Exercise Sheet 2 – Summer 2021

3.0 VU Semistructured Data

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. A submission-ready archive `ssd-exercise2-ss21.zip` is produced via the command: `ant zip`. After submitting your file, you need to register for an exercise interview. **You can only receive points for this exercise if you attended the interview.**

Instructions for using the 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.

- Copy your files `vaccination-plan-xsd.xml` and `vaccination-plan.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 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.

This exercise sheet contains 3 exercises, with a total of 15 points achievable.

Deadlines

at the latest, Th. June 10th	**12 noon**	Upload your submission on TUWEL
Do not forget!	$\Longrightarrow$	Register for an exercise interview slot in TUWEL

Exercise Interviews

In the exercise interviews, 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 exercise interview. 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 show your student id on the video**. It is not possible to score your solution without an id.

Changes due to COVID-19

Due to the ongoing situation with COVID-19 we will not offer in-person office hours for the exercise sheets. If you have technical issues, trouble understanding the tasks on this sheet, or other questions please use the TUWEL forum.
We also recommend that you get involved in the forum and actively discuss with your colleagues on the forum. From experience we believe that this helps all parties in the discussion greatly to improve their understanding of the material.

TUWEL Forum

For any further questions, regarding organisation or the material, use the TUWEL forum.
Please do not post your (even partial) solutions on the forum.

Exercise 1 (XSLT) [6 Points]

Create an XSLT document `src/vaccination-plan-overview.xsl` which transforms a valid vaccination-plan XML document (`vaccination-plan-xsd.xml`), where validity is defined against the XSD schema `vaccination-plan.xsd`, into an HTML document. This HTML document should provide an overview over all vaccine types.
We provide you for this exercise a draft in the document `vaccination-plan-overview.xsl`. Create within this draft the following XSLT templates:

- A template for `vaccination-plan` elements, which calls the templates of all `vaccine` nodes which are children of `vaccine-types` elements.
  Note: *This is already provided to you in the draft.*

- A named template `populateTable`, which takes a parameter called *vaccine* and outputs the following:
  - It iterates over all `batch` elements whose parent `vaccine` element has an `type_ref` attribute value matching the one in the parameter.
  - For each such `batch` element, it will create a table and the header of the table will be filled by calling the template of this batch. Below the header there should be a list of all patients which have received doses of this batch. This is done by looking for any `patient` nodes containing a `vaccine` child node which has a `ref_batch` attribute matching the id of the current batch. The output of the patients should be sorted by their birth year, in descending order. For patients without a `birth_year` attribute, order is left undefined.
  - For each such patient, the template will create a cell in the table (`<tr>` and within `<td>`) and *call the named template* `patient` to fill the cell.
  - Finally, the last cell of the table should provide the count of all patients which have received this batch.

- A named template `patient` which has a parameter called *batch* and outputs the following:
  - The named template should assume that the currently selected node is a `patient` element. For this node, it should output the `name` attribute, and the age of the patient. This age should be calculated by taking the `birth_year` attribute and subtracting it from the current year. The XPath 2.0 expression `year-from-date(current-date())` can be used to get the value of the current year.
  - Furthermore, the template should also list all residences of the patient, by outputting the contents of any `main` or `second` child nodes of it. This output should be sorted such that the main residence appears first.
  - The template should output the risk group status of the patient.
  - Finally, it should be listed which doses of the selected batch the patient has received. Two things should be output: whether it is the first or second dose the patient has received and the date of vaccination. You may assume that the appearance of `vaccine` and `vaccine-date` child nodes is always ordered by date, i.e. later vaccine doses are also further down the list of children. Only `vaccine` nodes where the `ref_batch` matches the parameter should be output.

- A template for `vaccine` elements, which prints as a header the name of the vaccine (`name` child node), and then a line below the type of the vaccine (content of the child node `type`). Lastly, this template shall check the value of the child node `authorized` and if it is set to “true”, it shall print the message “This vaccine is authorized” or otherwise the message

"Warning! This vaccine is not yet authorized!". If there are batches of this vaccine (defined as a match between `type_ref` of their parent node and the name of the vaccine), then this template shall call the named template `populateTable` , with the name of this vaccine as the parameter. If no suitable batches exist, then the message "No batches received so far!" should be output instead.

- A template for `batch` elements, outputting the id as a header, and below it the content of the node.

Note: *An example output is provided under* `resources/vaccination-plan-overview.html`.

How to run: Run the command `ant run-xslt`. It will use your stylesheet to create an HTML document `vaccination-plan-overview.html` in the directory `output`. Open it in a browser.

