Exercise Sheet 2 (SS 2020)

3.0 VU Semistrukturierte Daten

General Information

For the second exercise sheet, you will access the XML document you designed as part of the first exercise sheet, and access it via various APIs and query languages. Specifically, you will create an HTML overview page via XSLT, evaluate a query via XQuery, and lastly, access and modify the XML file via the Java APIs SAX and DOM.

Template. As a framework for this exercise sheet, you will find on TUWEL a zip archive with a basic project structure and resources which you should use as a template for this exercise sheet. Furthermore, within this framework you also have an ant script (build.xml), which simplifies the testing and submission of the exercise sheet.

Please note:

- Copy your files `art-xsd.xml` and `art.xsd`, from your solution for the first exercise sheet, into the path `resources`.
- All mentioned files and paths of this exercise sheet refer to the provided template.
- Precise instructions on how to use the ant targets are provided in the individual exercises.
- You *need* to use the template for solving this exercise sheet.
- To ensure that your solutions run our system, test them on Java 8.

The exercise sheet contains 3 exercises, for which you can receive 15 points in total.

Submission

A submission-ready archive `ssd-exercise2-ss20.zip` is produced via the command: `ant zip`

Deadline

at the latest June 5th	**23:55**	Upload your submission on TUWEL
Please do not forget!	$\Longrightarrow$	Register for an exercise interview in TUWEL

Exercise Interviews

In the solution discussion, the correctness of your solution as well as your understanding of the underlying concepts will be assessed. The scoring of your submission is primarily based on your performance at the solution discussion. Therefore it is possible (in extreme cases) to get 0 points even though the submitted solution was technically correct. Please be punctual for your solution discussion. Otherwise we cannot guarantee that your full solution can be graded in your assigned time slot. Remember to bring your student id to the solution discussion. *It is not possible to score your solution without an id.*

Q&A via TUChat (optional)

We offer Q&A chats as personal support, hosted on TUChat. The aim is to help understand the material, not solve your exercises for you. All times and links are provided on TUWEL.

Further Questions – TUWEL Forum

For any further questions, regarding organisation or the material, use the TUWEL forum.

Exercises

Exercise 1 (XSLT) *[6 Points]*

Create an XSLT document `src/tags-overview.xsl` that transforms a valid art XML document (`art-xsd.xml`), where validity is defined against the XSD schema `art.xsd`, into an HTML document. This HTML document should provide an overview over all tags.
To help you with the format, we provide you a draft in the document `tags-overview.xsl`. Create within this draft the following XSLT templates:

- A named template `info`, which takes 2 parameters, *mode* and *tagname* and outputs the following:
  - It checks whether *mode* is set to "artist", if so, it will then check if there are any artists matching one of two conditions: 1) this `artist` element has a `t` element, directly matching *tagname*, or 2) there exists an `object` element, which has a `t` element, matching *tagname and which has a* `by` *element matching the artists name*. Informally: we want to capture all artists who are directly tagged with *tagname* or who created some object, which is tagged with *tagname*. If and only if, any such artists are found, it creates a table, with the header "Artists".
  - If *mode* is set to anything else (or empty), this named template will check for any objects matching one of two conditions: 1) this `object` element has a `t` element, directly matching *tagname*, or 2) there exists an `artist` element, which has a `t` element, matching *tagname and which has a* `name` *element matching with a* by *element of this object*. Informally: we want to capture all objects directly tagged with *tagname*, or where the artist who created them is tagged with *tagname*. If and only if, any such objects are found, it creates a table, with the header "Exhibitions".
  - For any artist (or resp. object) matching the conditions, this template will create a cell of a table (`<tr>` and within it `<td>`) and *call its template* to fill the cell, with the last cell containing a summary of the number of artists (or resp. objects) thus matched.
  - The output should be *sorted*. In case "mode" is set to "artist", it should be sorted by the *name* element of the matched artists, in case of objects it should match the first `year` element in their `label` element (if present). For objects without a year, order is undefined.
- A template for `art` Elements, which calls the templates of all `tag` child nodes.
  Note: This is already provided to you in the draft.
- A template for `tag` Elements, which prints as a header the name of the tag (`tagname` attribute), and then a line below it a description (content of the element). If the content of the element is empty, output as the description: `"No further information on"`, followed by the `tagname`.
  Note: It is enough for full points, to check the content against the empty string. You are not required to filter out whitespaces, though it is left as an additional challenge to properly ignore spaces, newlines and tabs here. Lastly, this template shall create a table with single row (`<tr>`) and then call the named template `info` exactly twice: Once with *mode* set to "artist", and again with *mode* set to "objects" (or left empty). Both times the *tagname* parameter shall be set to the value of the `tagname` attribute.
- A template for `artist` Elements, outputting the name, and then use the data from their `lived` element to output, in parenthesis, their birth date (`from` attribute) to their day of death (`to` attribute, if present), and their place of birth (`birthplace` attribute).

