5. Unit: XML Processing with Java (II)

Exercise 5.1 (XML Digester: River Networks in Mondial) Use the XML Digester in a Java class that creates an internal data structure about seas, rivers and their tributaries (including lakes) and generates an output similar to that in Exercise 2.1.
Additionally, this output should show for every river the sum of the length of all rivers flowing into it (directly or indirectly).

Exercise 5.2 (XML Schema & JAXB: Arithmetic Trees)

Design an XML markup for representing *arithmetic term trees* over integers (cf. Exercise Sheet 3 (XSLT), Exercise 4 of the XML lecture) that is appropriate for use with JAXB. Define an XML Schemaa and create base classes. Extend these classes with the common functionality for arithmetic trees: (i) loading a tree, (ii) outputting the tree contents in as a term, (iii) outputting the tree in a graphical way (e.g., nested tables) in HTML, (iv) evaluating the term.

Exercise 5.3 (Independence in JAXB)

- Do the independence use case exercise using JAXB.
  (`mondial.xsd` is available on the Web site)
  Do as much as possible with JAXB. Where does a problem occur, and why? Solve it with a reasonable workaround.
- Why is it not a good idea to do it with the Digester?

Exercise 5.4 (Parsing of and Queries against Sloppy HTML Pages)

Often, HTML pages are not strictly valid XHTML, and thus cannot be queried by XQuery (like wikipedia, country pages from `https://www.cia.gov/library/publications/the-world-factbook/` or from `https://www.citypopulation.de/`, soccer tables from `https://www.kicker.de/`). Usually, the problem are empty elements that consist only of opening tags instead of correct `<... />` syntax.

Consider the XML Parsing APIs DOM, SAX and StAX when parsing such input.

- (i) what happens?
- (ii) what does that "mean" wrt. the underlying metaphor of the DOM tree and of the stream in SAX/StAX?
- (iii) where does it *exactly* happen in the Java code (you can use the previous Mondial examples by just exchanging the mondial.xml with a sloppy HTML input)?
- (iv) which of the APIs can be rather easily adapted to be able to handle sloppy HTML?

Demonstrate your solution by implementing some simple query against a wikipedia page (e.g., elevation of the Mont Blanc).

If you like, you can extend it to loop over all mountains from mondial.xml.

Exercise 5.5 (Application: An E-Exams System) This exercise will be reused for a Web Service.

Consider an system for providing electronic exams - like the ILIAS system that has been used for the exam in the Summer Term. Here, we primarily focus on the examiners' side, and on XML technologies. The supporting user interface should be minimal, HTML-form-based, proof-of concept.

The structure of the application is as follows:

- Every exam ("the SSD/XML exam from Summer 2020") consists of the following:
  - an opening text (in HTML markup),

- an ordered list of exercises, where every exercise has the following:
  - a headline, like "XPath I",
  - a number of points,
  - an exercise text (in HTML markup) (maybe later there are also substructures for maintaining multiple choice and solutions – keep it simple first)
  - a type, e.g. "text", "multiple choice", "upload" "program code" (which determines e.g. the editor used for it; the examinees interface will not be implemented here), (opaque "file upload", which may include binaries like diagrams or voice recordings).
  - optionally a reference solution.

For each individual student's exam, "ParticipantsExam", also an appropriate XML fragment of the overall data structure is created (this can be as children of the above exam, or as a separate XML file containing all students):

- the student's ID, the reference to the exam ("the SSD/XML exam from Summer 2020").
- for each exercise, the student's solution as either:
  - HTML (for exercises that have answer text, the students' answer text is mapped in the student user interface to HTML),
  - an appropriate ID of multiple-choice answers (maybe several answers could be given and could be correct),
  - program code,
  - opaque upload stuff. As this can be binary, ILIAS uses base64 encoding into a string. An alternative is to use separate files and to store a file reference in an attribute. (this is not our focus here, but if somebody wants to play with...)
  - later: the corrector's comments as string or HTML.
- Testing Functionality: When the students finish the exam, their answers are submitted.
  - later: via Web Service/HTML form (we will not implement the students' interface in detail, use a simple mock HTML form)
  - now: simple Java methods, just for testing, e.g.
    - Exam.createParticipantsExam(matrnr)
    - ParticipantsExam.receiveSolution(ExerciseNr, text/file/check-item-no) which then depends on the type of the exercise
    - Exam.receiveSolution(Participant, ExerciseNr, text/file/check-item-no)

This exercise consists of the following steps (to be continued later):

a) Design an appropriate XML Schema.

b) Generate basic classes and interfaces for the schedule application, using the JAXB Schema binding compiler on your XML schema.

c) Two dump files have been created by ILIAS from the Summer Term 2020 "Semistructured Data and XML" exam (anonymized and a little bit cleaned):

- `https://www.dbis.informatik.uni-goettingen.de/Teaching/XML-P/Ilias/Ex_qti.xml`: The exercise texts and additional data of the exercises (according to the QTI ("IMS Question and Test Interoperability") standard)
- `https://www.dbis.informatik.uni-goettingen.de/Teaching/XML-P/Ilias/Ex_results.xml`:
  The solutions and achieved points of the exercises (the correction comments are missing ... to be debugged in Ilias)

The first one is OK (nested XHTML in XML), the latter is a rather dirty dump: All data is dumped as attributes.

- Thus, HTML answers are not actually nested HTML, but serialized as a string escaped by using Entities instead of nested tags. The answers for code questions are base64-encoded as attribute values (the answers to text queries are escaped HTML).

Maybe the best way to do it is as follows:

- extend your XML Schema with (optional) fields `rawAnswer` which holds the dirty uuencoded/escaped String.
- write an XSL stylesheet that transforms the dump XML format into XML files `ssdExamData.xml` (the questions and their texts) and `ssdExamSolutions.xml` (the individual exams) according to your XML Schema, the latter using the `rawAnswer` field.
- Unmarshal them into memory using JAXB.
- extend the constructor (if possible) or add a method to ParticipantExamExercise that
  - (depending on the query type "text" or "program code") replaces the escaped entities in the escaped XHTML by "<" and ">" to obtain XML source code and then parses this into an XHTML tree,
  - uudecodes the code answers. Java Sketch:

```
import java.util.Base64;
import java.util.Base64.Decoder;

Decoder decoder = Base64.getDecoder();
byte[] decodedBytes = decoder.decode(input);
output = new String(decodedBytes);
```

- delete the `rawAnswer` field
- Marshal the result XML into a file to have a look at it whether it is OK.
- (The next step is then to generate LaTeX/DocBook(?)/XSL-FO(?) from this to print the participants' exams).