Dokumentation
WBA2 – Phase 1

Autor: David Wachs

Inhaltsverzeichnis

Aufgabe 1

Erklären Sie kurz die Begriffe Wohlgeformtheit, Validität und Namespaces im Bezug auf XML und XML-Schema.

Wohlgeformtheit bedeutet, dass es in einem XML-Dokument nur genau einen Wurzelknoten bzw. ein Wurzelelement gibt. XML Metadaten also Kommentare und Anweisungen für verarbeitende Prozesse (Processing Instructions) sind davon ausgeschlossen.
Außerdem müssen nichtleere Elemente eine öffnende und schließende Markierung haben. Desweiteren müssen alle Werte von Attributen durch Hochkommas oder durch Anführungszeichen eingeschlossen sein.
Nicht-Wurzelelemente müssen von ihren Elternelementen vollständig eingeschlossen sein.
Es dürfen nur Zeichen aus dem spezifizierten Zeichensatz verwendet werden.

Validität (Gültigkeit) bedeutet, dass ein XML-Dokument nach den Vorgaben bzw. Definitionen eines XML-Schemas bzw. einer Grammatik geschrieben wurde. Es wird als gültig bzw. valide bezeichnet wenn es wohlgeformt ist, den Verweis auf eine Grammatik enthält und das von der Grammatik beschriebene Format einhält.

Namespaces (Namensräume) werden verwendet um zu vermeiden, dass bei der Verwendung von mehreren Vokabularen bzw. XML-Sprachen in einem Dokument Mehrdeutigkeiten auftreten. D.h. damit wird sichergestellt, dass die verwendeten Vokabulare eindeutig identifiziert werden können.

Aufgabe 2

a)

Erzeugen Sie ein XML-Dokument, dass die Daten des folgenden Formulars vollständig erfasst:
http://www.gm.fh-koeln.de/~vsch/anmeldung/gruppenanmeldung.html
Füllen Sie das Dokument mit einem Beispieldatensatz. Achten Sie darauf, dass über das Formular mehrere Personen gleichzeitig erfasst werden können.
***Wichtig:** Es sollte nicht die HTML-Struktur der Webseite in der XML-Datei abgebildet werden, sondern die zu übertragenden Daten.*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Gruppenanmeldung>
    <Gruppenleiter id="1">
        <Vorname>Max</Vorname>
        <Nachname>Mustermann</Nachname>
        <Email>max.mustermann@gmail.com</Email>
        <Geburtstag>17.05.1990</Geburtstag>
        <Erfahrung>Amateur</Erfahrung>
        <Schlagzeug>nicht vorhanden</Schlagzeug>
        <Anmerkung></Anmerkung>
    </Gruppenleiter>
```

Bei dieser Aufgabe habe ich mich zuerst darüber informiert, was genau der Unterschied zwischen einem Attribut und einem Element ist. Ein Attribut beschreibt eine Eigenschaft eines Elements. Ein Element kann mehrere Attribute enthalten. Attribute können keine geordneten Werte enthalten. Elemente hingegen geordnete Unterelemente. D.h. man kann die Reihenfolge der Daten festlegen. Desweiteren sind Attribute schwer erweiterbar. Sie bieten sich an, wenn Daten nur einmal vorkommen sollen wie bei Identifikatoren. Attribute sollten für Informationen verwendet werden, die nicht relevant für die Daten sind, welche später dem Benutzer präsentiert werden.
Deshalb habe ich für meine XML-Struktur hauptsächlich Elemente verwendet. Nur die ID des Gruppenleiters habe ich als Attribut gespeichert.
Allerdings ergaben weitere Recherchen, dass es keine festgelegte Regel gibt, wann Attribute und wann Elemente verwendet werden sollten. Man könnte z.B. den Vornamen auch als Attribut speichern, da sich die Struktur dieser Daten nicht ändert und die Daten auch keine Unterstruktur besitzen und voraussichtlich auch niemals besitzen werden.

Allerdings habe ich dann versucht mich an den „Quasi“-Standard (siehe z.B. w3schools) zu halten der besagt, dass man Attribute nur für Informationen verwendet, welche für die eigentlichen Daten nicht relevant sind, also Metainformationen. Bei der Aufgabe 3 habe ich mich dann nicht mehr Konsequent daran gehalten.

b)

Erzeugen Sie ein JSON-Dokument, dass zu ihrem XML-Dokument äquivalent ist.