- A template for `object` Elements, outputting their title (`title`) and in upper-case letter their type (`kind`), either sculpture or painting . Then below their label (`label`). Be sure to properly print *all* content in the label. Then below that, it should output "Created By", and within an unnumbered list (`<ul>`) the name of its creator. In case the `by` element matches the `name` element of an existing artist, it should call its template. If no such artist exists in the XML file, then it should output the name followed by "(Unregistered Artist)".

Note: An example output is provided under `resources/tags-overview.html`.

Execution: Run the command `ant run-xslt`. It will use your stylesheet to create an HTML document `tags-overview.html` in the directory `output`. You can use a browser to open it.

Please note: A correct output in `output/tags-overview.html` is required to receive all points for this exercise. For a syntactically incorrect stylesheet, you will receive 0 points!

Your submission will consist of:

- An XSLT document: `src/tags-overview.xsl`

Exercise 2 (XQuery) *[3 Points]*

Create an XQuery `src/xquery.xq`, which is to receive as input an XML file, which is valid against the `art.xsd` schema from the first exercise. It shall output all artists who have created more than 3 objects within the same year (having the same `year` child node in their `label` elements). Objects with multiple year elements in their labels shall be counted towards each year. The name of the artist shall appear as an attribute `name`, as well as the year (`year`). The output should be sorted by the birth date (the string in `from`) of the artists. For these artists and the qualifying years, a count of *all* objects they created in that year should be outputted, as well as exactly 3 objects of the corresponding year, where only the `title` of these objects needs to be output. If there are multiple objects in the same year, your query just needs to output three objects from it. **Note:** Avoid duplicate year entries (i.e. covering for the same artist the same year more than once).

An example output:

```
<busyYearsForArtists>
  <artist name="Pablo Picasso" year="1952">
      <objectCount>21</objectCount>
      <object>Portrait of Madame Helene Parmelin</object>
      <object>Flying Dove (in the Arc-en-ciel)</object>
      <object>Mediteranean Landscape</object>
  </artist>
</busyYearsForArtists>
```

Be sure that your query `xquery.xq` can be run via the command `ant run-xquery`. The output of your query, located in `src/xquery.xq`, will be saved in the XML document `output/xquery-out.xml`.

Please note: A correct output in `output/xquery-out.xml` is required to receive all points. You will receive 0 points for submitting a syntactically incorrect XQuery!

Your submission will consist of:

- An XQuery: `src/xquery.xq`

Exercise 3 (DOM/SAX) *[6 Points]*

The aim of this exercise is to parse the the `art-xsd.xml` document via DOM, and then use SAX to parse an `exhibition.xml` document and apply certain changes to the `art-xsd.xml` document via the information in the `exhibition.xml` document, i.e., to include new objects, and remove indicated old ones. The `exhibition.xml` document has the following structure:

- The root is an `exhibition` Element.
- It is followed by an arbitrary amount of `object` elements. They have as child nodes a `title`, a `type` element (containing as content either "painting" or "sculpture"), where the `type` element can have optional attributes for paint and used surface, a list of `tag` elements, which are text nodes, a `creator` element, stating the name of the artist who created it, a `description` element, and a `year` element, indicating the creation year.
- Furthermore, there is a `lending` element, containing a number of `catalog` child nodes.

