Eui Hyeon Kang
Stian Teien

IBO - 3
Pôle Universitaire Léonard de Vinci
Spring 2020

REPORT - BICYCLE STATIONS

web data mining & semantics

1 Abstract:

The task given was creating an user interface (UI) showing status of bike stations in various cities. Information should be mined from the web and converted to RDF for generalization of the data structure. The data is gathered with JSON-LD, then edited with convert_json.py, then used an online converter to RDF. This RDF is read with JENA and used in an UI made with Swing and AWI in Java.

2 Introduction:

This report describes how we used gathered data from an API as JSON-LD, and convert the data into RDF. From there we used JENA with SPARQL query to use and read our RDF. The UI is made with java, so unfortainly it's not able to see the RDFa with HTML.

The task was to define a vocabulary in Protege, then use one of the given APIs to download JSON-LD with all data. This should be converted into RDF triplets and be able to access with SPARQL queries. The query should be shown in an user interface. In the bike program we made it possible to read RDF with SPARQL and display it in a list and the whole output of the query. We used a website to convert JSON-LD to RDF so it's hard to convert live data in to the program.

To use bicycle program you first pick what RDF you want to query. This RDF defines which city we want information from. It's a dropdown menu to the left telling which city we have data from. When clicking "Confirm" you active the query set and it display the results in a table and as raw output on the right side. When picking a city, bicycle program will display a rather small image over the city. In bicycle program it's possible to open query.txt to edit the desire query you want to display in the table.

Programs you need to have in order to run the bicycle program is java. In order to add new data as RDF you need to get the information with an API, use a small python program to modify the data a little, then use an online converter from JSON-LD to RDF. Then save the newly made RDF file to the right place under data and cities in the folder structure and you're ready to go.

Here is a list over all files you need to run the program:

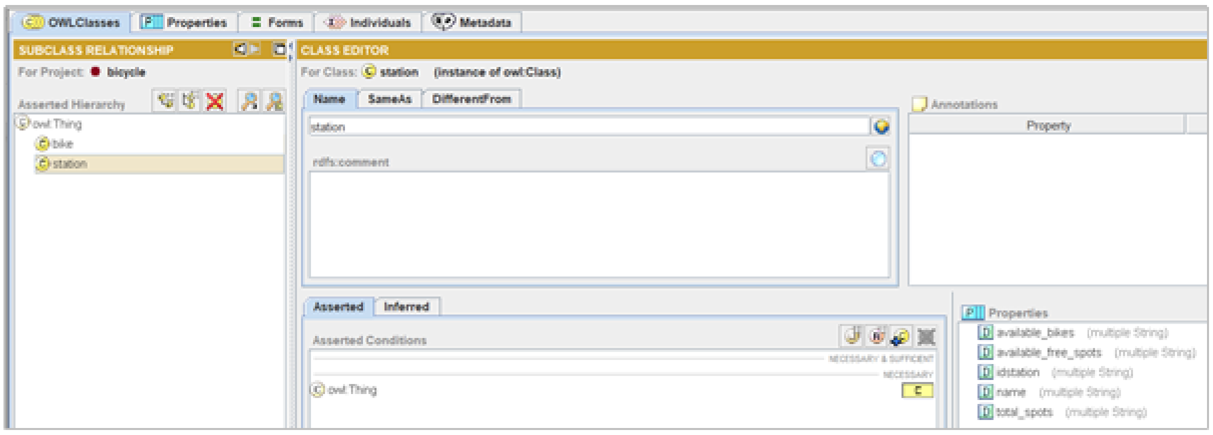
Program name	**Made in/runs in**	**Main purpose**	**Where in the process**
gui_bike.java	Java	Runs UI	Main program
Main.java	Java	Collects Query and return results	Main program
JenaEngine.java	Java	Runs Jena to read RDF	Main program
FileTool.java	Java	Tools for Jena	Main program
query.txt	txt/Java	Query using SPARQL	Main program
bicycle.owl	owl/Java	Vocabulary	Main program
rules.txt	txt/Java	Rules for SPARQL	Main program
owlrules.txt	txt/Java	Owlrules for SPARQL	Main program
lyon_bikes.rdf rennes_bikes.rdf	RDF/Java	Data in RDF format	Main program
http://www.easyrdf.org/converter	website	Convert JSON-LD to RDF	modify data
convert_json_lyon.py convert_json_rennes.py	Python	Modify data in right format	modify data
Bike API [websites]	website	Get data from API	Gather data