```json
{
Gruppenleiter": [
        {
        "vorname": "Max", "nachname":"Mustermann", "email":"max.mustermann@gmail.com",
        "Geburtsdatum":"17.05.1990", "Erfahrung":"Amateur", "Schlagzeug":"nicht vorhanden",
        "Anmerkung":"" },
        { "vorname": "Hans", "nachname":"Mustermann", "email":"hansmustermann@hotmail.de",
        "Geburtsdatum":"20.12.1985", "Erfahrung":"Profi", "Schlagzeug":"vorhanden",
        "Anmerkung":"" }
    ]
}
```

In JSON werden die Daten als Array gespeichert Diese können weitere Objekte enthalten. JSON orientiert sich dabei an der Syntax von JavaScript. Ich habe hier das Array „Gruppenleiter“, welches dann die Daten der einzelnen Gruppenleiter enthält.

Aufgabe 3

a)

Gegeben ist folgendes Rezept:
http://www.chefkoch.de/rezepte/24641006006067/Lenchen-s-Schokoladenkuchen.html
Entwickeln Sie ein XML-Dokument, in dem die Daten des Rezeptes abgebildet werden. Achten Sie darauf, dass das Dokument semantisch möglichst reichhaltig ist. Bei dieser und den folgenden Aufgaben lassen sie bitte die Daten in der Marginalspalte auf der rechten Seite weg.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Rezept bezeichnung="Lenchen's Schokoladenkuchen">
    <zutatenliste anzahlPortionen="16">
        <zutat name="Butter" menge="200" einheit="g" />
        <zutat name="Zucker" menge="200" einheit="g" />
        <zutat name="Schockolade" menge="200" enheit="g" />
        <zutat name="Mehl" menge="120" einheit="g" />
        <zutat name="Backpulver" menge="0.5" einheit="Tl" />
        <zutat name="Vanillezucker" menge="1" einheit="Pkt." />
        <zutat name="Ei(er)" menge="4" />
    </zutatenliste>
    <zubereitung dauer="60" schwierigkeit="normal" brennwert="295">
        <Beschreibung>Butter und Schokolade im Wasserbad schmelzen...</Beschreibung>
    </zubereitung>
    <Kommentarliste>
        <Kommentar id="1">
            <Username>swieselchen</Username>
            <Datum>07.02.2002 18:49</Datum>
            <Text>Habe Deinen Kuchen gestern gebacken...</Text>
        </Kommentar>
        <Kommentar id="2">
            <Username>Mane</Username>
            <Datum>07.01.2003 15:06</Datum>
            <Text>Habe ihn eben gebacken und bin begeistert.Super Rezept:-)</Text>
        </Kommentar>
    </Kommentarliste>
</Rezept>
```

Hier bin ich etwas von der Struktur aus Aufgabe 2 abgewichen. Ich habe also Attribute und Elemente an verschiedenen Stellen verwendet. Alle Zutaten sind innerhalb einer Zutatenliste gespeichert. Da jede Zutat eindeutig ist habe ich die Daten der Zutat als Attribute und nicht als Elemente gespeichert. Außerdem ist es unwahrscheinlich, dass sich die Struktur dieser Daten später noch ändern wird.
Die Kommentare werden anders gespeichert. Es gibt wie für die Zutaten ein umfassendes Element, jedoch habe ich die einzelnen Daten als Elemente gespeichert. Es könnte z.B. vorkommen, dass die Struktur des Datums später geändert werden soll. Dies ist bei einem Element einfacher zu realisieren als bei einem Attribut. Jeder Kommentar hat noch ein Attribut „ID“ für die eindeutige Identifizierung.

b)

Betrachten Sie nun andere Rezepte auf der Webseite http://www.chefkoch.de. Beschreiben Sie welche Gemeinsamkeiten die Rezepte hinsichtlich ihrer Daten haben und worin Sie sich unterscheiden.

