

LZ HTML – PA FORMULARERSTELLUNG

- (1) Es ist ein HTML-Formular zu erstellen, durch das die Suche nach POIs (PointOfInterest) gemäß ihrer Kategorie ermöglicht wird. Die Suchmaske besteht aus
 - (1) Feld zur Eingabe eine POI-Kategorie
 - (2) Button zum Start der Suche nach den jeweiligen POIs



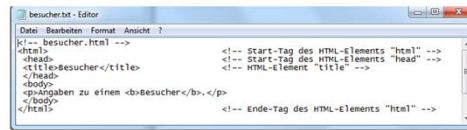
In dieser Praktischen Aufgabe ist ein einfaches HTML-Formular unter Nutzung von HTML (und gewissen Formular-Befehlen) zu erstellen. Die Aufgabe ist wieder aus dem Kontext des KIT-Smart-Campus (KIT-SC) und den darin genutzten Menge von PointOfInterest (POI) gewählt.

KIT-SC
POI

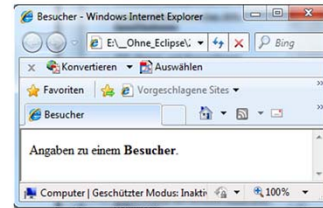
KIT-Smart-Campus
PointOfInterest

- (1) Das Dokument
"besucher.html" ist mit
einem Texteditor zu
erstellen und in einem
Web-Browser darzustellen

- (1) Erstellung mit einem
Standard-Editor
- (2) Starten des Internet
Explorer
- (3) Laden des HTML-
Dokuments



```
besucher.html - Editor
Datei Bearbeiten Format Ansicht ?
<!-- besucher.html -->
<html>
<head>
<title>Besucher</title>
</head>
<body>
<p>Angaben zu einem <b>Besucher</b></p>
</body>
</html>
```



(1) Diese erste Praktische Aufgabe dieser Kurseinheit ist eine Art "Hello World"-Beispiel zu HTML.

(1.1) Beispielsweise unter Windows: "Start" -> "Programme" -> "Zubehör" -> "Editor"

Hinweis: Der Dateiname ist gemäß folgender Konvention zu bilden:

<Nummer der WASA-Kurseinheit>_<Name der Praktischen Aufgabe>

(in diesem Fall also: "2-3_DOKUMENTERSTELLUNG")

(1.3) Das Laden des Dokuments ist über den im Menü "File" angebotenen Befehl "Open" möglich [Li05-HTML:3] .

(1.2) (1.3) Alternativen zum Aufruf des Dokuments:

(i) Umbenennen der Endung von ".txt" in ".html" führt dazu, dass das Dokument automatisch mit dem Web-Browser (z.B. Internet Explorer) geöffnet wird.

(ii) Angabe des Pfades und des Dateinamens oben im Pfad- und Dateieingabe-Feld (neben dem Explorer-Logo): Der Browser interpretiert den Inhalt nur dann als HTML, wenn die Datei die Endung ".html" besitzt. Im Falle von ".txt" wird der HTML-Quelltext im Browser-Fenster angezeigt.

[Li-HTML] Y. Daniel Liang: HTML and XHTML Tutorial, www.prenhall.com/liang,
<http://cs.armstrong.edu/liang/intro7e/supplement/Supplement6aHTML.pdf>.

```
1. <!DOCTYPE html >
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
5.     <title>Insert title here</title>
6.   </head>
7.   <body>
8.     <form method="POST" action="poiManager">
9.       <p>
10.        <label for="poiCategoryInput"> Enter POI category:
11.      </label>
12.      <input type="text" name="poiCategory" id="poiCategoryInput" />
13.    </p>
14.    <input type="submit" value="Search" />
15.  </form>
16. </body>
17.</html>
```

2-3_FORMULARERSTELLUNG.html

Die HTML-Datei realisiert eine Suchmaske durch die ein Benutzer Anfragen an das Servlet leisten kann.

(1. - 7.) Standard -HTML.

(8.) Durch diese Zeile wird die Weiterbehandlung der Eingabe (Übertragung und Verarbeitung) festgelegt.

(10.) Diese HTML-Zeile generiert ein Eingabefeld für die POI-Kategorie. Der Wert des "name"-Attributs wird benutzt, um die Daten zu identifizieren. Im Servlet kann man die Daten aus der POST-Anfrage entnehmen.

(14.) Diese HTML-Zeile generiert den Submit-Knopf, der den Browser veranlasst, eine POST-Anfrage an das Servlet, das im Attribut "action" des "form"-Elements adressiert wird, zu schicken.