3 Method:

To describe the method used for getting bicycle program to work it will be splitted into 6 smaller parts. They all describe a small and important step for getting data ready for UI. The steps also include screenshots of code and processes.

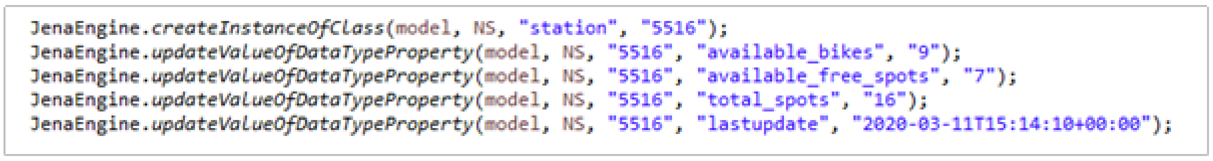
3.1 Making a vocabulary

To make a vocabulary we used an .owl file made with Protege 3.0. In Protege it's possible to define classes with properties. In this program we made two classes; "bikes" and "station". The plan was to order each bike in the bike class and every station in the station class. Since the information just contained station information we mainly focused on station class. This class got a series of dataframe properties like name, avaiable_bikes and total_spots. Here is an image of how it looks like in Protege 3.0:



Protege 3.0

Protege created a bicycle.owl we used as vocabulary. With loading this it was possible to add data with JenaEngine. This would be perfect if we could control from API to UI just with Java. Here is a snapshot from how it was possible to add data with vocabulary in Java with Jena. Unfortunately we could not manage to that.



```
JenaEngine.createInstanceOfClass(model, NS, "station", "5516");
JenaEngine.updateValueOfDataTypeProperty(model, NS, "5516", "available_bikes", "9");
JenaEngine.updateValueOfDataTypeProperty(model, NS, "5516", "available_free_spots", "7");
JenaEngine.updateValueOfDataTypeProperty(model, NS, "5516", "total_spots", "16");
JenaEngine.updateValueOfDataTypeProperty(model, NS, "5516", "lastupdate", "2020-03-11T15:14:10+00:00");
```

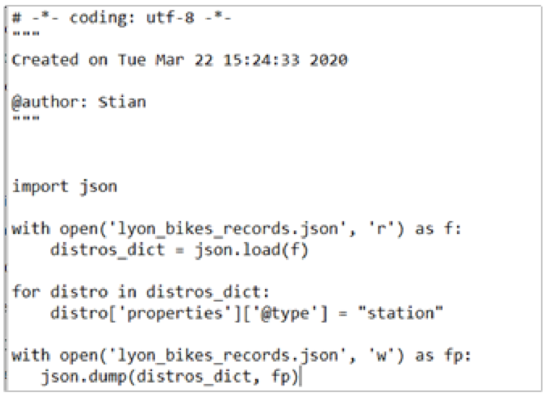
Code to add data with vocabulary

3.2 Data mining - JSON-LD

We used a few of the API from the project description to get data. This data was given as a JSON-LD and looks for the naked eye like a chaos, but JSON actually contains a large amount of somewhat good structured data. When opening the API we copied the information easily to a JSON file and saved it in our space for editing.