- **Unterschiede**
  - Anzahl der Zutaten
  - Zutaten
  - Anzahl der Bilder
  - Arbeitsdauer
  - Einheit der Mengenangaben
- **Alle Rezepte haben**
  - Eine Bezeichnung
  - Eine Liste von Zutaten
    - Zutatenname
    - Menge (*ist nicht immer ganzzahlig*)
    - Mengeneinheit (*unterschiedlich z.B. g, l, …)*
  - Eine Zubereitungsanleitung
    - Arbeitsdauer (*immer in Minuten angegeben*)
    - Schwierigkeitsgrad (*simpel, normal, pfiffig*)
    - Brennwert (*immer in kcal angegeben*)
    - Beschreibung / Vorgehensweise (*immer Text*)
  - Eine Angabe über die Anzahl der Portionen
  - Kommentare

c)

Arbeiten Sie die Kriterien heraus, die für die Entwicklung einer XML-Schema-Datei beachtet werden müssen.
Die Schema-Datei soll die Struktur für eine XML-Datei definieren, in der mehrere unterschiedliche Rezepte gespeichert werden können.
Ziel ist es, dass das XML-Schema möglichst restriktiv ist, so dass in der XML-Datei möglichst semantisch sinnvolle Daten bezüglich der Rezepte gespeichert werden können. Ziehen Sie beim Aufstellen der Kriterien u.A. folgende Fragestellungen in Betracht:

- *Welche Daten müssen in simple und welche in complex-types abgebildet werden?*
- *Für welche Daten ist die Abbildung in Attributen sinnvoller?*
- *Welche Datentypen müssen für die Elemente definiert werden?*
- *Welche Restriktionen müssen definiert werden?*

Kriterien

- Unterschiede zwischen den Rezeptdaten
- Abhängigkeiten der Rezeptdaten (z.B. die Mengeneinheit hängt von der Zutat ab)
  - Eine Zutat hat also verschiedene Eigenschaften, die unterschiedlich sein können => Komplexer Typ
- Wie oft dürfen bestimmte Daten für ein Rezept vorkommen? Z.B. mehrere Zutaten aber immer nur ein Titel pro Rezept
- Simple types
  - Rezeptbezeichnung/Name (String)
  - Zutatbezeichnung (String)
  - Menge (float)
  - Anzahl der Portionen (integer)
  - Beschreibung/Vorgehensweise (string)
  - Brennwert (integer)
  - Arbeitsdauer (integer)

- Complex Types
  - Liste mit Zutaten
  - Zutaten
  - Zubereitung
  - Kommentare
- Restricitons
  - Mengeneinheit
  - Schwierigkeitsgrad

d)

Erstellen Sie nun ein XML-Schema auf Basis ihrer zuvor definierten Kriterien. Generieren Sie nun auf Basis des Schemas eine XML-Datei und füllen Sie diese mit zwei unterschiedlichen und validen Datensätzen.

Mein XML-Schema ist wie folgt aufgebaut:

```
3  <xs:element name="rezeptliste">
4      <xs:complexType>
5          <xs:sequence maxOccurs="unbounded">
6              <xs:element name="rezept">
7                  <xs:complexType>
8                      <xs:all maxOccurs="1">
9                          <xs:element name="zutatenliste">
10                             <xs:complexType>
11                                 <xs:sequence maxOccurs="unbounded">
```