A correct output in `output/vaccination-plan-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/vaccination-plan-overview.xsl`

Exercise 2 (XQuery) [3 Points]

Create an XQuery `src/xquery.xq`, where the input is an XML file, which is valid against the `vaccination-plan.xsd` schema from the first exercise. The output must be all vaccines (child of `vaccine-types`) which were taken by patients from at least two different risk groups. Here "taken by" is defined as there being a `patient` node having a child node `vaccine` with `ref_batch` attribute matching a batch id of a vaccine (child of `vaccines`) with `type_ref` attribute matching the name of the vaccine (child of `vaccine-types`). Patients which have received doses from multiple vaccines should be counted for each of them. The name and the type of the vaccine should appear as attributes `name` and `type`.
Note: *Be sure to count patients from* all *batches of the same vaccine towards the same sum.*

The output should be sorted alphabetically ascending by the name of the vaccine (content of the child node `name`). For all qualifying vaccines, there should also be a count of *all* patients they have been administered to and also exactly 2 patients which have received it, where the `name` of the patients needs to be output, as well as their risk group as an attribute.

An example output:

```
<vacccinesByRiskGroups>
  <vaccine name="AstraZeneca" type="Adenovirus-based">
      <patientsCount>21</patientsCount>
      <patient riskgroup="High">Person A</patient>
      <patient riskgroup="Low">Person B</patient>
  </vaccine>
</vacccinesByRiskGroups>
```

How to run: 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`.

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 `vaccination-plan-xsd.xml` document via DOM, and then use SAX to parse a `batch-delivery.xml` document and apply certain changes to the `vaccination-plan-xsd.xml` document via the information in the `batch-delivery.xml` document, i.e., to include new objects, and remove indicated old ones. The `batch-delivery.xml` document has the following structure:

- The root is a `batch-delivery` element.
- It is followed by an arbitrary amount of `batch` elements. Each batch element has an attribute `description`. Each batch element has as child nodes a `vaccine` element, with attributes `name` and `type`, followed by a `size` element, and an `order-date` element. There is a list of `patient` elements, which always have as attributes `name`, `birth_year`, `residence` and `risk-group`, and they contain up to two `vaccination_date` child nodes.
- Secondly, there is a `processing` element, with a number of `batch-id` child nodes.

Note: *An example for a* `batch-delivery` *document is found in* `resources/batch-delivery.xml`.

The goal is twofold: 1.) integrate the new batch and patients into the vaccination-plan XML document, *while maintaining the validity of the schema!* Be sure to create new, valid ids and pids for the new batches or patients. Check if a vaccine with the same name already exists in `vaccination-plan`, adding new node if needed. Assume that the `ref-batch` for vaccines of patient nodes is the same as the id of the current batch, and 2.) remove the vaccines (child of `vaccines`) with batches that have an `id` matching those occurring in the `processing` element. To maintain key constraints, remove patients that have received vaccines of these batches.

Hint: To properly test the removal of objects, change the provided `batch-delivery` document to refer to a batch id 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 VPHandler provides a SAX handler, which parses the `batch-delivery.xml` document and modifies a `vaccination-plan-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 batch-deliveryPath, 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 `batch-deliveryPath` variable. For this purpose, you should create an instance of the VPHandler class, which will need the above defined "Document" object as an argument in its constructor. Now parse the `batch-delivery.xml`. The VPHandler will change the document. The final result should be called via the method

`getDocument()` from the class VPHandler 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: VPHandler
  - *Variables*:
    - `static XPath xPath`: can be used to evaluate XPath queries over an XML file.
    - `Document vpDoc`: a DOM representation of an `vaccination-plan-xsd.xml` document.
    - `String eleText`: saves the text content of XML elements.
    - Feel free to declare further variables as needed.
  - *Methods*:
    - `VPHandler(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 `vpDoc`.
    - Define here further methods, to parse the `batch-delivery.xml` document (e.g.: `startElement`, etc.) and to change the `vpDoc` object.

How to run: The command to run the code in `src/ssd/SSD.java` needs three command-line arguments. The first argument is the vaccination-plan document (e.g.: `vaccination-plan-xsd.xml` from Exercise Sheet 1). Next is a `batch-delivery.xml` file (e.g.: `resources/batch-delivery.xml`). The last argument is the file name of the output (e.g.: `output/vaccination-plan-out.xml`). In the ant file, there are two preconfigured targets:

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

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

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

```
XPathExpression xpathExpr = xPath.compile("//batch");
NodeList batchList = (NodeList)xpathExpr.evaluate(vpDoc, XPathConstants.NODESET);
```

Your submission will consist of:

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