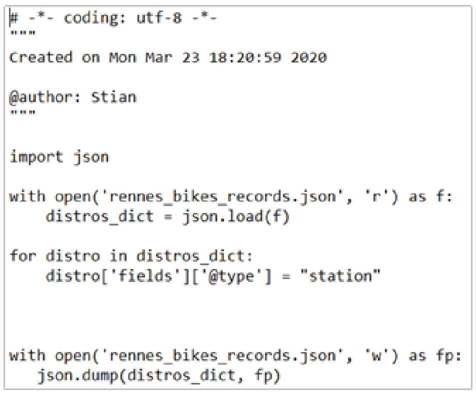
3.3 Modify JSON-LD

For making RDF out of our JSON-LD we needed to specify which information we need to extract and drop. To correspond to our own vocabulary we needed to rename some information and make each individual station belong to the station class. This was done with edit the values we want in top of json before converting and to specify every station to the station class we had to add ‘@type’: ‘station’ in all individual. This was not something we would have done manually, so a python script is used to convert it. The python script had to be a little different from every API we used because the data came out a little different. Here is a snapshot from convert_json_lyon.py and convert_json_rennes.py:



```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 22 15:24:33 2020

@author: Stian
"""


import json

with open('lyon_bikes_records.json', 'r') as f:
    distros_dict = json.load(f)

for distro in distros_dict:
    distro['properties']['@type'] = "station"

with open('lyon_bikes_records.json', 'w') as fp:
   json.dump(distros_dict, fp)
```

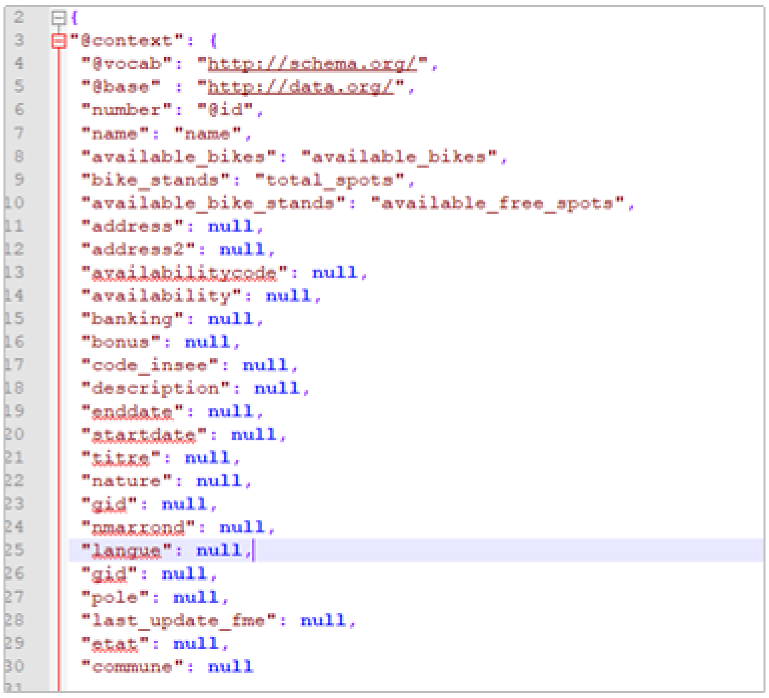
convert_json_lyon.py



```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 23 18:20:59 2020

@author: Stian
"""

import json

with open('rennes_bikes_records.json', 'r') as f:
    distros_dict = json.load(f)

for distro in distros_dict:
    distro['fields']['@type'] = "station"


with open('rennes_bikes_records.json', 'w') as fp:
   json.dump(distros_dict, fp)
```

convert_json_rennes.py

Now as every individual station would be a station class we need to find out what information we needed from the JSON-LD. Because it contains a lot of information! We used context and JSON syntax for RDF. This is what we added before converting to filter out data:

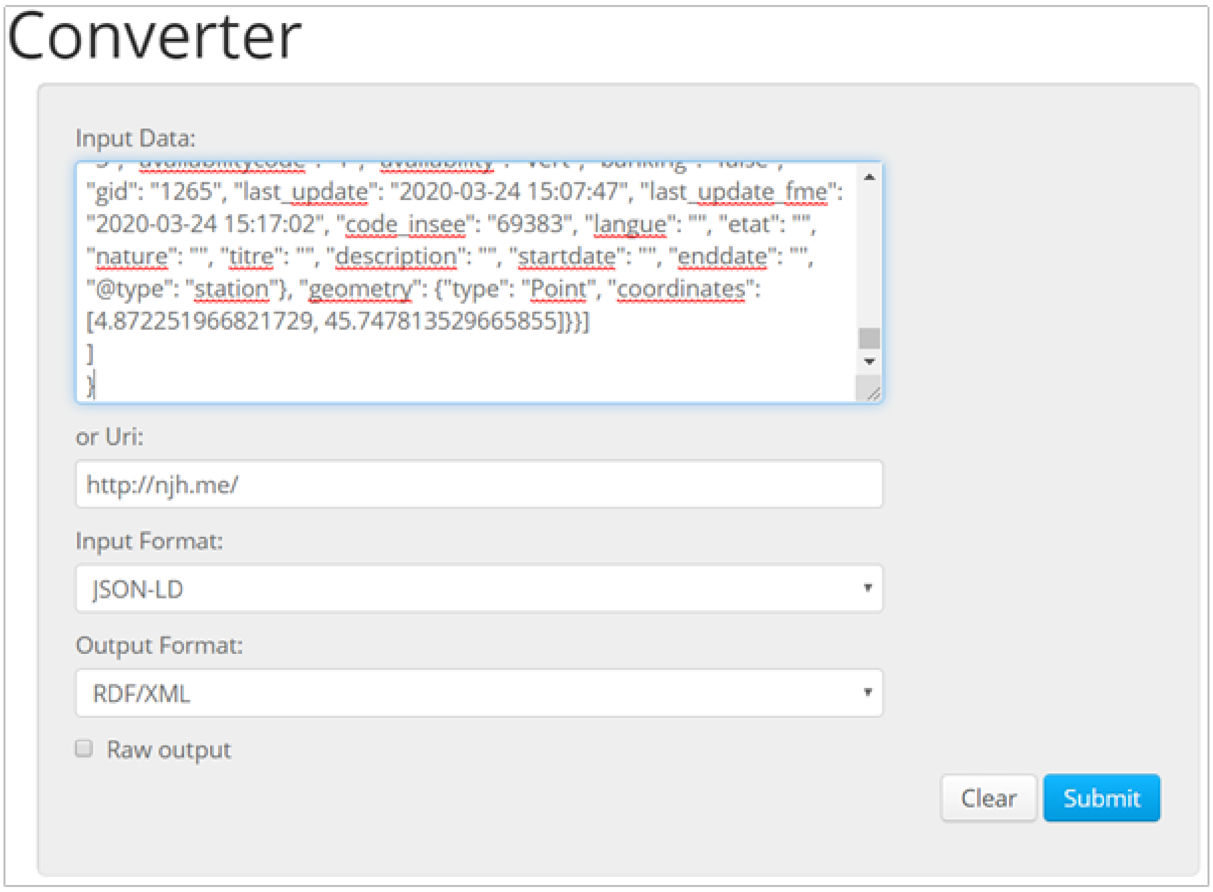


```
{
"@context": {
 "@vocab": "http://schema.org/",
 "@base" : "http://data.org/",
 "number": "@id",
 "name": "name",
 "available_bikes": "available_bikes",
 "bike_stands": "total_spots",
 "available_bike_stands": "available_free_spots",
 "address": null,
 "address2": null,
 "availabilitycode": null,
 "availability": null,
 "banking": null,
 "bonus": null,
 "code_insee": null,
 "description": null,
 "enddate": null,
 "startdate": null,
 "titre": null,
 "nature": null,
 "gid": null,
 "nmarrond": null,
 "langue": null,
 "gid": null,
 "pole": null,
 "last_update_fme": null,
 "etat": null,
 "commune": null
```

JSON-LD before converting to RDF

3.4 Convert JSON-LD to RDF

Converting the JSON-LD to RDF was an easy operation once the context was right. By using a website for converter called http://www.easyrdf.org/converter we could simply insert the whole JSON-LD in the input window, pick what we wanted as output and press “Submit”.