Zuerst muss immer ein Element „rezeptliste“ erstellt werden, welches die verschiedenen Rezepte enthält. Da die Anzahl der Rezepte nicht begrenzt sein soll, muss beim *sequence*-Element der *maxOccur*-Indikator auf „*unbounded*“ also „unbegrenzt“ gesetzt werden (Zeile 5).
Anschließend kommt innerhalb des Sequence-Elements ein Element für das Rezept. Dies habe ich als Komplexen Typ definiert, da ein Rezept aus verschiedenen Elementen und Attributen besteht.
Ich habe dann in Zeile 8 den all-Indikator verwendet, da innerhalb eines Rezeptes die einzelnen Elemente, wie z.B. die Zutatenliste und die Zubereitungsbeschreibung nur einmal vorkommen dürfen. Den *maxOccur*-Indikator hätte man innerhalb des all-Indikators auch weglassen können. Aus Gründen der Übersicht habe ich diesen aber trotzdem verwendet. Ab Zeile 9 habe ich dann ein Element „zutatenliste“, welches die einzelnen Zutaten des Rezepts enthält. Das Element selber darf innerhalb des Rezeptes nur einmal verwendet werden, die Anzahl der enthaltenen Zutaten allerding ist unbegrenzt (Zeile 11).
Ab Zeile 12 beginnt dann die Definition des Zutaten-Elements. Eine Zutat hat drei Attribute die sie beschreiben. Einen Namen, eine Menge und eine Mengeneinheit. Die Mengeneinheit habe ich als *restriction* definiert, da sie sich von Zutat zu Zutat unterscheidet.
Die Mengeneinheit ist als optionales Attribut angegeben, da nicht alle Zutaten eine Mengeneinheit benötigen (z.B. 1 Ei).

```
12 <xs:element name="zutat">
13     <xs:complexType>
14         <xs:attribute name="zutatenname" type="xs:string" />
15         <xs:attribute name="menge" type="xs:integer" />
16         <xs:attribute name="mengeneinheit" use="optional">
17             <xs:simpleType>
18                 <xs:restriction base="xs:string">
19                     <xs:enumeration value="g"/>
20                     <xs:enumeration value="l"/>
21                     <xs:enumeration value="Tl"/>
22                     <xs:enumeration value="El"/>
23                     <xs:enumeration value="ml"/>
24                     <xs:enumeration value="Bund"/>
25                     <xs:enumeration value="Würfel"/>
26                     <xs:enumeration value="Becher"/>
27                     <xs:enumeration value="Pkt."/>
28                     <xs:enumeration value="evtl."/>
29                     <xs:enumeration value="Stück"/>
30                 </xs:restriction>
31             </xs:simpleType>
32         </xs:attribute>
33     </xs:complexType>
34 </xs:element>
```

Den Vorgang der Zubereitung habe ich auch in einem Element abgebildet. Dieses Element befindet sich aber nicht innerhalb der Zutatenliste sondern auf gleicher Ebene, also innerhalb des all-Indikators. Es darf also nur einmal pro Rezept verwendet werden. Die Zubereitung besteht aus drei Attributen und einem Element. Es gibt immer eine Beschreibung, welche immer aus einem Text besteht. Deshalb ist der Typ hier *string*. Die Beschreibung habe ich als Element definiert, da die Möglichkeit besteht, dass sich die Struktur dieser ändert. Z.B. könnte es sein, dass man die Beschreibung später in verschiedene Schritte unterteil. Mit einem Attribut wäre das nicht möglich.

```
38        <xs:element name="zubereitung">
39            <xs:complexType>
40                <xs:all minOccurs="1">
41                    <xs:element name="beschreibung" type="xs:string" />
42                </xs:all>
43                <xs:attribute name="dauer" type="xs:integer" use="required" />
44                <xs:attribute name="brennwert" type="xs:integer" use="optional" />
45                <xs:attribute name="schwierigkeit" use="required">
46                    <xs:simpleType>
47                        <xs:restriction base="xs:string">
48                            <xs:enumeration value="simpel"/>
49                            <xs:enumeration value="normal"/>
50                            <xs:enumeration value="pfiffig"/>
51                        </xs:restriction>
52                    </xs:simpleType>
53                </xs:attribute>
54            </xs:complexType>
55        </xs:element>
```

Es gibt immer eine Zeitangabe für die Dauer. Diese wird immer in Minuten und ganzzahlig (integer) angegeben, deshalb habe ich es nicht für nötig befunden die Einheit der Zeitangabe extra zu speichern. Außerdem gibt es eine Brennwert-Angabe. Dies ist aber ein optionales Attribut, da nicht bei allen Rezepten ein Brennwert angegeben wurde. Desweiteren ist die Einheit dieser Angabe genau wie bei der Dauer immer gleich, deshalb habe ich auf eine zusätzliche Speicherung verzichtet.
Zuletzt ist auch immer ein Schwierigkeitsgrad angegeben. Dieser wird mit den drei Optionen *simpel*, *normal* und *pfiffig* angegeben. Deshalb habe ich dieses Attribut als *restriction* definiert.

