

2. Übungsblatt (SS 2019)

3.0 VU Semistrukturierte Daten

Informationen zum Übungsblatt

Allgemeines

Thema dieses 2. Übungsbeispiels ist der Zugriff auf das in Beispiel 1 erstellte XML Dokument mittels APIs und Abfragesprachen. Dabei wird mittels XSLT eine Übersichtsseite in HTML erstellt, mittels XQuery eine Abfrage ausgewertet und über die Java APIs SAX und DOM auf das Dokument zugegriffen.

Als Grundgerüst finden Sie im TUWEL ein ZIP-Archiv mit einer grundlegenden Projektstruktur und Ressourcen die Sie als Vorlage für diese Übung nutzen müssen. Weiters wird Ihnen in diesem Gerüst ein ant-Skript zur Verfügung gestellt (build.xml), welches das Testen und die Abgabe der Übung vereinfacht.

Beachten Sie:

- Alle Dateien und Pfade beziehen sich auf diese Vorlage.
- Genauere Instruktionen zur Verwendung der ant Targets finden Sie in den einzelnen Übungsbeschreibungen.
- Die Verwendung dieser Vorlage ist verpflichtend.
- Um sicherzugehen, dass ihre Implementierung auf unserem System lauffähig ist, testen Sie Ihre Lösung mit Java 8.

Das Übungsblatt enthält 3 Aufgaben, auf welche Sie insgesamt 15 Punkte erhalten können.

Abgabe

Um ein abgabefertiges Archiv `ssd-exercise2-ss19.zip` zu erzeugen, führen Sie folgenden Befehl aus:

```
ant zip
```

Deadlines

bis 6.6. 12:00 Uhr Upload der Abgabe über TUWEL
ab 10.6. 12:00 Uhr Anmeldung zu einem Kontrollgespräch

Kontrollgespräch

Im Rahmen des Kontrollgespräches wird nicht nur die Korrektheit, sondern vor allem das Verständnis der Konzepte überprüft. Durch die Übung sollen sowohl Ihre praktische Problemlösungskompetenz als auch das theoretische Wissen über Semistrukturierte Daten gefördert werden. Sie müssen daher bei Ihrem Kontrollgespräch in der Lage sein, nicht nur Ihre Beispiele zu erklären, sondern ebenfalls zeigen, dass Sie die in der Vorlesung behandelte Theorie zu diesen Beispielen ausreichend verstanden haben. Dies soll Ihnen die Vorbereitung für die Prüfung erleichtern und so können Sie Ihr Wissen während der Kontrollgespräche selbst testen und gegebenenfalls vertiefen.

Die Bewertung Ihres Übungsblattes basiert zum überwiegenden Teil auf Ihrer Leistung beim Kontrollgespräch! Es ist daher im Extremfall durchaus möglich, dass eine korrekte Abgabe

mit 0 Punkten bewertet wird. Insbesondere werden nicht selbstständig gelöste Abgaben immer mit 0 Punkten bewertet!

Erscheinen Sie in Ihrem eigenen Interesse bitte pünktlich zum Kontrollgespräch, da andernfalls nicht garantiert werden kann, dass Ihre gesamte Lösung in der verbleibenden Zeit beurteilt werden kann.

Bringen Sie bitte Ihren Studentenausweis zum Kontrollgespräch mit. Ein Kontrollgespräch ohne Ausweis ist nicht möglich.

Tutorensprechstunden (freiwillig)

Rund eine Woche vor der Abgabedeadline bieten die TutorInnen Sprechstunden an. Falls Sie Probleme mit oder Fragen zum Stoff des Übungsblattes haben, es Verständnisprobleme mit den Beispielen oder technische Fragen gibt, kommen Sie bitte einfach vorbei. Die TutorInnen beantworten Ihnen gerne Ihre Fragen zum Stoff, oder helfen Ihnen bei Problemen weiter.

Ziel der Sprechstunden ist es, Ihnen beim **Verständnis des Stoffs** zu helfen, nicht, das Übungsblatt für Sie zu rechnen, oder die eigenen Lösungen vorab korrigiert zu bekommen.

Die Teilnahme ist vollkommen freiwillig — Termine und Orte der Tutorensprechstunden finden Sie in TUWEL.