Converter

Input Data:

```
"gid": "1265", "last_update": "2020-03-24 15:07:47", "last_update_fme":
"2020-03-24 15:17:02", "code_insee": "69383", "langue": "", "etat": "",
"nature": "", "titre": "", "description": "", "startdate": "", "enddate": "",
"@type": "station"}, "geometry": {"type": "Point", "coordinates":
[4.872251966821729, 45.747813529665855]}}]
]
}
```

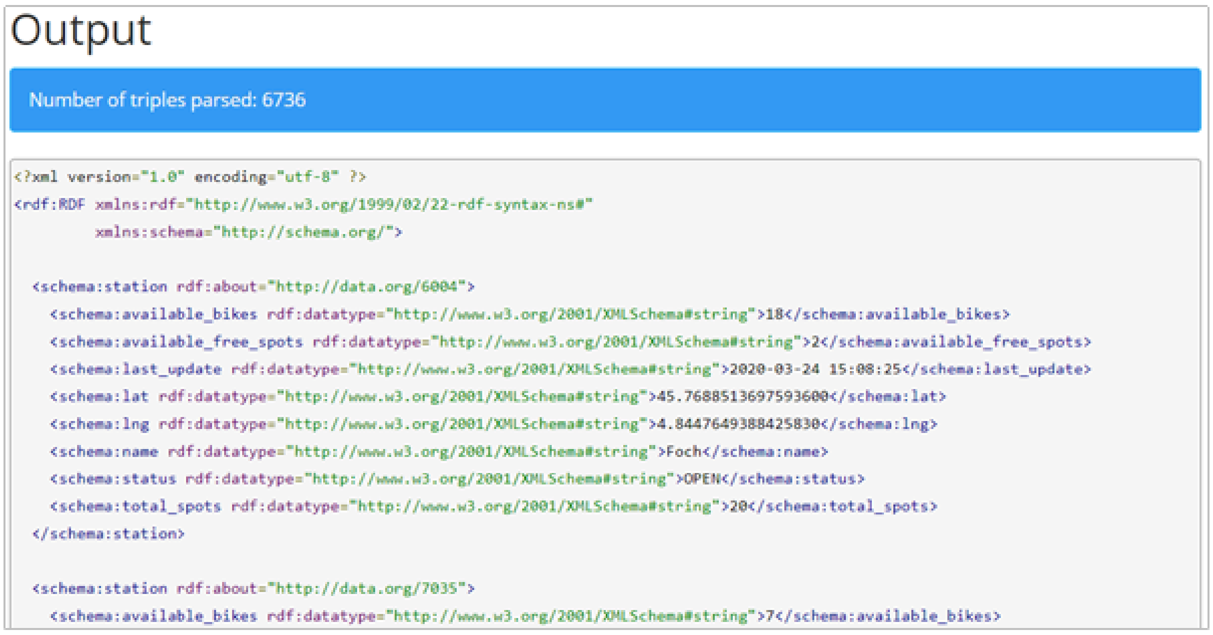
or Uri: http://njh.me/

Input Format: JSON-LD

Output Format: RDF/XML

Convert JSON->RDF website

When hitting Submit conversion was done in a second and we got a whole lot data now in RDF format to copy. This is how the new RDF data looks like:



Output

Number of triples parsed: 6736

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:schema="http://schema.org/">

