Übungen zu Web- und XML-Technologien

Sommersemester 2018

Dr. Stefan Lüttringhaus-Kappel

Stand: 13. Juli 2018

Die Regeln

Abgabe Abgabe im eCampus spätestens am angegebenen Tag um 14:00 Uhr

Abgabeformat Bei jeder Aufgabe ist genau spezifiziert, welche Dateien abzugeben sind.

- Bei Bedarf können in einer Datei namens `README.txt` Nachrichten mitgeschickt werden.
- Laden Sie alle Dateien unter *Abgabe der Übungsaufgaben* einzeln, d. h. nicht als Archiv, in den eCampus hoch.
- Alle Dateien müssen in UTF-8 kodiert sein.
- Die bereits hochgeladenen Dateien werden Ihnen angezeigt.
- Es werden auch alte Versionen einer Datei angezeigt, falls Sie die Datei mehrmals hochgeladen haben. Löschen Sie bitte alte Versionen, damit Ihre Abgabe eindeutig ist.

Zu verwendende Software

1. Java: JDK 8
2. Apache Xerces2 Java Parser, Version 2.11.0. Benötigt werden `Xerces-J-bin.2.11.0.tar.gz` oder `Xerces-J-bin.2.11.0.zip`
3. Node.js (JavaScript runtime and engine), Version 9 oder höher.
4. Saxon (XSLT and XQuery Processor), in der Version für Java.
   Saxon Download Seite bei Sourceforge, Datei `SaxonHE9-8-0-11J.zip`.

Abgabe in 2er-Gruppen Ist diese Option bei einer Aufgabe vorgesehen und wird sie gewählt, müssen dennoch *alle* Teilnehmer die Abgabefunktion im eCampus aufrufen:

- Ein Teilnehmer lädt auf die oben beschriebene Art die Dateien hoch.
- Die weiteren Teilnehmer laden eine Datei `README.txt` hoch, die die Information enthält, welcher Teilnehmer die gemeinsame Lösung abgegeben hat.

Automatisches Testen Alle Abgaben, die ausführbaren Code oder sonstige maschinenlesbare Daten beinhalten, werden automatisch skriptgesteuert getestet. Jede Abweichung von vorgebenen Dateinamen und Programmfunktionalitäten führt zu Punktabzügen.

Die Tests werden mit den oben angegebenen Softwareversionen durchgeführt.

Aufgabe 1 (Wohlgeformte Dokumente)

```
1  <!-- Mein zweiter Versuch, XML zu schreiben :-) -->
2  <?xml version="1.1" encoding="UTF-8"?>
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4    <head id="e1">
5      <title>My First Document</title>
6    </head>
7    <body id="e1">
8      <p>Moved to <a href="http://example.org/">example.org</a>.</p>
9      <p lang="de"/>
10       Gr&uuml;ndlich pr&uuml;fen!.
11       Der Tabulator &#9; ist sehr hilfreich.
12     </p>
13     <foo>Another English sentence.</foo>
14     <table align="left" align="top">
15       <tr>
16         <td align=center>
17           1
18         </tr>
19       </td>
20     </table>
21   </body>
22   <data xmlns="http://example.org/">
23     Using <strong>strong</strong> and <html:em>em</html:em> elements
24     & (and) similar elements. More special elements: <1a>one</1a>, ...
25     <function prototype="<f>int</f>"/>
26   </data>
27 </html>
28 <appendix>
29   <p>Hier steht der Anhang</p>
30 </appendix>
```

In unser XML Dokument haben sich eine Reihe von Fehlern eingeschlichen. Finde sie alle! Löse die Aufgabe zuerst ohne Computer. Verifiziere das Ergebnis dann mit Hilfe eines XML-Parsers (siehe unten). Begründe kurz, worin jeder Fehler besteht.

Hinweis: Es geht in dieser Aufgabe nur um XML 1.1, weitere W3C-Empfehlungen sind nicht anzuwenden.

XML parsen mit Apache Xerces

Die benötigten Archive `xercesImpl.jar` und `xercesSamples.jar` müssen auf dem `CLASSPATH` liegen oder mit `-cp` beim Aufruf angegeben werden.

Ein Aufruf `java -cp .../xercesImpl.jar:.../xercesSamples.jar sax.Counter` (ohne weitere Argumente) gibt eine Hilfeseite aus, auf der man u. a. erfährt, wie man validierend parsen kann. (Pfade bitte anpassen. Unter Windows sind die Teilpfade mit `;` anstelle des `:` zu trennen.)

Nun parsen wir eine Datei namens `meine_datei.xml` (nicht validierend, ohne Namespaces):

```
java -cp ... sax.Counter -N -V meine_datei.xml
```

Aufgabe 2 (Wohlgeformte Dokumente)

```
1 <?xml version="1.1" encoding="UTF-8"?>
2 <test xml:lang='de'>
3   abc<![CDATA[<greeting/>]]>&#xA9; 2007
4 </test>
```

---

a) Untersuche, wie das gezeigte Dokument aus der XML-Grammatik, beginnend mit `document`, abgeleitet werden kann. Gib den Ableitungsbaum des Dokuments an. Liste dazu die verwendeten Produktionsregeln (z. B. top-down, left-to-right) sowie jeweils die daraus erzeugten Terminalsymbole (Strings) auf.

Hinweise:

Verwende die XML-1.1-Spezifikation.

Die Regeln für `Name` und triviale Konstrukte wie `S`, `Eq` usw. brauchen nur beim ersten Auftreten detailliert angewendet zu werden. Danach reicht jeweils ein kurzer Hinweis.

b) Zeichne den zu diesem Dokument gehörenden Baum (XML-Baum wie in der Vorlesung).