Weitere Fragen – TUWEL Forum

Sie können darüber hinaus das TUWEL Forum verwenden, sollten Sie inhaltliche oder organisatorische Fragen haben.

Aufgaben

Aufgabe 1 (XSLT) [6 Punkte]

Erstellen Sie ein XSLT Dokument `src/libraryarea-overview.xsl`, das ein (bezüglich des DTD Schemas `library.dtd`) gültiges library XML Dokument (`library-dtd.xml`) in ein HTML Dokument transformiert. Das HTML Dokument soll eine Überblicksdarstellung aller libraryAreas sein.

Um ihnen die Formatierung zu erleichtern, haben wir das Grundgerüst in dem Dokument `libraryarea-overview.xsl` bereits angelegt. Legen Sie darin folgende Templates an:

- Ein *named* Template `inventory`, das für jede `libraryArea` und jeden `woodenCase`, `steelCase` oder `lockedCase` folgendes ausgibt:
 - Die Anzahl aller `book`-Elemente deren `location` Kindelemente auf diese `libraryArea` verweisen, die Anzahl aller `woodenCase`, `steelCase` oder `lockedCase`-Elemente, die Nachfolger dieser `libraryArea` sind und zuletzt auch den Wert der Summe von allen `shelves` Attributen von `woodenCase`, `steelCase` oder `lockedCase`-Elementen, die in dieser `libraryArea` liegen.
 - Danach wird eine Tabelle angelegt. In dieser Tabelle soll für jedes `woodenCase`, `steelCase` oder `lockedCase` welches ein Kindelement der `libraryArea` ist, ihr Template ausgerufen werden.

In der ersten Zeile (im Tableheader) wird, “title”, “author”, “ISBN”, “categories”, “case” und “shelf” ausgegeben.

Im Rest der Tabelle soll jedes **book** Element, dass sich in dieser LibraryArea befindet, eine Zeile in dieser Tabelle darstellen. Achten Sie zusätzlich darauf, dass die Template Aufrufe der **woodenCase**, **steelCase** oder **lockedCase** Elemente sortiert passiert, aufsteigend nach dem **id** Attribut. Konkret: Ein **book** in einem Case mit **id** 1 sollte vor einem **book** Element in einem Case mit der **id** 2 ausgegeben werden.

- Ein Template für **libraryArea** Elemente, das als Überschrift den Namen der libraryArea ausgibt und dann das *named* Template **inventory** aufruft.
- **Ein** Template für **woodenCase**, **steelCase** und **lockedCase** Elemente, das für alle **book**-Elemente mit einem **location**-Kindelement, das auf dieses **woodenCase**, **steelCase** oder **lockedCase** Element verweist, ihr Template aufruft.
- Ein Template für **book** Elemente, das eine Tabellenzeile (**<tr>**) ausgibt, und darin den Titel (**titel**), den Autor (**author**), die ISBN (**isbn**), alle Kategorien (**category**) dieses Elementes (mit Komma getrennt) und auch das Regal (**case**) und das Fach (**shelf**), in dem es sich dem **location** Element nach befindet, ausgibt. Achten Sie darauf, dass falls gewisse Kindelemente nicht vorhanden sind, dies dennoch explizit eine Ausgabe erzwingt (z.B.: "... nicht vorhanden").

Eine Beispielausgabe finden Sie im Template unter **resources/libraryarea-overview.html**.

Aufruf: Führen Sie **ant run-xslt** aus. Es wird ein HTML Dokument **libraryarea-overview.html** im Verzeichnis **output** erzeugt. Dieses können Sie im Browser öffnen um die Ausgabe Ihres Stylesheets zu überprüfen.

Beachten Sie: Eine korrekter Output in **output/libraryarea-overview.html** ist Voraussetzung um alle Punkte bei dieser Aufgabe zu erhalten. Für ein syntaktisch falsches Stylesheet werden 0 Punkte vergeben!