  <schema:station rdf:about="http://data.org/6004">
    <schema:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#string">18</schema:available_bikes>
    <schema:available_free_spots rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2</schema:available_free_spots>
    <schema:last_update rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2020-03-24 15:08:25</schema:last_update>
    <schema:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#string">45.7688513697593600</schema:lat>
    <schema:lng rdf:datatype="http://www.w3.org/2001/XMLSchema#string">4.8447649388425830</schema:lng>
    <schema:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Foch</schema:name>
    <schema:status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">OPEN</schema:status>
    <schema:total_spots rdf:datatype="http://www.w3.org/2001/XMLSchema#string">20</schema:total_spots>
  </schema:station>

  <schema:station rdf:about="http://data.org/7035">
    <schema:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#string">7</schema:available_bikes>
```

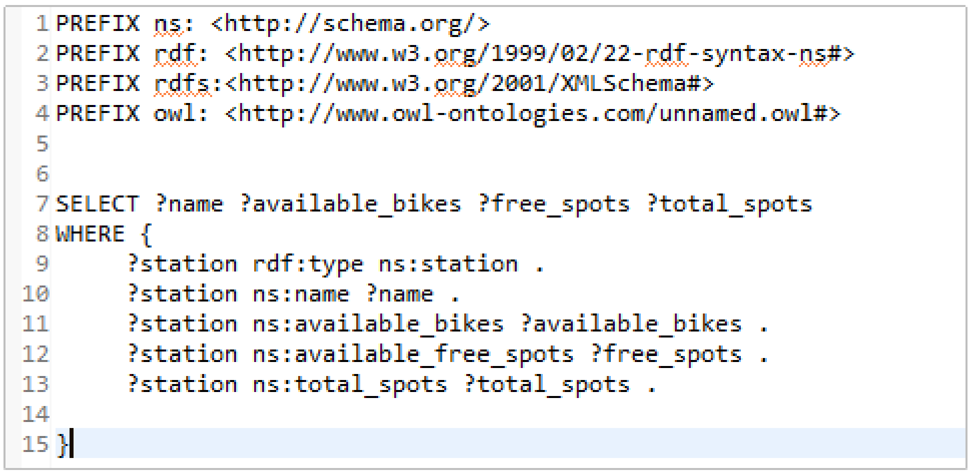
Output RDF/XML

If the file was real big like this one it was possible to get in raw format for easier copying. This was save to lyon_bikes.rdf for later usage!

3.5 RDF in JENA

In bicycle program we used JenaEngine with query written in SPARQL to look up data from the RDF. This was done by having a file namely query.txt where the query is written. We have added every individual to the class station meaning the that every individual has rdf:type → ns:station. This makes life a whole lot easier and we can simply find all station by query: *?station rdf:type ns:station .*

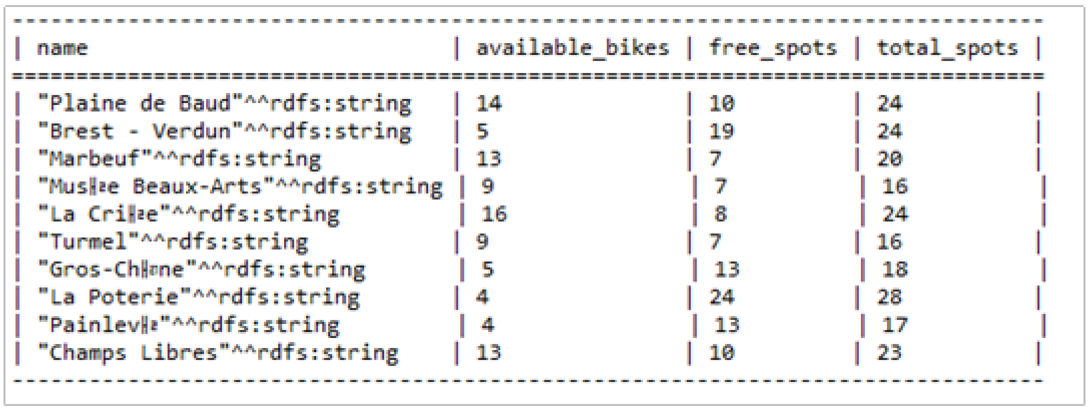
We make a little more interesting by adding some more in the query and it ends up looking like this:



```
PREFIX ns: <http://schema.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.owl-ontologies.com/unnamed.owl#>


SELECT ?name ?available_bikes ?free_spots ?total_spots
WHERE {
     ?station rdf:type ns:station .
     ?station ns:name ?name .
     ?station ns:available_bikes ?available_bikes .
     ?station ns:available_free_spots ?free_spots .
     ?station ns:total_spots ?total_spots .

}
```

query.txt

And the output looks like this when running Jena! By having some prefixes we can shorten down what is in the output.

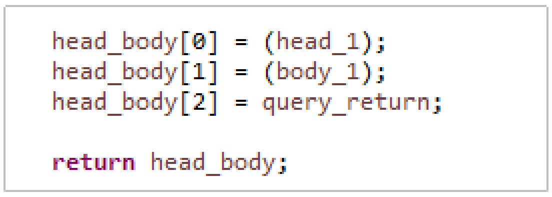


```
--------------------------------------------------------------------------------
| name                             | available_bikes | free_spots | total_spots |
================================================================================
| "Plaine de Baud"^^rdfs:string    | 14              | 10         | 24          |
| "Brest - Verdun"^^rdfs:string    | 5               | 19         | 24          |
| "Marbeuf"^^rdfs:string           | 13              | 7          | 20          |
| "Mus├⌐e Beaux-Arts"^^rdfs:string | 9               | 7          | 16          |
| "La Cri├⌐e"^^rdfs:string         | 16              | 8          | 24          |
| "Turmel"^^rdfs:string            | 9               | 7          | 16          |
| "Gros-Ch├¬ne"^^rdfs:string       | 5               | 13         | 18          |
| "La Poterie"^^rdfs:string        | 4               | 24         | 28          |
| "Painlev├⌐"^^rdfs:string         | 4               | 13         | 17          |
| "Champs Libres"^^rdfs:string     | 13              | 10         | 23          |
--------------------------------------------------------------------------------
```

Output using Jena

3.6 From JENA to UI

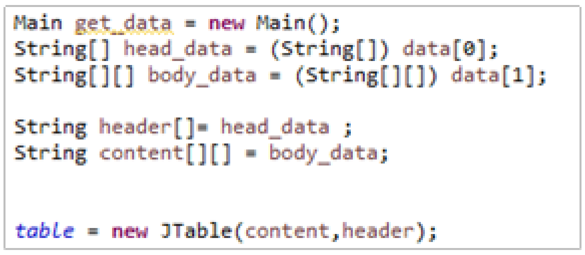
Getting from Jena to our UI we had to program our way there. The first challenge was to gather the data from Jena output. This we did with a some loops and regex into ArrayList, but then later converted the ArrayLists into normal String[] lists. By doing this we could transport the information as a multiple dimensional list with headers in 1 dimension and body in 2 dimension. Additional we added the raw Jena output to print it later in our UI.



```
head_body[0] = (head_1);
head_body[1] = (body_1);
head_body[2] = query_return;

