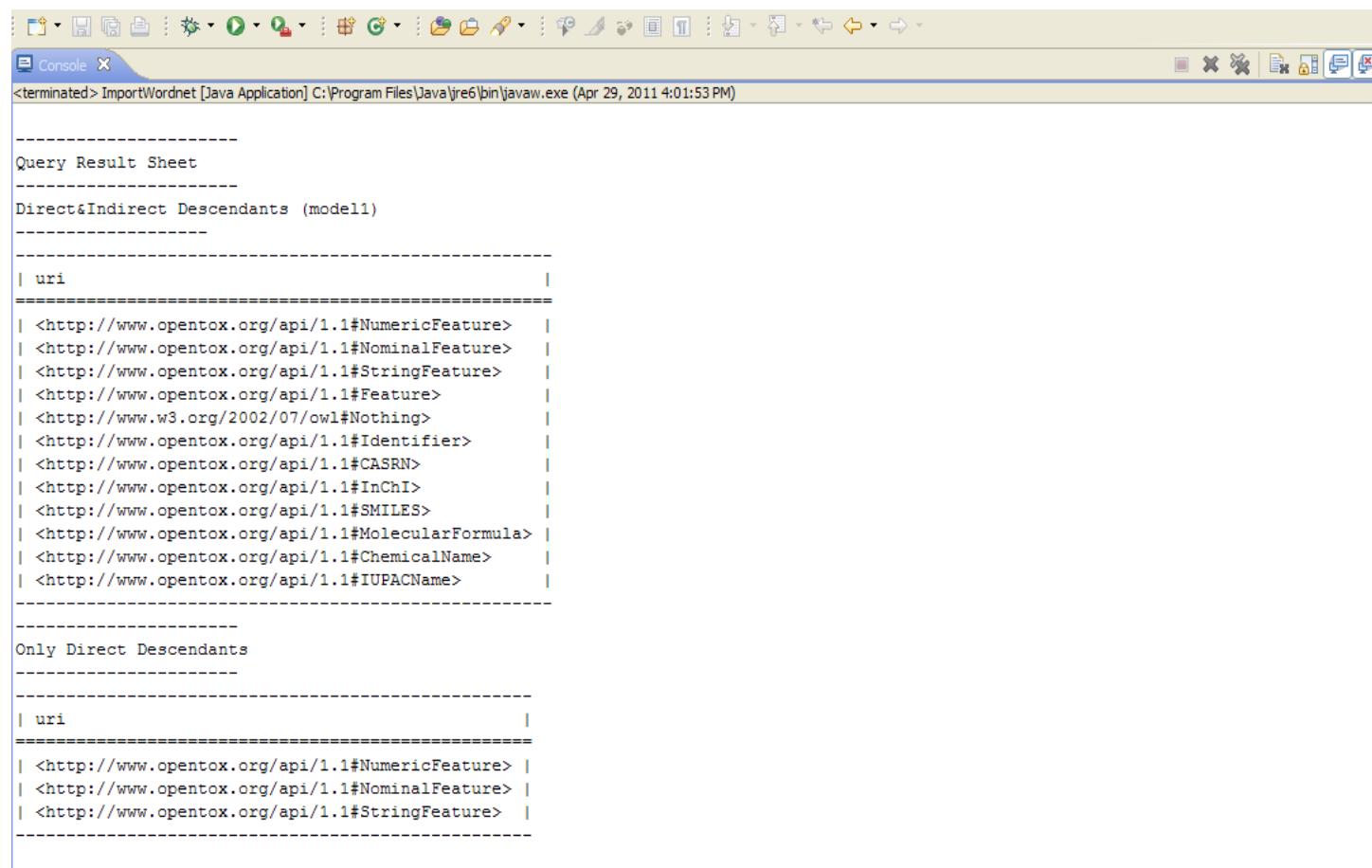
System.out.println("-----");
System.out.println("Only Direct Descendants");
System.out.println("-----");

// Execute the query and obtain results
qe = QueryExecutionFactory.create(query, model2);
results = qe.execSelect();

// Output query results

```

```
        ResultSetFormatter.out(System.out, results, query);
        qe.close();
    }
}
```



The screenshot shows a Java application window titled "ImportWordnet [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 4:01:53 PM)". The console output displays a query result sheet for "Direct&Indirect Descendants (modell1)". The results are listed in two sections: "Direct&Indirect Descendants (modell1)" and "Only Direct Descendants". Each section has a header followed by a table with a single column labeled "uri". The "Direct&Indirect Descendants" section lists approximately 15 URIs, while the "Only Direct Descendants" section lists approximately 3 URIs.