Aufgabe 3 (DTD)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE adressbuch SYSTEM "adrbuch.dtd">

<adressbuch>
  <adresse>
    <firma>
      <firmenname>Uni Bonn</firmenname>
      <abteilung>Informatik III</abteilung>
    </firma>
    <strasse>Römerstraße 164</strasse>
    <plz>53117</plz>
    <ort>Bonn</ort>
  </adresse>

  <adresse>
    <person titel="Herr">
      <vorname>Manfred</vorname>
      <nachname>Mustermann</nachname>
    </person>
    <strasse>Am Friedhof 11a</strasse>
    <plz>12345</plz>
    <ort>Irgendwo</ort>
  </adresse>

  <adresse land="uk">
    <person titel="Family">
      <nachname>Anyone</nachname>
    </person>
    <strasse>47 Eden Street</strasse>
    <ort>Cambridge</ort>
    <postcode>CB1 1JR</postcode>
  </adresse>

  <adresse land="us">
    <firma>
      <firmenname>Old Whiskey Brewery</firmenname>
    </firma>
    <strasse>8 Oak Avenue</strasse>
    <ort>Old Town</ort>
    <state>PA</state>
    <zip>95819</zip>
  </adresse>
</adressbuch>
```

(Fortsetzung der Aufgabe auf der nächsten Seite)

Wir benötigen eine DTD zu obigem Adressdokument, das ausschnittsweise bereits in der Vorlesung gezeigt wurde. Hier sind nun auch internationale Adressen berücksichtigt.

a) Schreibe eine DTD `adrbuch.dtd`, die möglichst gut zu obigem Dokument passt, d. h. die DTD soll genau die (sinnvollen) Kombinationen abdecken, die das Beispiel enthält.

b) Validiere das Dokument gegen diese DTD mit Xerces, wie unten beschrieben. Füge einen Testlauf bei, Dateiname: `testlauf.txt`

XML validieren mit Xerces

Siehe „XML parsen mit Apache Xerces bei Aufgabe 1. Setze zusätzlich die Option `-v` (*turn on validation*), also

```
java -cp ... sax.Counter -v adrbuch.xml
```

Aufgabe 4 (Namespaces)

```
<?xml version="1.0" encoding="UTF-8"?>
<e1 xmlns="http://X.de"
    xmlns:n1="http://Y.de"
    xmlns:n2="http://Z.de">
  <n1:e2
    a1="hallo"
    n2:a1=""
    xmlns:n3="http://U.de">
    <e3 xmlns="">
      <n3:e4/>
      <e5 xmlns:n2="http://V.de"/>
    </e3>
  </n1:e2>
  <e6>
    <e7 xmlns="http://U.de" a1="7">Hallo</e7>
  </e6>
</e1>
```

Gib für jedes Element und jedes Attribut in dem oben stehenden XML-Dokument den zugehörigen Namespace-URI an.

Aufgabe 5 (XML Information Set)

Selbststudium: Lies und verstehe die W3C Recommendation *XML Information Set*.

```
<?xml version="1.0" encoding="UTF-8"?>
<e1 xmlns="http://X.de"
    xmlns:n1="http://Y.de">
  <n1:e2 a="hello" n1:b="world">Uni Bonn</n1:e2>
</e1>
```

Gib alle Properties aller Information Items in obigem Dokument an.

Aufgabe 6 (XML Schema) 15 P. Termin: 2018-05-18 (Einzelabgabe oder in 2er-Gruppen)

```xml
<?xml version="1.1" encoding="UTF-8"?>

<library xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
         xsi:noNamespaceSchemaLocation='library.xsd'>

  <book>
    <author-ref>J.K.Rowling</author-ref>
    <title>Harry Potter und der Stein der Weisen</title>
    <language>de</language>
    <year>1998</year>
  </book>

  <book>
    <author-ref>J.K.Rowling</author-ref>
    <title>Harry Potter und die Kammer des Schreckens</title>
    <language>de</language>
    <year>1999</year>
  </book>

  <book>
    <author-ref>T.Pratchett</author-ref>
    <title>The Colour of Magic</title>
    <year>1983</year>
  </book>

  <book>
    <author-ref>T.Pratchett</author-ref>
    <author-ref>N.Gaiman</author-ref>
    <title>Good Omens: The Nice and Accurate Prophecies...</title>
    <language>en</language>
  </book>

  <book>
    <author-ref>R.Munroe</author-ref>
    <title>What If?</title>
    <language>en</language>
  </book>

  <book>
    <author-ref>G.Orwell</author-ref>
    <title>1984</title>
    <language>en</language>
  </book>

  <author id="J.K.Rowling">
    <last-name>Rowling</last-name>
    <first-name>Joanne K.</first-name>
  </author>

  <author id="N.Gaiman">
    <last-name>Gaiman</last-name>
    <first-name>Neil</first-name>
  </author>

```