Für die Kommentare habe ich ein Element kommentarliste erstellt, welches die Kommentare enthalten soll. Dieses Element kann genau wie die Zutatenliste nur einmal pro Rezept verwendet werden. Die Anzahl der Kommentare innerhalb der Liste ist aber unbegrenzt

```
56        <xs:element name="kommentarliste" minOccurs="1">
57            <xs:complexType>
58                <xs:sequence minOccurs="0" maxOccurs="unbounded">
59                    <xs:element name="kommentar">
60                        <xs:complexType>
61                            <xs:all minOccurs="1" maxOccurs="1">
62                                <xs:element name="username" type="xs:string" />
63                                <xs:element name="datum" type="xs:date" />
64                                <xs:element name="text" type="xs:string" />
65                            </xs:all>
66                            <xs:attribute name="id" type="xs:int" />
67                        </xs:complexType>
68                    </xs:element>
69                </xs:sequence>
70            </xs:complexType>
71        </xs:element>
```

Ein Kommentar besteht aus drei Elementen → Username des verfassenden Users, Zeitpunkt des Kommentars und der Nachricht und dem Kommentartext. Alle Attribute sind Pflichteingaben.
Ich habe diese Daten als Elemente definiert, da es z.B. sein kann, dass entschieden wird, die Struktur des Datums zu ändern, sodass z.B. der Tag, Monat und das Jahr in eigenständigen Elementen/Attributen gespeichert werden können. Das habe ich hier noch nicht modelliert, da das Datum bei der Eingabe auf der Webseite als Ganzes angegeben wird.
Zuletzt kommen noch zwei Attribute, welche sich direkt auf das Rezept beziehen. Die Bezeichnung des Rezeptes und eine ID. Die Angabe ist pro Rezept nur einmal möglich.

```
74                    <xs:attribute name="bezeichnung" type="xs:string" use="required"></xs:attribute>
75                    <xs:attribute name="id" type="xs:string" use="required"/>
76                </xs:complexType>
77            </xs:element>
78        </xs:sequence>
79    </xs:complexType>
80    </xs:element>
81</xs:schema>
```

Aufgabe 4

In dieser Aufgabe entwickeln Sie mit Hilfe des JAXB Frameworks ein Java-Programm, welches die XML-Datei aus der vorigen Aufgabe einlesen, modifizieren und ausgeben kann.

a)

Erzeugen Sie zunächst aus der Schema-Datei der vorherigen Aufgabe Java-Objekte. Nutzen Sie dazu den XJC-Befehl über das Terminal und fügen Sie die generierten Klassen ihrem Java-Projekt hinzu. Alternativ zur Terminal-Eingabe existiert ein JAXB Eclipse Plug-In welches hier herunter geladen werden kann: http://sourceforge.net/projects/jaxb-builder.
Dieses kann wie ein normales Plugin in Eclipse eingebunden werden. Zur Nutzung des Plugins klicken Sie mit der rechten Maustaste auf die Schema-Datei und wählen Sie aus dem Kontextmenü Generate => JAXB-Classes... und folgen Sie den weiteren Anweisungen in dem Dialogfenster.

Zur Erstellung der Java-Klassen anhand des XML-Schemas habe ich das JAXB-Plugin verwendet. Dabei bin ich, wie in der Aufgabenstellung beschrieben, vorgegangen. Die Dateien habe ich in einem separaten Paket abgelegt (aufgabe_4)

b)

Entwickeln Sie nun das Java-Programm. Es soll die XML-Datei öffnen, einlesen und die enthaltenen Daten über die Konsole wieder ausgeben. Benutzen Sie bitte bei der Bearbeitung der Aufgabe die generierten JAXB-Klassen aus der vorherigen Teilaufgabe.

