Exercise Sheet 2 (SS 2023)

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 `shipment-xsd.xml` and `shipment.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. Make sure you run the commands in the folder where build.xml resides.
- 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-ss23.zip` is produced via the command: `ant zip`

Deadline

at the latest May 30th	**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.*

Office Hours (optional)

Before the submission deadline we offer office hours with our tutors. If you have questions or problems with the material of the exercise sheet, you can get personal support during these office hours. The aim of the office hours is to help your **understanding of the material** and not to check your solutions or to solve the exercise fo you. Attendance is completely optional – times and locations for the office hours will be published 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 shipment XML document (`shipment-xsd.xml`), where validity is defined against the XSD schema `shipment.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 "ship", if so, it will then check if there are any ships matching one of two conditions: 1) this `ship` element has a `tag` element, directly matching *tagname*, or 2) there exists a `product` element, which has a `tag` element, matching *tagname*, and *which has a `ref` element with a `sid` attribute matching the ship's `sid` attribute.* Informally: we want to capture all ships which are directly tagged with *tagname* or which transports some product, which is tagged with *tagname*. If and only if, any such ships are found, it creates a table, with the header "Ships".
  - If *mode* is set to anything else (or empty), this named template will check for any products matching one of two conditions: 1) this `product` element has a `tag` element, directly matching *tagname*, or 2) there exists a `ship` element, which has a `tag` element, matching *tagname*, and *which has a `sid` attribute matching with the `sid` attribute of a `ref` element of this product.* Informally: we want to capture all products directly tagged with *tagname*, or where the ship which transports them is tagged with *tagname*. If and only if, any such products are found, it creates a table, with the header "Products".
  - For any ship (or resp. product) 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 ships (or resp. products) thus matched.
  - The output should be *sorted*. In case "mode" is set to "ship", it should be sorted by the `name` element of the matched ships, in case of products it should match the first `destination` element in their `label` element (if present). For products without a destination, order is undefined.
- A template for `shipment` Elements, which calls the templates of all `t` child nodes.
  Note: This is already provided to you in the draft.
- A template for `t` 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 "ship", and again with *mode* set to "products" (or left empty). Both times the *tagname* parameter shall be set to the value of the `tagname` attribute.

- A template for `ship` Elements, outputting the name, and then use the data from their `info` element to output, in parenthesis, their first tour date (`firstTour` attribute) to their last tour date (`lastTour` attribute, if present), and their place of construction (`placeOfConstruction` attribute).
- A template for `product` Elements, outputting their title (`name`) and in upper-case letter their type (`type`), either clothing or food. Then below their label (`label`). Be sure to properly print *all* content in the label. Then below that, in case the `sid` attribute of a `ref` element matches the `sid` attribute of an existing ship, it should output "Transported By", and within an unnumbered list (`<ul>`) the identifier of the ship that transports it, by calling its template. If no such match exists, then it should output "No shipment detail.".

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 `shipment.xsd` schema from the first exercise. It shall output all ships which transport more than 3 products to the same destination (having the same `destination` child node in their `label` elements). Products with multiple destination elements in their labels shall be counted towards each destination. The name of the ship shall appear as an attribute `name`, as well as the destination (`destination`). The output should be sorted by the first tour date (the string in `firstTour`) of the ships. For these ships and the qualifying destinations, a count of *all* products the ships are transporting should be outputted, as well as exactly 3 products with the same destination, where only the `name` of these products needs to be output. If there are multiple products with the same destination, your query just needs to output three products from it. **Note:** Avoid duplicate destination entries (i.e. covering for the same ship the same destination more than once).

An example output:

```
<largeOrdersOnShips>
   <ship name="Ship 1" destination="UK">
       <productCount>4</productCount>
       <product>Kiwi</product>
       <product>Beef</product>
       <product>Coat</product>
   </ship>
</largeOrdersOnShips>
```

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

- The root is a `freight` Element.
- It is followed by an arbitrary amount of `product` elements. They have as child nodes a `name`, a `type` element (containing as content either "food" or "clothing"), where the `type` element can have optional attributes for type of food and storage information, a list of `tag` elements, which are text nodes, a `producer` element, stating the name of the producer, a `label` element, a `destination` element, indicating where the product will be shipped to, and a `ref` element, with `sid` attribute providing the identifier of the ship that will transport the product.
- Furthermore, there is a `shortage` element, containing a number of `catalog` child nodes.

Note: An example for a `freight` document can be found under `resources/freight.xml`. If you use your own `shipment-xsd.xml` document, make sure to modify the `sid` attributes in the `freight` document to refer to existing ships in your XML document.

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

Hint: To properly test the removal of products, change the provided `freight` 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 ShipmentHandler provides a SAX handler, which parses the `freight.xml` document and modifies a `shipment-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 freightPath, 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 `freightPath` variable. For this purpose, you should create an instance of the ShipmentHandler class, which will need the above defined "Document" object as an argument in its constructor. Now parse the `freight.xml`. The ShipmentHandler will change the document. The final result should be called via the method `getDocument()` from the class ShipmentHandler 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: ShipmentHandler
  - *Variables*:
    - `static XPath xPath`: this XPath instance can be used to evaluate XPath queries over an XML file.
    - `Document shipmentDoc`: saves a DOM representation of an `shipment-xsd.xml` document.
    - `String eleText`: saves the text content of XML elements.
    - Feel free to declare further variables as needed.
  - *Methods*:
    - `ShipmentHandler(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 `shipmentDoc`.
    - Define here further methods, to parse the `freight.xml` document (e.g.: `startElement`, etc.) and to change the `shipmentDoc` 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 shipment document (z.B. `shipment-xsd.xml` from Exercise Sheet 1). The next is a `freight.xml` file (e.g.: `resources/freight.xml`). The last argument is the file name of the output (e.g.: `output/shipment-xsd-out.xml`). In the Ant file, there are two preconfigured targets:

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

The files `resources/shipment-sample-xsd.xml` and `resources/shipment-sample-xsd-out.xml` are provided in the template on TUWEL, with the latter being a sample shipment XML document, created after running the program with the `freight.xml` and the former shipment 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(shipmentDoc, XPathConstants.NODESET);
```

Your submission will consist of:

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