```
<terminated> ImportWordnet [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 4:01:53 PM)

-----
Query Result Sheet
-----
Direct&Indirect Descendants (modell1)
-----

| uri
|-----
| <http://www.opentox.org/api/1.1#NumericFeature>
| <http://www.opentox.org/api/1.1#NominalFeature>
| <http://www.opentox.org/api/1.1#StringFeature>
| <http://www.opentox.org/api/1.1#Feature>
| <http://www.w3.org/2002/07/owl#Nothing>
| <http://www.opentox.org/api/1.1#Identifier>
| <http://www.opentox.org/api/1.1#CASRN>
| <http://www.opentox.org/api/1.1#InChI>
| <http://www.opentox.org/api/1.1#SMILES>
| <http://www.opentox.org/api/1.1#MolecularFormula>
| <http://www.opentox.org/api/1.1#ChemicalName>
| <http://www.opentox.org/api/1.1#IUPACName>
|-----


Only Direct Descendants
-----

| uri
|-----
| <http://www.opentox.org/api/1.1#NumericFeature>
| <http://www.opentox.org/api/1.1#NominalFeature>
| <http://www.opentox.org/api/1.1#StringFeature>
|-----
```

```
//6.Creating model,resources and properties with vcards
```

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

/** Tutorial 3 Statement attribute accessor methods
 *
 * @author bwm - updated by kers/Daniel
 * @version Release='\$Name: \$' Revision='\$Revision: 1.3 \$' Date='\$Date: 2005/10/06 17:49:05
\$'
 */

public class Sgraph extends Object {

    public static void main (String args[]) {

        // 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));

// list the statements in the graph

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() + "\\");
}
}

```

```

    }

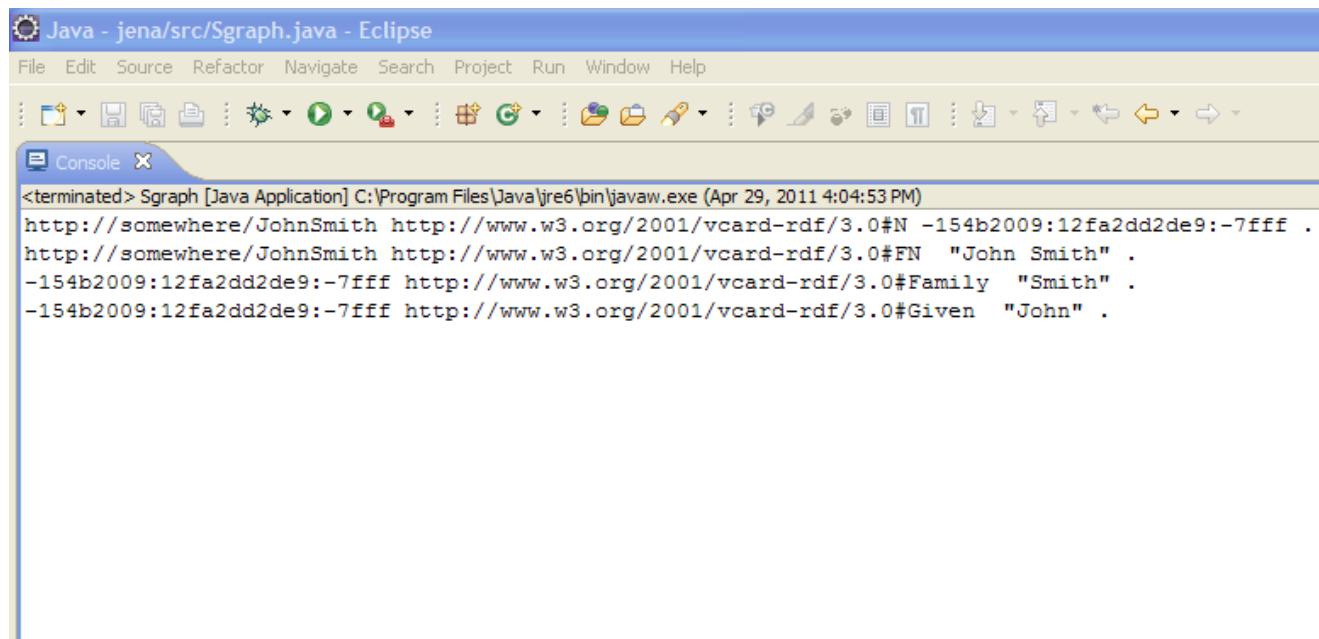
    System.out.println(" .");

}

}

}

```



The screenshot shows the Eclipse IDE interface with the title bar "Java - jena/src/Sgraph.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. The main area is titled "Console" and contains the following RDF triples:

```

<terminated> Sgraph [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 4:04:53 PM)
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N -154b2009:12fa2dd2de9:-7fff .
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN "John Smith" .
-154b2009:12fa2dd2de9:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family "Smith" .
-154b2009:12fa2dd2de9:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given "John" .

