3. Unit: XML Processing with Java - SAX, DOM, JAXB

Exercise 3.1 (SAX: Cities in Mondial)

Write SAX parsers in Java for the following tasks:

a) Output the name of all countries in mondial.xml into an HTML page.

b) Output the population of the capital of Germany via System.out.

c) For each country in mondial.xml, output an HTML table containing the names and – if present – population sizes for each city in the country. Use a <ul> (unordered list) environment with one list item per country.
Example:. ...

- ...
- **Germany**

Stuttgart	588482
Mannheim	316223
Karlsruhe	277011
...	...

- **Hungary**

Bekescaba	404000
Hodmezovasarhely	-
...	...

- ...

Some cities have multiple population entries. What must be done to select only the most actual one? (only describe idea)

d) Output all country names, the country's capital and the country capital's population – if available – into an HTML table.
Example:

country	**capital**	**capital population**
Albania	Tirane	192000
Greece	Athens	885737
Macedonia	Skopje	
...	...	...

e) Modify the event handler, enabling the parser to output for each country with more than 10 valid city population entries:

- the country's name
- the overall number of cities
- each city with name and population number
- the average city population
- the empiric deviation for the city populations:
  the average of the absolute value of the difference between a city's population and the average city population, or to put it shorter:

$$f_{dev}(p_1, \ldots, p_n) = (1/n)\Sigma_{i=0}^{n}|p_i - p_{avg}|$$

  for a country with n cities with populations $p_1, \ldots, p_n$ and an average city population p_{avg}.
- inside the city table, mark (either by color, font etc) (1) the capital city, and (2) the city whose population is closest to the average city population of the country.

Exercise 3.2 (DOM/Schema: Schedules in XML)

Imagine an XML structure for representing a personal schedule ("Terminkalender") with dates:

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule name="Meine Termine">
    <year n="2006">
        <month n="1">
            <day n="16">
                <date id="a1" starttime="15:00" duration="2:00">
                    <name>Institutsbesprechung</name>
                    <description>monatliche Sitzung, Tagesordnung zz noch nicht bekannt</description>
                    <location>Sitzungszimmer</location>
                </date>
            </day>
        </month>
    </year>
</schedule>
```

- Each schedule contains zero or more year elements.
- Each year element contains zero or more month elements.
- Each month element contains zero or more day elements.
- Each day element contains zero or more date elements.
- Each date element, describes a certain date, yielding information about starting time, duration, location and a textual description of the date.
- dates are no longer than 24 hours and do not span over more than one day.
- You can assume the month and day elements to appear in their correct temporal order (first january, then february, then march etc) inside the document.

a) Create an XML Schema for the above XML snippet. Assume that each date has a unique ID attribute.

b) Parse the given XML snippet into a DOM tree.

c) Complete the DOM structure in the memory:
each year element has to contain 12 month elements, and each month element has to contain the correct number of days (use the java.util.GregorianCalendar classes for calculating if the current year is a leap year etc).

d) Write a method insertDate for inserting dates into the DOM Tree. If the insertion leads to a collision between the newly inserted and the old dates, output a meaningful warning at runtime.

e) Write a method which returns all dates during a time interval specified by start time (day and time), and end time (day and time).
Write a / expand your test program demonstrating the functionality of inserting dates and receiving dates from specified time intervals.

f) Use the stylesheet from the earlier schedule-XSLT-exercise. Apply the stylesheet DIRECTLY to the dom tree (without writing the data into an XML file and then transforming the file) and output the resulting HMTL table also into a file.
Use the javax.xml.transform classes.

Exercise 3.3 (XML Schema & JAXB)

Remember the "schedule" DOM-tree exercise from the last section.

- Use the "Java API for XML Binding" (JAXB) for creating a schedule application similar to that from the previous section:
  a) Generate basic classes and interfaces for the schedule application, using the JAXB Schema binding compiler on your XML schema.
  ***Hint**: command: "xjc -help" issued on the command line displays usage and possible options. At `/afs/informatik.uni-goettingen.de/course/xml-prakt/xml/JAXB-README` you find an explanation for installing, testing and using JAXB on your system.*
  b) Use the already familiar schedule1.xml file. Use JAXB for unmarshalling the schedule data from the XML file into memory.
  c) Implement a fillUp(...) method – in analogy to the DOM exercise – which fills up missing months inside a year object, and missing days inside a month object.
  d) Now, marshal the in-memory schedule into a in-memory DOM Tree, again using JAXB.
  e) Apply the stylesheet from the earlier XSLT exercise to the in-memory DOM structure, writing the HTML output into a output-file.

The Java WebService Tutorial under http://java.sun.com/webservices/docs/1.4/tutorial/doc/index.html will be a great deal of help working with JAXB.