```
55    <author id="T.Pratchett">
56      <last-name>Pratchett</last-name>
57      <first-name>Terry</first-name>
58    </author>
59
60    <author id="R.Munroe">
61      <last-name>Munroe</last-name>
62      <first-name>Randall</first-name>
63    </author>
64
65    <author id="G.Orwell">
66      <last-name>Orwell</last-name>
67      <first-name>George</first-name>
68    </author>
69
70  </library>
```

Gegeben sei eine Liste von bibliografischen Angaben in obigem Format. (Die Datei ist in eCampus als `library-example.xml` zu finden.)

(Fortsetzung der Aufgabe auf der nächsten Seite. . .)

a) Beschreibe das Datenmodell der bibliografischen Angaben als XML Schema in einer Datei `library.xsd`. Nutze die Möglichkeiten zur präzisen Beschreibung dieses Modells in XML Schema, insbesondere die eingebauten Datentypen. Das Schema soll möglichst gut zu Daten wie in obigem Dokument passen, d. h. alle (sinnvollen) Kombinationen abdecken, die das Beispiel enthält. Die Elementinhalte und Attributwerte sollen so weit wie möglich eingeschränkt werden.

b) Das Schema soll auch folgende *Constraints* enthalten:

- Die Werte der `author-ref`-Unterelemente jedes Buches sind paarweise verschieden. Realisiere das mit `xsd:unique`.
- Das `id`-Attribut der `author`-Elemente enthält für jeden Autor einen eindeutigen Wert. Realisiere das mit `xsd:key`.
- Jeder Wert von `author-ref` kommt auch in einem `id`-Attribut eines `author`-Elements vor. Realisiere das mit `xsd:keyref`.

c) Validiere das XML-Dokument gegen das Schema aus Teil a) und b). Verwende Apache Xerces, siehe Skript. Abgabe des Testlaufs (Xerces-Aufruf und Ausgabe im Terminal) als Datei `testlauf.txt`. Ebenfalls abzugeben ist eine erweiterte Datei `library.xml` mit sinnvollen Beispielen, die neben den obigen Testdaten mindestens je 3 weitere Autoren und Bücher enthält.

Insgesamt sind also abzugeben:

1. `library.xsd`
2. `library.xml`
3. `testlauf.txt`

Hinweise zur Abgabe

Die Abgaben werden automatisch getestet, siehe Hinweis auf Seite 1.

Ihre Abgabe testen wir zunächst mit dem Aufruf

```
java -cp ... sax.Counter -v -s -f library.xml
```

Danach parsen wir weitere von uns vorgebene Dateien, die ebenfalls auf Ihr `library.xsd` zugreifen:

```
java -cp ... sax.Counter -v -s -f unsere_testdatei1.xml
...
```

Aufgabe 7 (XML Schema)

```
1  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
2    <xsd:element name="p" type="type1"/>
3    <xsd:element name="bs" type="type2"/>
4
5    <xsd:complexType name="type1">
6      <xsd:sequence>
7        <xsd:element name="a" type="xsd:string"/>
8        <xsd:element ref="bs"/>
9        <xsd:element name="c" type="xsd:string" minOccurs="0"/>
10     </xsd:sequence>
11   </xsd:complexType>
12
13   <xsd:complexType name="type2">
14     <xsd:sequence>
15       <xsd:element name="a" type="xsd:int"/>
16       <xsd:element name="b" type="xsd:boolean" maxOccurs="4"/>
17     </xsd:sequence>
18   </xsd:complexType>
19 </xsd:schema>
```

Gib ein Beispiel für eine Instanz, die die maximale Anzahl von Elementen enthält.

Zeichne den Baum zu dieser Instanz, oder schreibe ihn seriell als XML-Dokument.

Aufgabe 8 (XPath)

```
1  <!ELEMENT book (div* | chapter*) >
2  <!ELEMENT div (chapter+) >
3  <!ELEMENT chapter (section|para)+ >
4  <!ELEMENT section (para+) >
5  <!ELEMENT para (#PCDATA|olist|em|kw)* >
6  <!ELEMENT olist (item+) >
7  <!ELEMENT item (#PCDATA|kw)* >
8  <!ELEMENT em (#PCDATA|kw)* >
9  <!ELEMENT kw (#PCDATA) >
10
11 <!ATTLIST book    xml:lang NMTOKEN #IMPLIED >
12
13 <!ATTLIST div     xml:lang NMTOKEN #IMPLIED
14                   id       ID      #IMPLIED >
15
16 <!ATTLIST chapter xml:lang NMTOKEN #IMPLIED
17                   id       ID      #IMPLIED >
18
19 <!ATTLIST section xml:lang NMTOKEN #IMPLIED
20                   id       ID      #IMPLIED >
21
22 <!ATTLIST para    xml:lang NMTOKEN #IMPLIED
23                   id       ID      #IMPLIED >
```

Wir erweitern das Beispiel aus der Vorlesung und betrachten XML-Dokumente zu obiger DTD. (Intendierte Verwendung: `em` für Hervorhebungen, `kw` zur Markierung von Keywords.)

a) Beschreibe zu jedem der folgenden XPath-Ausdrücke umgangssprachlich, welche Knoten er selektiert.

1. `/descendant::chapter[1]/*/para[1]`
2. `/descendant::para[1][@xml:lang='en']`
3. `/descendant::para[@xml:lang='en'][1]`
4. `/descendant::section[count(para)=2]`

b) Gib einen XPath-Ausdruck an, der alle `para`-Elemente selektiert, die direkt unter einem `section`-Element liegen.

c) Gib einen XPath-Ausdruck an, der alle `section`-Elemente selektiert, die ein `para`-Element als *child* haben.

d) Gib einen XPath-Ausdruck an, der alle `section`-Elemente selektiert, die genau ein `para`-Element als *child* haben.

e) Gib einen XPath-Ausdruck an, der alle `section`-Elemente selektiert, die ein `kw`-Element als *descendant* haben.

f) Gib einen XPath-Ausdruck an, der alle Elemente selektiert, die entweder selbst oder in einem *ancestor* das Attribut `xml:lang='de'` haben.

g) Wenn die Sprache entlang der Unterbäume „vererbt“ werden soll, reicht der vorherige Ausdruck noch nicht aus. Modifiziere den vorherigen Ausdruck, so dass nur das nächstliegende `xml:lang`-Attribut berücksichtigt wird. In folgendem Beispiel haben also nur die Elemente mit den Ids `d2`, `c3`, `d31`, `p313` und `p323` die Sprache `de`.