```

#### 7.validator checks the correctness of the RDF formats.

```

import java.util.Iterator;
import com.hp.hpl.jena.rdf.model.InfModel;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.reasoner.ValidityReport;
import com.hp.hpl.jena.util.FileManager;

public class Validator
{
    public static void main(String args[])
    {
        Model data = FileManager.get().loadModel("student.rdf");
        InfModel infmodel = ModelFactory.createRDFSModel(data);
        ValidityReport validity = infmodel.validate();
    }
}

```

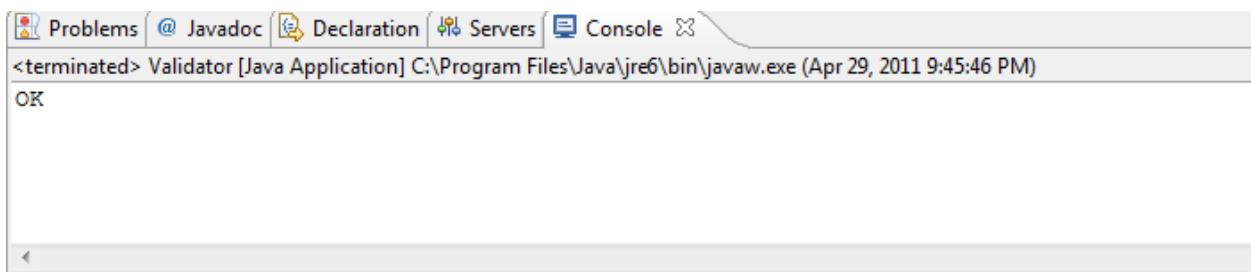
```

        if (validity.isValid()) {
            System.out.println("OK");
        } else {
            System.out.println("Conflicts");
            for (Iterator i = validity.getReports(); i.hasNext(); ) {
                System.out.println(" - " + i.next());
            }
        }
    }
}

```

Output:

Correct--



//8. Example implementing the reasoner in jena

```

import com.hp.hpl.jena.rdf.model.InfModel;

import com.hp.hpl.jena.reasoner.Reasoner;

import com.hp.hpl.jena.rdf.model.Model;

import com.hp.hpl.jena.rdf.model.ModelFactory;

import com.hp.hpl.jena.rdf.model.Property;

import com.hp.hpl.jena.rdf.model.Resource;

import com.hp.hpl.jena.rdf.model.Statement;

import com.hp.hpl.jena.rdf.model StmtIterator;

import com.hp.hpl.jena.reasoner.ReasonerRegistry;

import com.hp.hpl.jena.util.FileManager;

```

```
import com.hp.hpl.jena.util.PrintUtil;

import com.hp.hpl.jena.vocabulary.RDF;

public class SchemaDemo {

    public static void main(String args[])
    {

        Model schema = FileManager.get().loadModel("owlDemoSchema.owl");

        Model data = FileManager.get().loadModel("owlDemoData.rdf");

        Reasoner reasoner = ReasonerRegistry.getOWLReasoner();

        reasoner = reasoner.bindSchema(schema);

        InfModel infmodel = ModelFactory.createInfModel(reasoner, data);

        Resource nForce = infmodel.getResource("urn:x-hp:eg/nForce");

        System.out.println("nForce *:");

        printStatements(infmodel, nForce, null, null);

        Resource gamingComputer = infmodel.getResource("urn:x-hp:eg/GamingComputer");

        Resource whiteBox = infmodel.getResource("urn:x-hp:eg/whiteBoxZX");

        if (infmodel.contains(whiteBox, RDF.type, gamingComputer)) {

            System.out.println("White box recognized as gaming computer");

        } else {

            System.out.println("Failed to recognize white box correctly");

        }

    }

}
```

```

}

public static void printStatements(Model m, Resource s, Property p, Resource o) {

for (StmtIterator i = m.listStatements(s,p,o); i.hasNext(); ) {

    Statement stmt = i.nextStatement();

    System.out.println(" - " + PrintUtil.print(stmt));

}

}

}

```

Output:

The screenshot shows a Java application window with the following content in the console:

```

Problems @ Javadoc Declaration Servers Console 
<terminated> SchemaDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Apr 29, 2011 9:52:48 PM)
nForce *:
- (eg:nForce owl:sameAs eg:nForce)
- (eg:nForce owl:sameAs eg:unknownMB)
- (eg:nForce eg:hasGraphics eg:gamingGraphics)
- (eg:nForce rdf:type owl:Thing)
- (eg:nForce rdf:type rdfs:Resource)
- (eg:nForce rdf:type eg:MotherBoard)
- (eg:nForce rdf:type 2aad267e:12fa41bb42e:-7ffb)
- (eg:nForce eg:hasComponent eg:gamingGraphics)
White box recognized as gaming computer

```