Exercise - Data Formats

Robert Meusel
University of Mannheim, Germany

September, 2018

In the following three exercise sections you will be asked to perform basic parsing and extraction tasks with Java. Each subsection is dedicated to one of the three formats: XML, JSON and RDF. The tasks are rather basic and the goal is to refresh your knowledge in Java in general and in particular in parsing those formats.

1 XML

This subsection is dedicated to the XML format. In particular you are asked to perform XPath queries on the *Mondial* dataset[1]. This dataset includes world geographic information integrated from the *CIA World Factbook*, the *International Atlas* and the *TERRA database*, to name just the pre-dominant sources. Please inspect the document manually (using a text editor) in order to explore the structure. You can also have a look at the *w3school XPath tutorial*[2] to solve the following tasks.

1.1 Mondial — `Read XML`

TASK 1: In order to get started with automatically parsing the XML file, write a Java class which reads the file using the *JAXP* library and prints the the root node of the XML document. (Hint: Have a look at the lecture slides in order to get an idea how to start.)

1.2 Mondial — `Schema Inspection`

Now that we have written our parser and explored the root node we can start digging deeper into the XML file.

TASK 2: Adapt the class from the previous task that a unique list of all

[1] The file can be downloaded from ILIAS but is also available here: `http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/data/mondial/mondial-3.0.xml`

[2] `https://www.w3schools.com/xml/xpath_intro.asp`

nodes below the root node is printed.

1.3 Mondial — `Basic XPath`

Now that we got an idea about the structure of the XML we are interested in the content.

TASK 3: Adapt the solution of the previous task in the way that it prints the names of all countries which belong to the continent with the name `Europe`. (Hint: Have a look at the schema of the node `country` to see how it is linked to the continent.)

1.4 Mondial — `XPath Predicates I`

With the solution of the previous task we are now able to get the countries for a selected continent. In a next step we want to extend this query that we can get countries which belong to two continents.

TASK 4: Extend the XPath for the former task in order to retrieve only countries which are part of `Europe` and `Asia`.

1.5 Mondial — `XPath Predicates II`

In a final step we want to gather all attributes from a selection of nodes, without explicitly knowing their names.

TASK 5: Extend the solution of the former task in order to navigate (using XPath) to the `country` node and print all attribute names and values. (Hint: You can use the `getAttributes()` method to detect all available attributes of the current node).

2 JSON

In the second part of this exercise we focus on the JSON format. As you already have some experience with the *Mondial* dataset in a first step, you are asked to transform parts of the XML into a JSON. In order to do so make use of the `Google Gson` Java library[3].

[3] You can find the library at the google code page: `https://sites.google.com/site/gson/`. A user guide can be found on this page: `https://github.com/google/gson/blob/master/`

2.1 Mondial — `XML to JSON`

TASK 6: Create a JSON file (*.json) which contains all countries which are located in Europe with the attributes of the `country` node from the original `mondial-*.xml`. (Hint: Have a look at the last exercise of the former section. `Gson` offers a method to simply translate a `HashMap` into a JSON string, which than can be written to a file.)

2.2 Mondial — `Reading JSON`

In a second step we want to create Java objects from the JSON file we just created, but we are not interested in all attributes.

TASK 7: Write a small program, which reads the JSON file (which was the output of the former task) and transforms each line into a Java object (named `Country.java`). The country should have four values: the `id` (String), the `name` (String), the `car_code` (String), and the `population` (Long). Do you have to pay attention to type conversion? What is the total number of inhabitants of those countries? (Hint: Have a look in the example code of the lecture.)

3 RDF

In the last part of this exercise session we will focus on RDF and SPARQL. In ILIAS you can find the European countries with their name, population and the spoken languages stored as RDF file. The file was generated from the original *mondial* XML file.[4] In the following you will be asked to formulate SPARQL queries to answer questions about the dataset using the *Jena* Java Framework[5]. In addition to the lecture the *W3* site of *SPARQL Query Language* can help you to answer the questions.[6]

3.1 Mondial — `Query with SPARQL I`

TASK 8: Write a small program, which reads the RDF file (from ILIAS) and formulate a SPARQL query which returns the name and id of all countries within the dataset ordered by the name. What is the last country in this list. In order to explore the property names and namespaces have a look at the RDF file or at the code which was used to generate the file. (Hint: Have a look at the example code of the lecture.)

`UserGuide.md`

[4] The code which was used to generate the file can also be found in the Java project of this (see `de.dwslab.lecture.wdi.rdf.Converter.java`).

[5] The documentation of the framework can be found at their website: `https://jena.apache.org/`

[6] `https://www.w3.org/TR/rdf-sparql-query/`

3.2 Mondial — `Query with SPARQL II`

As we now have setup the code to query against our dataset, we are interested in the largest countries. But as we already know that Russia and Germany are pretty large, we want to generate a list of the second top 5 largest countries by population.

TASK 9: What is the SPARQL query which returns the second five (6^{th} to 10^{th}) most populated countries in Europe? And which countries are these?

3.3 Mondial — `Query with SPARQL III`

In a last exercise you are asked to write a SPARQL query which selects all countries whose inhabitants speak a defined language.

TASK 10: How does the SPARQL query look like, which returns a list of all German-speaking countries with their name and id?