```
1  ...
2    <div id="d2" xml:lang="de">
3      <chapter id="c3">
4        <section id="d31">
5          <para id="p311" xml:lang="en">bla bla bla (9)</para>
6          <para id="p312" xml:lang="en">bla bla bla (10)</para>
7          <para id="p313">bla bla bla (11)</para>
8        </section>
9        <section id="d32" xml:lang="it">
10         <para id="p321">bla bla bla (12)</para>
11         <para id="p322" xml:lang="en">bla bla bla (13)</para>
12         <para id="p323" xml:lang="de">bla bla bla (14)</para>
13       </section>
14     </chapter>
15   </div>
16 ...
```

Hinweis: Die vordefinierte XPath-Funktion `fn:lang()` darf in dieser Aufgabe nicht verwendet werden.

XPath 2.0 mit Saxon

Erstelle eine Datei mit der gewünschten XPath- oder XQuery-Anfrage, z. B. `query.xpath` mit dem Inhalt:

```
/descendant::section[count(para)=3]
```

Rufe nun Saxon als XQuery-Prozessor auf:

```
java -cp saxon9he.jar net.sf.saxon.Query -s:book.xml query.xpath
```

(`saxon9he.jar` muss ggf. noch um den Installationspfad ergänzt werden.)

Mit `-s:book.xml` wird das anzufragende Dokument spezifiziert, `query` ist der Name der Datei, die die Anfrage enthält.

Aufgabe 9 (XSLT)

Wir wollen mit XSLT HTML-Dokumente erzeugen. Dazu ist der Text eines Buchs (XML-Dokument) zu formatieren, die Abschnitte sind zu nummerieren, und zusätzlich wird ein Inhaltsverzeichnis erzeugt.

Auf der nächsten Seite sehen Sie ein Beispieldokument. Beachten Sie bitte, dass die `section`-Elemente beliebig tief verschachtelt sein können.

Auf der übernächsten Seite sehen Sie die endgültige Ausgabe. Nutzen Sie Ihr CSS-Stylesheet, um das Aussehen Ihres Buches hieran anzupassen. (Kleine Abweichungen sind kein Problem.)

Hinweise

- Das erzeugte HTML5-Dokument samt CSS-Stylesheet muss validieren.
- Verwenden Sie `h1` für den Buchtitel, `h2`, ..., `h5` für die obersten vier `section`-Ebenen, und `h6` für alle tieferen.
- Nummerieren Sie die Abschnitte mit `xsl:number`.
- Zum Erzeugen der Links kann die Funktion `generate-id()` nützlich sein. Alternativ könnte man auch die erzeugten Abschnittsnummern als Ids nutzen.

(Fortsetzung der Aufgabe auf der nächsten Seite...)

```
<book>
  <title>Unsere Haustiere</title>
  <section>
    <title>Fische</title>
    <para>Fische brauchen immer Wasser</para>
  </section>
  <section>
    <title>Säugetiere</title>
    <section>
      <title>Katzen</title>
      <para>Katzen kratzen an den Möbeln.</para>
      <section>
        <title>Die Katze</title>
        <para>Hier Text einfügen... Und Bilder!</para>
      </section>
      <section>
        <title>Der Kater</title>
        <para>Der Kater ist der Mann bei den Katzen.</para>
      </section>
    </section>
  </section>
  <section>
    <title>Saurier</title>
    <section>
      <title>Tyrannosaurus rex</title>
      <para>Der T-rex ist ein liebes Haustier.</para>
    </section>
    <section>
      <title>Brontosaurus</title>
      <para>Der Brontosaurus passt nicht in deine Garage.</para>
    </section>
  </section>
  <section>
    <title>Zusammenfassung</title>
    <para>Das war doch schön, oder?</para>
  </section>
</book>
```

(Fortsetzung der Aufgabe auf der nächsten Seite...)

Unsere Haustiere

Inhaltsverzeichnis

1 Fische
2 Säugetiere
  2.1 Katzen
    2.1.1 Die Katze
    2.1.2 Der Kater
3 Saurier
  3.1 Tyrannosaurus rex
  3.2 Brontosaurus
4 Zusammenfassung

1 Fische

Fische brauchen immer Wasser

2 Säugetiere

2.1 Katzen

Katzen kratzen an den Möbeln.

2.1.1 Die Katze

Hier Text einfügen... Und Bilder!

2.1.2 Der Kater

Der Kater ist der Mann bei den Katzen.

3 Saurier

3.1 Tyrannosaurus rex

Der T-rex ist ein liebes Haustier.

3.2 Brontosaurus

Der Brontosaurus passt nicht in deine Garage.

4 Zusammenfassung

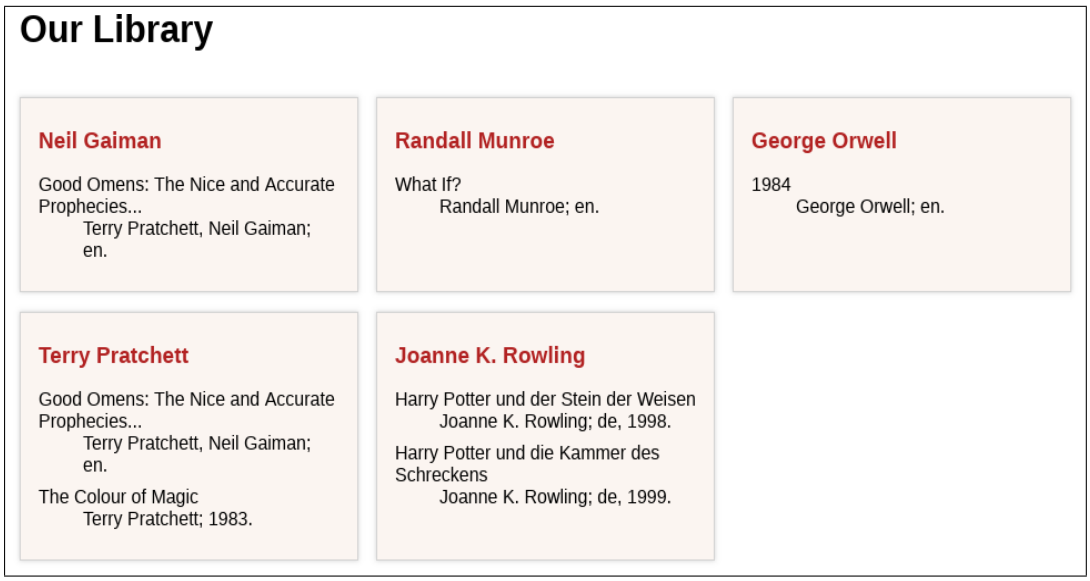
Das war doch schön, oder?

Aufgabe 10 (XSLT) 15 P. Termin: 2018-07-06
(Einzelabgabe oder in 2er-Gruppen)

Wir betrachten wieder die bibliografischen Angaben aus Aufgabe 6 und schreiben ein XSLT-Stylesheet `library2html.xsl`, das eine solche Liste in eine strukturierte HTML5-Darstellung überführt. Die Bücher werden nach Autoren gruppiert, dabei sind die Gruppen nach Autorennamen sortiert. Innerhalb jeder Gruppe sind die Bücher nach den Titeln sortiert.

Teste mit sinnvollen Testdaten (wie in Aufgabe 6 plus mindestens je 3 weitere Autoren und Bücher), die als Datei `library.xml` mit abgegeben werden. Die erzeugte Ausgabe ist als `library.html` beizufügen. Die Sprache des Buchs soll, soweit bekannt, als `lang`-Attribut an geeigneter Stelle im HTML-Code eingefügt werden.

Die semantisch korrekte HTML-Repräsentation kann auf Listen, Tabellen o. ä. basieren. Geeignete CSS-Regeln verschönern die Ausgabe:



Jedes einigermaßen sinnvolle eigene Design ist erlaubt, auch schlichtere Varianten. Außer `normalize.css` (optional) sind aber keine Stylesheets von Dritten erlaubt.

Technische Hinweise

- Verwende nicht `xsl:for-each-group`. Man kommt hier mit den grundlegenden Funktionen von XSLT aus.
- Die Funktion `current()` könnte nützlich sein, siehe die XSLT-Recommendation.
- Das erzeugte HTML5-Dokument und das CSS-Stylesheet müssen validieren.
- Das erzeugte HTML5-Markup muss auf allen Ebenen semantisch korrekt sein (gemäß dem HTML5-Standard).

Hinweise zur Abgabe

- Füge Saxon-Aufruf und -Ausgabe (Terminal) als `testlauf.txt` bei.
- Insgesamt sind abzugeben: `library2html.xsl`, `library.css`, evtl. `normalize.css`, `library.xml`, `library.html`, und `testlauf.txt`.

- Die Abgaben werden automatisch getestet, siehe Hinweis auf Seite 1.

XSLT 2.0 mit Saxon

Rufe Saxon als XSLT-Prozessor auf, z. B.:

```
java -cp saxon9he.jar net.sf.saxon.Transform -s:library.xml -o:library.html library2html.xsl
```

Mit `-s` wird das Eingabedokument und mit `-o` das Ausgabedokument angegeben. `saxon9he.jar` muss ggf. noch um den Installationspfad ergänzt werden.

Aufgabe 11 (XSLT)

Wir betrachten wieder bibliografische Angaben wie in Aufgabe 6. Nun sollen einige Auswertungen im *Textformat* (Plain Text) erstellt werden.

a) Schreibe ein XSLT-Stylesheet `lib-a.xsl`, das die Bücher alphabetisch nach Titel sortiert listet, in folgendem Format:

```
1  Good Omens: The Nice and Accurate Prophecies.... Terry Pratchett & Neil Gaiman
2  Harry Potter und der Stein der Weisen. Joanne K. Rowling
3  Harry Potter und die Kammer des Schreckens. Joanne K. Rowling
4  The Colour of Magic. Terry Pratchett
```

b) Schreibe ein XSLT-Stylesheet `lib-b.xsl`, das die Bücher nach Autoren gruppiert, in folgendem Format:

```
1  ** Gaiman, Neil **
2  Good Omens: The Nice and Accurate Prophecies.... Terry Pratchett & Neil Gaiman
3
4  ** Pratchett, Terry **
5  Good Omens: The Nice and Accurate Prophecies.... Terry Pratchett & Neil Gaiman
6  The Colour of Magic. Terry Pratchett
7
8  ** Rowling, Joanne K. **
9  Harry Potter und der Stein der Weisen. Joanne K. Rowling
10 Harry Potter und die Kammer des Schreckens. Joanne K. Rowling
```

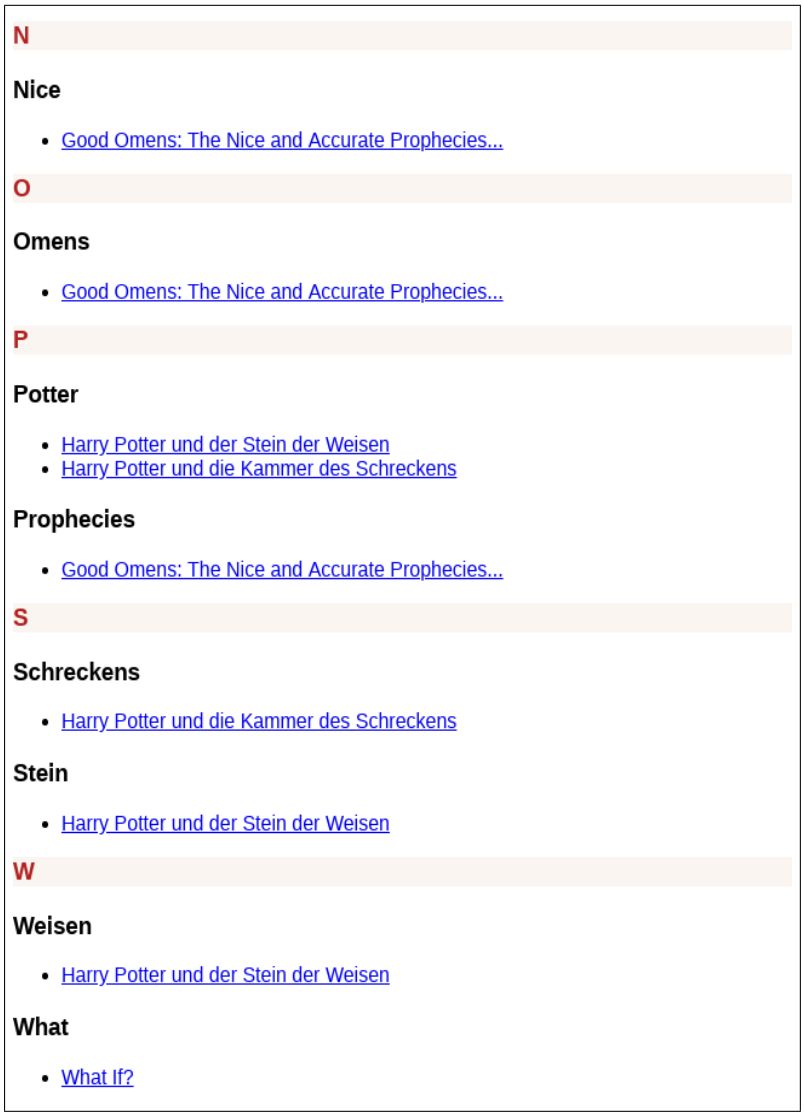
Die Gruppen sind dabei nach dem Namen der Autoren zu sortieren. Innerhalb jeder Gruppe wird nach Buchtitel sortiert.

c) Schreibe ein XSLT-Stylesheet `lib-c.xsl`, das Autoren und die Anzahl ihrer Bücher (als Balkendiagramm) auflistet. Sortiert wird dabei nach Anzahl der Bücher, die ein Autor geschrieben hat (größte Zahl zuerst). Außerdem sollen nur die ersten drei Autoren in dieser Rangliste ausgegeben werden. Format:

```
1  Pratchett, Terry   ##
2  Rowling, Joanne K. ##
3  Gaiman, Neil       #
```

Aufgabe 12 (XSLT) 20 P. Termin: 2018-07-13
(Einzelabgabe oder in 2er-Gruppen)

Wie in Aufgabe 10 wollen wir die bibliografischen Angaben aus Aufgabe 6 mittels eines XSLT-Stylesheets `library-index2html.xsl` in HTML5 formatieren und zusätzlich einen Index anhängen. Der Index enthält die in den Buchtiteln vorkommenden Wörter (im Folgenden *Indexbegriffe* genannt), abzüglich der in einer Stoppwortliste aufgeführten Wörter. Für jeden vorkommenden Anfangsbuchstaben eines Indexbegriffs wird eine Zwischenüberschrift generiert. Der Index könnte zum Beispiel so aussehen (Ausschnitt):



Die blauen Indexeinträge entsprechen den Buchtiteln, die den jeweiligen Indexbegriff enthalten, und sind Links auf die Bücherliste im oberen Teil des HTML5-Dokuments (siehe Aufgabe 10). Anfangsbuchstaben, Indexbegriffe und Indexeinträge sind alphabetisch sortiert.

Jedes einigermaßen sinnvolle eigene Design ist erlaubt. Außer `normalize.css` (optional) sind aber keine Stylesheets von Dritten erlaubt.

Technische Hinweise und Anforderungen

- Beachte auch die Hinweise in der Übungsstunde.

- Verwende `xsl:for-each-group`.
- Das erzeugte HTML5-Dokument und das CSS-Stylesheet müssen validieren.
- Das erzeugte HTML5-Markup muss auf allen Ebenen semantisch korrekt sein (gemäß dem HTML5-Standard).
- Die Stoppwortliste liegt in einer Plain-Text-Datei mit Namen `stopwords.txt` vor (Encoding UTF-8), in der die Wörter in Kleinschreibung enthalten und durch Whitespace getrennt sind, z. B.:

```
1 der die das des dem den
2 ein eine einer eines
3 und oder
4 a the
5 and or
6 of
```

- Die Buchtitel enthalten unerwünschte Sonderzeichen. Entferne diese mit folgendem Funktionsaufruf (`$e` ist der Textknoten oder das Element, das den Titel enthält):

  ```
  replace($e, '[^ \p{Nd}\p{L}-]', '')
  ```

  Schlage die Bedeutung des regulären Ausdrucks nach und erläutere in einer Plain-Text-Datei `replace.txt` in 3–4 Sätzen die Funktion des Ausdrucks sowie die Bedeutung seiner Bestandteile. Gib außerdem die URL des W3C-Dokuments an, in dem diese Details regulärer Ausdrücke definiert sind.
- Es kann sich als nützlich erweisen, eine Hilfsfunktion als Stylesheet-Funktion (`xsl:function`) zu definieren, die zu einem gegebenem Element (z. B. `title`) die Sequenz der darin enthaltenen relevanten Indexbegriffe berechnet.

Hinweise zur Abgabe

- Füge Saxon-Aufruf und -Ausgabe (Terminal) als `testlauf.txt` bei.
- Insgesamt sind abzugeben: `library-index2html.xsl`, `library.css`, evtl. `normalize.css`, `library.xml`, `library-index.html`, `stopwords.txt`, `replace.txt` und `testlauf.txt`.
- Die Abgaben werden automatisch getestet, siehe Hinweis auf Seite 1.

Aufgabe 13 (XQuery) 20 P. Termin: 2018-07-20
(Einzelabgabe oder in 2er-Gruppen)