Zum Auslesen der XML-Datei mit JAXB, benötigt man zunächst ein JAXBContext-Objekt dem der Java-Paket-Name von dem Paket übergeben wird, in dem sich die vom JAXB-Plugin generierten Klassen befinden. Dieses Objekt ermöglicht den Zugriff auf die JAXB API.

```
24  private static Rezeptliste rezepteAuslesen(File xmlFile)
25  {
66      try
67      {
68          JAXBContext context = JAXBContext.newInstance("aufgabe_4");
69          Unmarshaller unmarshaller = context.createUnmarshaller();
70          return (Rezeptliste)unmarshaller.unmarshal(xmlFile);
71      }
72      catch(Exception ex)
73      {
74          return null;
75      }
76  }
```

Anschließend benötigt man ein Unmarshaller-Objekt, welches mit Hilfe des JAXBContext-Objekts erstellt wird. Das Unmarshaller-Objekt wird dazu verwendet um den Inhalt der XML-Datei in die Java-Objekte zu parsen. Am Ende stehen die Daten der XML-Datei in dem Java-Objekt und können weiter verarbeitet werden.

c)

Erweitern Sie ihr Programm so, dass es möglich ist, über die Konsole neue Kommentare zu einem Rezept hinzuzufügen. Benutzen Sie auch hierfür die generierten JAXB-Klassen. Erstellen Sie ein Menü, dass in der Konsole angezeigt wird. Über dieses Menü sollen die Auswahl der Funktionen, zum Ausgeben der Daten und Erstellen neuer Kommentare, möglich sein.

```
131     private static void schreibeInDatei(Rezeptliste liste, File file) {
132         try
133         {
134             JAXBContext context = JAXBContext.newInstance("aufgabe_4");
135             Marshaller m = context.createMarshaller();
136             m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
137             m.marshal(liste, file);
138         }
139         catch(Exception ex)
140         {
141             System.out.println(ex.getMessage());
142         }
143     }
```

Zum bearbeiten der Daten in der XML Datei wird wie beim Auslesen ein JAXBContext-Objekt für den Zugriff auf die JAXB-API benötigt (Zeile 134). Anschließend benötigt man ein Marshaller-Objekt, dass die Daten des Java-Objekts so Formatiert, dass diese zurück in die XML-Datei geschrieben werden können (Zeile 135-137). Mit dieser Methode kann allerdings immer nur das komplette Root-Element zurück in die XML-Datei geschrieben werden. Es können keine einzeln geänderten Subelemente zurückgeschrieben werden.

Aufgabe 5

Diskutieren Sie, warum es sinnvoll ist Daten in Formaten wie XML oder JSON zu speichern. Stellen Sie außerdem die beiden Formate gegenüber und erläutern Sie kurz deren Vor- und Nachteile.

Durch die Verwendung von XML oder JSON kann man sicherstellen, dass es eine Trennung von Inhalt, Layout und Struktur gibt, was eine Webseite oder ein Programm flexibler macht. So kann man die Darstellung der Daten verändern ohne eine Änderung an der XML oder JSON Datei machen zu müssen, was den Aufwand gering hält.
Die Verwendung von XML oder JSON kann auch eine Zeitersparnis bzw. eine Reduzierung des Traffics mit sich bringen. Wenn z.B. ein Benutzer Daten für eine Webseite von einem Server anfordert, kann dieser die Daten z.B. als XML-Daten an den User senden. Dieser kann dann die Daten durchsuchen oder sortieren oder andere Operationen ausführen, ohne dass es nötig ist, die Seite komplett neu vom Server laden und über das Netz senden zu lassen.

XML	JSON
Gute Struktur , gut Lesbarkeit (+)	Gute Lesbarkeit (+)
Sieht ein bisschen aus wie HTML (+)	Kompakte aufs Minimum reduzierte Syntax (+)
Lizenzfrei und Plattformunabhängig (+)	Lizenzfrei und Plattformunabhängig (+)
Weit verbreitet (+)	Durch kompakte Syntax ist das Datenvolumen geringer als bei XML (+)
XML kann überprüft werden (+)	Reines JavaScript => lästige und zeitaufwendige Parsing-Vorgänge entfallen (Direktaustausch von JavaScript-Objekten (+)
Leicht erweiterbar (+)	Gewöhnungsbedürftige Syntax (-)
	Schlechtere Möglichkeiten, Metadaten und Kommentare zu integrieren (-)
	Nicht so verbreitet wie XML (-)
	Keine Namespaces (-)