3. Versuch: XML und Java: SAX & StAX, DOM, JAXB

Aufgabe 3.1 (DOM Basics)

Erzeugen Sie eine DOM-Instanz von mondial.xml und implementieren Sie die folgende Anfrage mit den DOM-Operationen (wenden Sie dabei nciht XPath direkt an): Geben Sie für alle Organisationen, die ihren Sitz in der Hauptstadt eines Mitgliedslandes haben, den Namen der Organisation sowie der betreffenden Stadt aus.

Aufgabe 3.2 (SAX: Queries against Mondial)

Schreiben sie SAX Event Handler für die folgenden Aufgaben:

a) Erzeugen Sie ein HTML-File, das die Namen aller Länder in mondial.xml listet.

b) Geben Sie die Einwohnerzahl der Hauptstadt von Deutschland auf den Bildschirm aus. Prüfen Sie, dass Ihr Algorithmus auch für Slowenien anstelle von Deutschland korrekt arbeitet.

c) Erweitern Sie die Lösung des vorhergehenden Aufgabenteils: Schreiben Sie den Namen jedes Landes, die Hauptstadt und die Einwohnerzahl der Hauptstadt – falls vorhanden – in eine HTML-Tabelle.
Beispiel:

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

d) Geben sie für jedes Land in mondial.xml eine HTML-Tabelle mit den Namen und – falls vorhanden – Bevölkerungsdaten jeder Stadt im betreffenden Land aus. Setzen sie die Tabellen in eine -Umgebung, mit einem <li> list-item für jede Tabelle.
Beispiel:

- ...
- **Germany**

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

- **Hungary**

Bekescaba	404000
Hodmezovasarhely	-
...	...

- ...

Einige Städte haben mehrere Population-Einträge. Wie sorgt man dafür dass nur der aktuellste Wert ausgegeben wird? (nur die Idee skizzieren)

e) Erweitern Sie den Event-Handler aus dem vorhergehenden so dass er für jedes Land mit mehr als zehn Städten deren Einwohnerzahl bekannt ist, folgende Daten in eine Tabelle ausgibt:
- den Namen des Landes
- die Anzahl der Städte in dem jeweiligen Land
- jede Stadt mit Namen und – falls vorhanden – Einwohnerzahl
- die mittlere Einwohnerzahl aller Städte des Landes
- in der Tabelle mit den Städten, markieren Sie mittels Farbe, Schriftart, etc. (1) die Landeshauptstadt, sowie (2) die Stadt, deren Einwohnerzahl am dichtesten an der durchschnittlichen Einwohnerzahl der Städte dieses Landes liegt.

f) Schreiben Sie (ggf. unter Zuhilfenahme weiterer Java-XML-Pakete) einen Event-Handler der alle (direkten und indirekten) Zuflüsse des Zaire sowie ihre Gesamtlänge ausgibt.

Aufgabe 3.3 (StAX: Queries against Mondial)

a) Implementieren Sie die Aufgabenteile a) und d) der SAX-Aufgabe unter Verwendung von StAX. Verwenden Sie dabei soweit möglich denselben Programmcode wieder und passen Sie nur die Aufrufe an die StAX-Umgebung an.

b) Implementieren Sie die folgende Anfrage in StAX: Geben Sie für alle Organisationen, die ihren Sitz in der Hauptstadt eines Mitgliedslandes haben, den Namen der Organisation sowie der betreffenden Stadt aus.

c) Experiment: Setzen Sie es so um, dass (b) Ergebnisse der Form

```
<result>
 <organization name="European Union"/>
 <city name="Brussels"/>
</result>
```

erzeugt und diese in einen XMLOutputStream schreibt, auf den Sie mit einem zweiten Thread einen Reader setzen, der wiederum auch über StAX nur die <city>-Elemente ausfiltert und ausgibt. Lassen Sie beide Threads nach System.out loggen, um die Nebenläufigkeit zu veranschaulichen.

In den folgenden beiden Aufgaben wird ein Terminkalender auf zwei Arten in Java implementiert. Beide sollen dieselbe DTD/XML Schema verwenden und dieselbe grundsätzliche Funktionalität anbieten:

1) DOM,
2) JAXB.

In der nachfolgenden "Web Service"-Aufgabe werden sie kombiniert.

Aufgabe 3.4 (DOM/Schema: Schedules in XML)

Gegeben sei eine XML-Struktur zum Darstellen eines Terminkalenders mit eingetragenen Terminen (using xsd:dateTime):

```
<?xml version="1.0" encoding="UTF-8"?>
<schedule name="Meine Termine">
  <year n="2010">
    <month n="1">
      <day n="16">
        <entry starttime="T15:00" duration="PT02:00">
          <name>Institutsbesprechung</name>
          <description>monatliche Sitzung, Tagesordnung zz noch nicht bekannt</description>
          <location>Sitzungszimmer</location>
        </entry>
      </day>
    </month>
    <month n="5">
      <day n="2">
        <entry starttime="T18:00" duration="PT00:50">
          <name>Telefonat</name>
        </entry>
      </day>
    </month>
  </year>
</schedule>
```