```
<product>
  <sum>
    <fraction>
      <numerator>1</numerator>
      <denominator>2</denominator>
    </fraction>
    <fraction>
      <numerator>1</numerator>
      <denominator>3</denominator>
    </fraction>
    <fraction>
      <numerator>1</numerator>
      <denominator>4</denominator>
    </fraction>
  </sum>
  <fraction>
    <numerator>2</numerator>
    <denominator>3</denominator>
  </fraction>
</product>
```

Wir betrachten arithmetische Ausdrücke über Brüchen, die wie in obigem Beispiel in XML codiert sind:

- Der Operatorbaum besteht aus Summen (Element `sum`), Produkten (Element `product`) und Brüchen (Element `fraction`). Die Elemente können beliebig geschachtelt sein, Brüche tauchen aber nur als Blätter auf. Alle drei Elemente (`product`, `sum` oder `fraction`) können Wurzelelement des Baums sein.
- `product`-Elemente und `sum`-Elemente können beliebig viele (0 bis n) Kindelemente (`product`, `sum` oder `fraction`) haben.
- `fraction`-Elemente haben genau die Kindelemente `numerator` (Zähler) und `denominator` (Nenner). Diese wiederum haben den Typ `xs:integer`. Der Nenner ist niemals 0.

Schreibe ein XQuery-Modul `calculator.xqm`, das in dem Namespace `http://wob.iai.uni-bonn.de/exercises/xquery/calculator` folgende Funktionen definiert:

a) `evaluate($tree as element()) as element(fraction)`

Erhält einen Operatorbaum als Eingabe und wertet diesen zu einem Bruch aus (`fraction`-Element mit oben erklärter Struktur). Dabei sollen leere Produkte den Wert 1/1 und leere Summen den Wert 0/1 bekommen. Das Ergebnis soll vollständig gekürzt sein und der Nenner soll positiv sein. Aus 12/-6 wird also -2/1. Das Kürzen ist leicht mit dem Euklidischen Algorithmus zur Berechnung des ggT implementiert (https://de.wikipedia.org/wiki/Euklidischer_Algorithmus).

b) `print($tree as element()) as xs:string`

Stellt einen Operatorbaum als String dar. Brüche sollen dabei ungekürzt als Zähler/Nenner geschrieben werden. Bei Produkten werden die Faktoren durch den Operator `*` getrennt und bei Summen steht zwischen Summanden ein `+`. Summen müssen gegebenenfalls mit Klammern umschlossen werden, um die korrekte Auswertung zu repräsentieren („Punktrechnung geht vor Strichrechnung“). Ein leeres Produkt soll als `1` und eine leere Summe als `0` geschrieben werden.

Obiges Beispiel liefert somit `(1/2 + 1/3 + 1/4) * 2/3`.

c) Füge zu allen Funktionen aussagekräftige Testläufe mit verschiedenen Operatorbäumen bei. Dazu ist ein *Main Module* `calculator-test.xql` zu schreiben, das `calculator.xqm` importiert und alle Tests durchführt. Die Ausgaben erscheinen im Terminal und werden als `testlauf.txt` abgegeben.

Hinweise

1. Verwende zum Testen Saxon, wie in Aufgabe 8 erläutert. Der Aufruf ist also

   ```
   java -cp saxon9he.jar net.sf.saxon.Query calculator-test.xql
   ```

2. Saxon unterstützt in der kostenlosen Home Edition leider keine Higher-Order-Functions, so dass Funktionen aus XQuery 3.1 wie `fold-left` oder `fold-right` nicht genutzt werden können. Das kann dazu führen, dass ein Teil des Codes für Summen und Produkte dupliziert werden muss. Dies führt aber zu keinen Abzügen bei der Bewertung.

Abzugeben

`calculator.xqm`

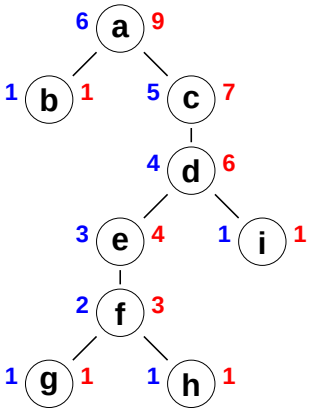
`calculator-test.xql`

`testlauf.txt`

`*.xml` Ggf. XML-Dateien mit Testdaten. Die Testdaten können aber auch in `calculator-test.xql` enthalten sein.

Die Abgaben werden automatisch getestet, siehe Hinweis auf Seite 1.

Aufgabe 14 (SAX)




```
1  <a>
2    <b/>
3    <c>
4      <d>
5        <e>
6          <f>
7            <g/>
8            <h/>
9          </f>
10       </e>
11       <i/>
12     </d>
13   </c>
14 </a>
```

Wir betrachten ein XML-Dokument als einen Baum von Elementen. Wie im oben stehenden Bild zu sehen, kann man in einem Baum die Tiefe (blau) und die Zahl der Elemente (rot) in Unterbäumen berechnen.

Schreibe ein Programm `TreeBalance.java`, das ein XML-Dokument einliest und das mit den berechneten Zusatzinformationen (Tiefe, Größe) versehene Dokument auf `System.out` ausgibt. Der Dateiname des auszuwertenden Dokuments wird als erstes Argument auf der Kommandozeile übergeben. Zum Beispiel: `java TreeBalance test1.xml`

Damit das Programm die Unterbäume nicht zwischenspeichern muss, sondern sofort wieder ausgeben kann, verpacken wir die berechneten Zusatzinformationen in ein neues Element, das als letztes Kindelement zur Wurzel des Unterbaums hinzugefügt wird. (Diese neuen Elemente werden nicht mitgezählt.) Um eine Unterscheidung der ursprünglichen und der neuen Elemente zu erreichen, verwenden wir für letztere einen Namespace. Alle weiteren Einzelheiten zu dem gewünschten Ausgabeformat sind aus der Ausgabe für obiges Beispiel ersichtlich:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <a>
3    <b><bl:info size="1" depth="1" xmlns:bl="http://iai.uni-bonn.de/xml"/></b>
4    <c>
5      <d>
6        <e>
7          <f>
8            <g><bl:info size="1" depth="1" xmlns:bl="http://iai.uni-bonn.de/xml"/></g>
9            <h><bl:info size="1" depth="1" xmlns:bl="http://iai.uni-bonn.de/xml"/></h>
10         <bl:info size="3" depth="2" xmlns:bl="http://iai.uni-bonn.de/xml"/></f>
11       <bl:info size="4" depth="3" xmlns:bl="http://iai.uni-bonn.de/xml"/></e>
12       <i><bl:info size="1" depth="1" xmlns:bl="http://iai.uni-bonn.de/xml"/></i>
13     <bl:info size="6" depth="4" xmlns:bl="http://iai.uni-bonn.de/xml"/></d>
14   <bl:info size="7" depth="5" xmlns:bl="http://iai.uni-bonn.de/xml"/></c>
15 <bl:info size="9" depth="6" xmlns:bl="http://iai.uni-bonn.de/xml"/></a>
```