Ihre Abgabe besteht aus:

- Einem XSLT Dokument: **src/libraryarea-overview.xsl**

Aufgabe 2 (XQuery) [3 Punkte]

Erstellen Sie eine XQuery **src/xquery.xq**, die eine (dem Schema **library.dtd** aus Beispiel 1 entsprechende) XML Datei als Eingabe erhält, und für jede libraryArea, mit 2 oder mehr Regalen vom Typ **woodenCase**, alle Regale vom Typ **steelCase**, sortiert nach dem **id** Attribut, ausgibt. Für diese steelCases soll die Anzahl aller enthaltenen Bücher ausgegeben werden. Dies wird gefolgt von allen **book** Elementen, die sich in diesem steelCase befinden, wobei für jedes **book** Element nur der **title** als Textelement ausgegeben wird. Zusätzlich geben Sie noch die Anzahl an Fächern (**shelves**) in diesem steelCase aus.

Die Ausgabe sollte wie folgt aussehen:

```
<specialSteelCases>
  <area name="hall1">
    <steelCase id="1">
      <bookCount>2</bookCount>
      <book>Set theory and the continuum problem</book>
      <book>To mock a mockingbird</book>
      <shelves>5</shelves>
```

```

    </steelCase>
  </area>
</specialSteelCases>

```

Stellen Sie sicher, dass sich Ihre XQuery `xquery.xq` mittels `ant run-xquery` ausführen lässt. Dabei wird die Ausgabe der Anfrage in `src/xquery.xq` im XML-Dokument `output/xquery-out.xml` erzeugt.

Beachten Sie: Eine korrekte Ausgabe in `output/xquery-out.xml` ist Voraussetzung um alle Punkte bei dieser Aufgabe zu erhalten. Für eine syntaktisch falsche XQuery werden 0 Punkte vergeben!

Ihre Abgabe besteht aus:

- Einer XQuery: `src/xquery.xq`

Aufgabe 3 (DOM/SAX) [6 Punkte]

Das Ziel dieser Aufgabe ist es ein `archive.xml` Dokument mittels SAX zu parsen und dabei das `library-dtd.xml` Dokument mittels DOM so zu ändern, dass die Informationen des `archive.xml` Dokuments in das `library-dtd.xml` Dokument übernommen werden. Das `archive.xml` Dokument hat folgende Struktur:

- Die Wurzel ist ein `archive` Element.
- Danach folgen beliebig viele `book` Elemente. Diese haben als Attribute `author` - einen Autor, `titel` - ein Titel, `year` - das Erscheinungsjahr, `description` - eine Beschreibung, `isbn` - eine ISBN und `categories`. `categories` wiederum besteht aus `category` Elementen.

Ein Beispiel für ein `archive.xml` Dokument findet sich im Template unter dem Verzeichnis `resources/archive.xml`.

Ihr Ziel ist es nun die Bücher in diesem Archiv mit dem Dokument `library-dtd.xml` zu integrieren, d.h. die entsprechenden, neuen `book` Elemente anzulegen. Das daraus resultierende Dokument soll noch immer bezüglich der gegebenen `library.dtd` validiert werden können.

Für die `location` dieser neuen `book` Elemente soll auch eine neue `libraryArea` namens "archive" angelegt werden (*falls Sie nicht bereits existiert!*), und alle `book` Elemente aus dem Archiv sollen das selbe Regal (vom Typ `lockedCase`) und das selbe Fach (`shelf`) als `location` haben.

Hinweis: Um die höchste bereits vergebene ID aller `book` Elemente zu berechnen, können Sie den XPath Ausdruck "`max(//book/number(substring(@id,2)))`" verwenden. Vergessen Sie nicht das Zeichen 'b' an neue IDs anzuhängen (z.B.: "b11" statt "11").

Beschreibung der Klassen

Das Template stellt zwei Klassen bereit. Die Klasse `SSD` beinhaltet die eigentliche Programmlogik. Die Klasse `LIBRARYHANDLER` stellt einen SAX Handler bereit, der `archive.xml` Dokumente parst und dabei ein `library-dtd.xml` Dokument manipuliert.