- Jeder Terminkalender enthält null oder mehr year-Elemente.
- Jedes year-Element enthält null oder mehr month-Elemente.
- Jedes month-Element enthält null oder mehr day-Elemente.
- Jedes day-Element enthält null oder mehr date-Elemente.
- Jedes entry-Element beschreibt einen Termin mittels Startzeitpunkt, Dauer, Ort und eines kurzen Textes.
- Ein Termin dauert niemals länger als 24 Stunden und geht nicht über mehrere Tage.
- Die year-, month-, day- und date-Elemente stehen im Dokument in ihrer korrekten zeitlichen Abfolge.

a) Erzeugen Sie dazu erst ein XML Schema (nutzen Sie dabei den dateTime-Datentyp aus XML Schema), und validieren Sie eine aussagekräftige Beispielinstanz.

b) Ergänzen Sie die DOM-Struktur im Speicher:
Jedes year-Element muss 12 month-Elemente enthalten, jedes month-Element muss die richtige Anzahl von Tagen enthalten. Sie können java.util.Calendar und java.util.GregorianCalendar verwenden zur Berechnung von Schaltjahren etc.

c) Schreiben Sie eine Methode void insertEntry um einen weiteren Termin in den DOM-Baum einzufügen. Falls der Termin kollisionsfrei einfügbar ist, wird er eingefügt, und "true" zurückgegeben. Anderenfalls wird "false" zurückgegeben.

d) Schreiben Sie eine Methode die alle Termine zurück liefert die zwischen einem gegebenen Start- und einem Endzeitpunkt liegen, jeweils beschrieben durch Datum und Uhrzeit.
Schreiben sie ein / erweitern Sie ihr Testprogramm, so dass es Termine in den DOM-Baum einfügt und Termine aus einem bestimmten Zeitintervall abfragt.

e) Verwenden Sie das Stylesheet aus der vorherigen Schedule-XSLT-Aufgabe und wenden Sie es DIREKT auf den DOM-Baum an (nicht über den Umweg, den DOM-Baum zuerst in eine Datei zu schreiben und dann die Datei zu transformieren). Geben Sie die resultierende HTML-Tabelle dann in eine Datei aus.
Verwenden Sie die javax.xml.transform-Klassen.

Aufgabe 3.5 (XML Schema & JAXB)

Erinnern Sie sich an die DOM-Baum-Aufgabe aus dem letzten Abschnitt.

- Verwenden Sie das "Java API for XML Binding" (JAXB) zum Schreiben einer Terminkalenderanwendung ähnlich der in der DOM-Aufgabe:
  a) Generieren Sie mit dem JAXB Schema binding Compiler aus Ihrem XML Schema schedule.xsl Basisklassen und Interfaces für Ihre Terminkalenderanwendung.
  Hinweis: Aufruf des Compilers von der Kommandozeile: "xjc -help" (liefert den Help-Screen auf dem die Optionen und Parameter erläutert sind. Unter
  `/afs/informatik.uni-goettingen.de/course/xml-prakt/xml/JAXB-README`
  finden Sie Installations- und Verwendungshinweise für JAXB.
  b) Verwenden Sie die schedule.xml-Datei aus der DOM-Aufgabe. De-serialisieren Sie diese Datei mittels JAXB in den Speicher ("unmarshalling").
  c) Implementieren Sie eine fillUp()-Methode – analog zur DOM-Aufgabe – die innerhalb eines Year-Objekts fehlende Monate und innerhalb eines Month-Objekts fehlende Tage ergänzt.
  d) Serializieren Sie das "aufgefüllte" schedule-Objekt in einen DOM-Baum im Speicher mittels ("marshalling") mit JAXB.
  e) Wenden Sie das Stylesheet aus der DOM-Aufgabe auf den DOM-Baum im Speicher an, um eine HTML-Repräsentation davon zu erzeugen. Geben Sie das Resultat in eine HTML-Datei aus.

Das Java WebService Tutorial auf http://java.sun.com/webservices/docs/1.4/tutorial/doc/index.html ist für die Arbeit mit JAXB ausgesprochen hilfreich.

Aufgabe 3.6 (Web Services)

Installieren Sie die beiden Kalender-Varianten aus den vorherigen Aufgaben als Web-Services (unter localhost).

Da beide prinzipiell dieselbe Funktionalität (und damit intern dieselben Methoden) unterstützen, können von dem folgenden einige Dinge für beide benutzt werden:

- Versehen Sie beide mit einem (einfachen) Web-Benutzerinterface:
  - Ausgabe des Terminkalender als HTML-Seite
  - Einfügen von Einträgen
- Erweitern sie beide Varianten um eine Methode, die für einen gegegebenen Zeitraum (Anfangs- und Enddatum) und eine gegebene Dauer (in Stunden) alle noch kollisionsfrei möglichen Slots (wochentags zwischen 8-18 Uhr) feststellt, und in einer (von Ihnen festzulegenden) Serialisierung als XML-Dokument ausgibt.
- Erweitern Sie beide Varianten um eine Servlet-Methode, die eine solche Liste von einer HTTP-Verbindung entgegennimmt, mit den eigenen Terminen abgleicht, und den ersten passenden Termin (i) einträgt, und (ii) per Service-Aufruf auch bei dem Anfragenden einträgt.

  Das Lesen der Liste aus der HTTP-Verbindung soll dabei SAX/StAX-basiert erfolgen.

  Testen Sie die Stream-Funktionalität, indem Sie einen langen Zeitraum wählen, so dass der Antworter bereits einen passenden Slot zurücksendet, bevor der Anfrage-Stream fertig gelesen ist (durch geeignetes logging nach catalina.out).