Verwende Java 1.5 (oder höher) und parametrisiere die generischen Klassen (z. B. `Stack`) mit geeigneten Typen.

Fortsetzung der Aufgabe auf der nächsten Seite...

Hinweise

1. Die Klasse `TreeBalance` soll im *Default Package* liegen, also bitte keine `package`-Deklaration angeben.

2. In dieser Aufgabe wird ausschließlich SAX verwendet (kein DOM o.ä.).

3. Die Klasse `java.util.Stack` wird wahrscheinlich in der Implementierung benötigt.

4. Die Ausgabe von XML-Dokumenten *darf nicht* mit Java-Print-Methoden erfolgen! Stattdessen verwendet man einen sogenannten Serialisierer, der SAX-Events annimmt und auf einen Ausgabestrom schreibt (siehe Skript). Verwende hier die Klasse `com.sun.org.apache.xml.internal.serialize.XMLSerializer`.

5. Das Eingabedokument kann natürlich auch Namespaces verwenden, die wiederum unverändert in der Ausgabe erscheinen müssen. Alle Namespaces außer `http://iai.uni-bonn.de/xml` sind im Eingabedokument erlaubt. Im Eingabedokument kann natürlich auch der Präfix `bl` vorkommen.

6. Die Klasse `TreeBalance` leitet man sinnvollerweise aus einer geeigneten Oberklasse (aus dem JDK) ab. Welche Oberklasse eignet sich am besten? Wäre `XMLSerializer` als Oberklasse geeignet?

7. Ein stufenweises Vorgehen kann beim Erlernen neuer Klassenbibliotheken nützlich sein. So könnte die erste Programmversion lediglich die Eingabe parsen, dann nimmt man vorübergehend eine Testausgabe mit `System.err.println(...)` der gefundenen Elemente hinzu, dann rechnet man und gibt die Ergebnisse testweise aus, und schließlich schreibt man den Code zur Ausgabe der Ergebnisse via SAX.

Aufgabe 15 (DOM)

Wir betrachten wieder das Beispiel aus Aufgabe 14.

Schreibe ein Programm `TreeBalanceDom.java`, das ein XML-Dokument einliest und das mit den berechneten Zusatzinformationen (Tiefe, Größe) versehene Dokument auf `System.out` ausgibt. Der Dateiname des auszuwertenden Dokuments wird als erstes Argument auf der Kommandozeile übergeben. Zum Beispiel:

```
java TreeBalanceDom test1.xml
```

Wir wollen die Problemstellung nun mit dem Document Object Model (DOM) lösen. Der Vorteil dabei ist, dass sich im DOM der gesamte Baum im Direktzugriff befindet. Dadurch können wir auf einfache Weise die berechneten Werte als Attribute zu den Elementen hinzufügen. Dabei verwenden wir einen Namespace, um die neuen Attribute von den zuvor vorhandenen abzugrenzen. Alle weiteren Einzelheiten zum gewünschten Ausgabeformat sind aus der Ausgabe für obiges Beispiel ersichtlich:

```
<a xmlns:bl="http://iai.uni-bonn.de/xml" bl:depth="6" bl:size="9">
  <b bl:depth="1" bl:size="1"/>
  <c bl:depth="5" bl:size="7">
    <d bl:depth="4" bl:size="6">
      <e bl:depth="3" bl:size="4">
        <f bl:depth="2" bl:size="3">
          <g bl:depth="1" bl:size="1"/>
          <h bl:depth="1" bl:size="1"/>
        </f>
      </e>
      <i bl:depth="1" bl:size="1"/>
    </d>
  </c>
</a>
```

Hinweise

1. Die Klasse `TreeBalanceDom` soll im *Default Package* liegen, also bitte keine `package`-Deklaration angeben.

2. Benutze die Klasse `DocumentBuilderFactory`, um eine Instanz von `DocumentBuilder` zu erhalten, mit der man direkt ein XML-Dokument in ein `Document`-Objekt (inkl. Unterbaum) parsen kann.

3. Zur Ausgabe des XML-Dokuments verwende — wie in der Vorlesung gezeigt — einen `javax.xml.transform.Transformer` mit einem `javax.xml.transform.stream.StreamResult`. (Die Ausgabe von XML-Dokumenten *darf nicht* mit Java-Print-Methoden erfolgen!)

4. Ein stufenweises Vorgehen ist beim Erlernen neuer Klassenbibliotheken nützlich. So könnte die erste Programmversion lediglich die Eingabe parsen, dann nimmt man vorübergehend eine Testausgabe mit `System.err.println(...)` der gefundenen Elemente hinzu, dann rechnet man und gibt die Ergebnisse testweise aus, und schließlich fügt man die Serialisierung hinzu.

5. Modifizieren Sie Ihre Lösung so, dass sie auch mit Eingaben umgehen kann, die den Präfix `bl` bereits verwenden.