Hier ist eine detaillierte Beschreibung der Klassen:

- Klasse: SSD
 - *Variablen:*
 - * `static DocumentBuilderFactory documentBuilderFactory`: speichert eine Instanz einer `DocumentBuilderFactory`.
 - * `static DocumentBuilder documentBuilder`: speichert eine Instanz eines `DocumentBuilder`.
 - *Methoden:*
 - * `static void main(String [] args) throws Exception`: Einstiegspunkt für das Programm. Parst die Kommandozeilenargumente und ruft die Methoden `initialize` und `transform` auf.
 - * `static void initialize() throws Exception`: Initialisiert die `documentBuilderFactory` und die `documentBuilder` Variablen.
 - * `static void transform(String inputPath, String archivePath, String outputPath) throws Exception`: Sie müssen diese Methode implementieren. Zuerst müssen Sie ein DOM Objekt (`Document`) aus dem Dateinamen, der in der `inputPath` Variable gespeichert ist, erstellen. Dann müssen Sie den SAX Parser erstellen und initialisieren um das Dokument zu parsen, das durch den in der `archivePath` Variable spezifizierten Dateinamen gegeben ist. Dazu erstellen Sie eine Instanz der `LIBRARYHANDLER` Klasse, welche das zuerst erstellte `Document` Objekt als Argument im Konstruktor benötigt. Jetzt parsen Sie das `archive.xml` Dokument. Der `LIBRARYHANDLER` wird das Dokument ändern. Zuletzt, soll das Dokument mit der Methode `getDocument()` von der Klasse `LIBRARYHANDLER` abgerufen und mittels de's Schemas validiert werden. Das Dokument wird in die durch die `outputPath` Variable spezifizierte Datei gespeichert.
 - * `static void exit(String message)`: Diese Methode kann benutzt werden um eine Fehlermeldung auszugeben und das Programm zu verlassen.
- Klasse: `LIBRARYHANDLER`
 - *Variablen:*
 - * `static XPath xPath`: diese `XPath` Instanz kann benutzt werden, um `XPath` Abfragen über ein XML Dokument zu evaluieren
 - * `Document libraryDoc`: speichert eine DOM Repräsentation eines `library-dtd.xml` XML Dokument
 - * `String eleText`: speichert den Textinhalt von XML Elementen
 - * Sofern nötig können Sie noch weitere Variablen in dieser Klasse spezifizieren.
 - *Methoden:*
 - * `LibraryHandler(Document doc)`: Der Konstruktor der Klasse erhält ein `DOM Document` als Argument.
 - * `void characters(char[] text, int start, int length)`: SAX ruft diese Methode auf um den Textinhalt eines Elementes abzurufen. Dieser Text wird in der `eleText` Variable gespeichert.
 - * `Document getDocument()`: Gibt das in `libraryDoc` gespeicherte XML Dokument zurück.

- * Geben Sie hier weitere Methoden an, um das `archive.xml` Dokument zu parsen (z.B.: `startElement`, etc.) und ändern Sie das `libraryDoc` Objekt.

Aufruf des Programms und Ausgabe

Der Code in `src/ssd/SSD.java` kann mittels drei Kommandozeilenargumenten aufgerufen werden. Das erste gibt ein System Dokument an (z.B. `library-dtd.xml` aus Übungsblatt 1). Das zweite ein `archive.xml` Dokument (z.B. `resources/archive.xml`). Das letzte einen Dateinamen für die Ausgabe (z.B. `output/library-out.xml`). In der Ant-Datei wurden zwei Targets vorkonfiguriert:

- **ant run-dry:** ruft das Programm mit dem `textttarchive.xml` Dokument `resources/archive.xml`, dem System Dokument `resources/library-dtd.xml` und speichert die Ausgabe in `output/library-out.xml`.
- **ant run-persistent:** ruft das Programm mit dem `textttarchive.xml` Dokument `resources/archive.xml`, dem System Dokument `resources/library-dtd.xml` und speichert die Ausgabe in `resources/library-dtd.xml`. Somit wurde die Änderung dauerhaft im XML Dokument gespeichert.

Nach den Abgabegesprächen von Beispiel 1 wird die Datei `resources/library-sample-out.xml` zum Template hinzugefügt, welche die Beispielausgabe des Programms enthält, wenn die Datei `resources/library.xml` auf die Datei `resources/library-dtd.xml` angewendet wird.

Hinweis: XPath Abfragen können über ein XML Dokument (oder relativ zu einem Element) mittels folgendem Code evaluiert werden:

```
XPathExpression xpathExpr = XPath.compile("//libraryArea");
```

```
NodeList playerList = (NodeList)xpathExpr.evaluate(libraryDoc, XPathConstants.NODESET);
```

Ihre Abgabe besteht aus:

- Zwei Java Quell-Dateien: `SSD.java` und `LibraryHandler.java`