return head_body;
```

return from Main.java

When we return from Main.java in the right dataformat it was easy to add data to our table and Jena-output pane.

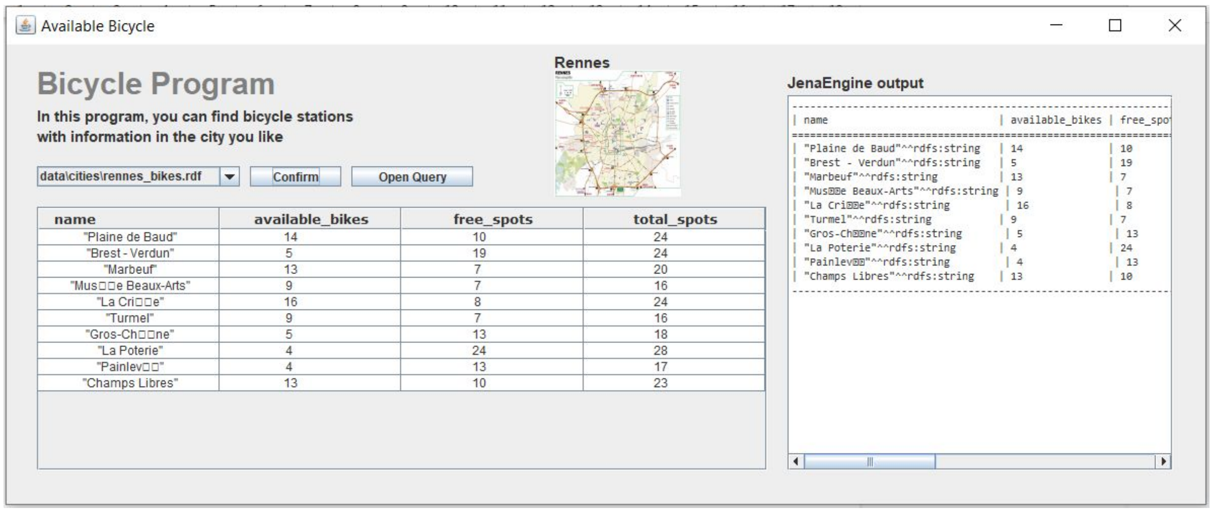


```
Main get_data = new Main();
String[] head_data = (String[]) data[0];
String[][] body_data = (String[][]) data[1];

String header[]= head_data ;
String content[][] = body_data;


table = new JTable(content,header);
```

Getting right dataformat from Jena

4 Results:

The results of the project came together in the Bicycle programs UI. In the UI we can read available files in a folder, pick the desired one and display it on a table. Here is an image on how bicycle program looks:



In this program, you can find bicycle stations with information in the city you like

name	available_bikes	free_spots	total_spots
"Plaine de Baud"	14	10	24
"Brest - Verdun"	5	19	24
"Marbeuf"	13	7	20
"Mus□□e Beaux-Arts"	9	7	16
"La Cri□□e"	16	8	24
"Turmel"	9	7	16
"Gros-Ch□□ne"	5	13	18
"La Poterie"	4	24	28
"Painlev□□"	4	13	17
"Champs Libres"	13	10	23

```
-----------------------------------------------------------------
| name                             | available_bikes | free_spo
=================================================================
| "Plaine de Baud"^^rdfs:string    | 14              | 10
| "Brest - Verdun"^^rdfs:string    | 5               | 19
| "Marbeuf"^^rdfs:string           | 13              | 7
| "Mus⊠⊠e Beaux-Arts"^^rdfs:string | 9               | 7
| "La Cri⊠⊠e"^^rdfs:string         | 16              | 8
| "Turmel"^^rdfs:string            | 9               | 7
| "Gros-Ch⊠⊠ne"^^rdfs:string       | 5               | 13
| "La Poterie"^^rdfs:string        | 4               | 24
| "Painlev⊠⊠"^^rdfs:string         | 4               | 13
| "Champs Libres"^^rdfs:string     | 13              | 10
-----------------------------------------------------------------
```

Bicycle program GUI

Here it is possible to pick the right data from the dropdown menu and press confirm to execute the order. By pressing Open Query it's possible to open the query edit or watch what it does. The city name and city map will appear when data is inserted. As well as JenaEnigne output is shown in raw format to the right.

We can conclude that we solved the task. If we could do something different it would be making it live with connection of API, and using a HTML instead of GUI from Java.