Note: An example for an `exhibition` document can be found under `resources/exhibition.xml`.

The goal is twofold: 1.) integrate the new objects into the art XML document, *while maintaining the validity of the schema!* Be sure to create a new, valid catalog and use it for all these objects. Also be sure to maintain any relevant key constraints, and 2.) remove the objects with `catalog` element matching those occurring in the `lending` element.

Hint: To properly test the removal of objects, change the provided `exhibition` document to refer to a catalog value actually occurring in your XML document.

Description of Classes
The template provides two classes. The class SSD provides the actual logic for executing the program. The class ArtHandler provides a SAX handler, which parses the `exhibition.xml` document and modifies a `art-xsd.xml` document.
Here is a detailed description of the classes:

- Class: SSD
  - *Variables*:
    - `static DocumentBuilderFactory documentBuilderFactory`
    - `static DocumentBuilder documentBuilder`
  - *Methods*:
    - `static void main(String [] args) throws Exception`: Entry point of the program. Parses the command line arguments and calls the methods `initialize` and `transform`.
    - `static void initialize() throws Exception`: Initialises the `documentBuilderFactory` and the `documentBuilder` variables.
    - `static void transform(String inputPath, String exhibitionPath, String outputPath) throws Exception`: You need to implement this method. First you need to create a DOM object (referred to as "Document") from the file name, provided by the `inputPath`. Then you need to create the SAX parser and initialise it to parse the document from the path in the `exhibitionPath` variable. For this purpose, you should create an instance of the ArtHandler class, which will need the above defined "Document" object as an argument in its constructor. Now parse the `exhibition.xml`.

The ArtHandler will change the document. The final result should be called via the method `getDocument()` from the class ArtHandler and validated against the schema. Finally, this output should be saved in the path specified by the variable `outputPath`.

  - `static void exit(String message)`: This method can be used to emit an error message and exit the program.

- Class: ArtHandler
  - *Variables*:
    - `static XPath xPath`: this XPath instance can be used to evaluate XPath queries over an XML file.
    - `Document artDoc`: saves a DOM representation of an `art-xsd.xml` document.
    - `String eleText`: saves the text content of XML elements.
    - Feel free to declare further variables as needed.
  - *Methods*:
    - `ArtHandler(Document doc)`: The constructor has as its argument a DOM document.
    - `void characters(char[] text, int start, int length)`: SAX calls this method to read the text content of an XML element. The value will be saved in the `eleText` variable.
    - `Document getDocument()`: returns the XML document saved in `artDoc`.
    - Define here further methods, to parse the `exhibition.xml` document (e.g.: `startElement`, etc.) and to change the `artDoc` object.

Running the program and output
The command to run the code in `src/ssd/SSD.java` needs three command-line arguments. The first argument is the art document (z.B. `art-xsd.xml` from Exercise Sheet 1). The next is an `exhibition.xml` file (e.g.: `resources/exhibition.xml`). The last argument is the file name of the output (e.g.: `output/art-out.xml`). In the Ant file, there are two preconfigured targets:

- `ant run-dry`: calls the program via the `exhibition.xml` document, and the art document `resources/art-xsd.xml` as input, and saves the output in `output/art-out.xml`.
- `ant run-persistent`: calls the program via the `exhibition.xml` document, and the art document `resources/art-xsd.xml` as input, and saves the output in `resources/art-xsd.xml`, thus permanently changing the XML document.

After the exercise interviews for Exercise Sheet 1, the files `resources/art-sample-xsd.xml` and `resources/art-sample-out.xml` will be added to the template on TUWEL, with the latter being a sample art XML document, created after running the program with the `exhibition.xml` and the former art XML document as input.

Hint: XPath queries over an XML document can be evaluated via the following code snippet:

```
XPathExpression xpathExpr = xPath.compile("//tags");

NodeList tagsList = (NodeList)xpathExpr.evaluate(artDoc, XPathConstants.NODESET);
```

Your submission will consist of:

- Two Java source files: `SSD.java` and `ArtHandler.java`