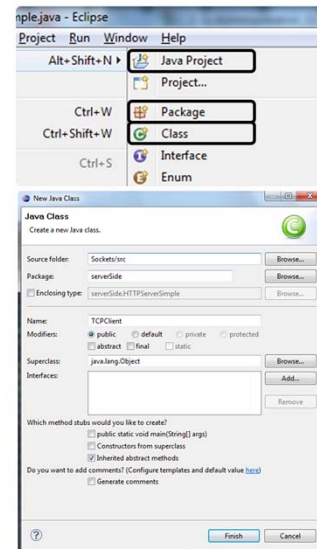
- (1) Es ist in der Sprache Java ein POI-Manager zu entwickeln
 - (1) Einlesen einer als XML-Dokument vorliegenden POI-Liste
 - (2) Umwandlung jedes POIs in Großbuchstaben
 - (3) Ausgabe der geänderten POI-Liste als XML-Dokument

Die Umwandlung von XML in eine entsprechende Datenstruktur (hier: Java-Objekte) der höheren Programmiersprache, in der die durch XML beschriebenen Daten verarbeitet werden und anschließend wieder in XML überführt werden, gehört zu den elementaren Aufgaben der XML-Programmierung. Diese Praktische Aufgabe zeigt, wie eine solche Umwandlung mittels der zwei wichtigsten XML-Technologien (SAX zur Umwandlung von XML in Java und DOM für die Rückumwandlung) am Beispiel einer Liste von POIs (PointOfInterest) in Java programmiert werden kann.

DOM	Document Object Model (XML)
POI	PointOfInterest
SAX	Simple API for XML

PA XML-PROGRAMMIERUNG – Anlegen eines neuen Projekts

- (1) "File" → "New"
 - (1) Java-Projekt anlegen und "2-3_XML-PROGRAMMIERUNG" nennen
- (2) Pakete "data", "model", "poiManagement" und "test" anlegen
- (3) Klassen bzw. Dokumente erstellen
 - (1) "data"-Paket: Dokumente "POIList.xml" und "POIList.xsd"
 - (2) "model"-Paket: Klasse "POI"
 - (3) "poiManagement"-Paket: Klassen "XMLPOIManager", "SAXReader", "DOMWriter"
 - (4) "test"-Paket: Klasse "XMLTester"



5

14.11.2013 WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

(1) Um eine bessere Übersicht im Workspace zu haben, ist für jede Praktische Aufgabe ein neues Projekt mit der Namenskonvention "<Kurseinheitnummer>_<PA-Kurzbezeichnung>" anzulegen.

(2) Die Programme in Java werden als Mengen von Paketen organisiert (Anlegen durch Rechtsklick auf das Projekt und Auswahl von "New" -> "Package"). Jedes Paket hat eigene Namen für Typen (z.B. Klassen), um Namenskonflikte zu vermeiden. Hierdurch lassen sich Klassen, die der gleichen Kategorie angehören oder ähnliche Funktionalität haben, organisieren. Ein Paket kann in einem Dateisystem abgelegt werden (als Ordner) oder in einer Datenbank. Pakete werden in der Regel mit Hilfe eines hierarchischen Benennungsmusters definiert, dessen Hierarchieebenen durch Punkte (.) (sprich "dot") getrennt sind [JLS-C7].

(2.1) Die Benutzung des Standardpaketes ist nicht zu empfehlen. Daher werden separate Pakete für (i) die Daten (XML-Dateien), (ii) das die Domäne beschreibende Modell und (iii) die Anwendungslogik vorgesehen. Da eine verteilte Anwendung entwickelt wird, ist die Einteilung in Pakete besonders sinnvoll.

(3.1) Im "data"-Paket liegen die XML- und XSD-Dateien. Diese Pakete könnte man sich als Datenquelle/Datenbank vorstellen.

(3.2) Im "model"-Paket liegt der Code, der die Geschäftsobjekte der Domäne, hier die Klasse "POI", beschreibt.

(3.3) Im "poiManagement"-Paket liegt der Code, der die Anwendungslogik enthält.

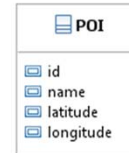
(3.4) Im "test"-Paket liegt der Code zur Ausführung des "XMLPOIManager".

JSC Java Language Specification

[JLS-C7] Java Language Specification Chapter 7, http://java.sun.com/docs/books/jls/third_edition/html/packages.html

PA XML-PROGRAMMIERUNG – Die POI-Klasse

- (1) Eine einfache POI-Klasse wird modelliert, die die wichtigsten Elemente beinhaltet
- (2) Das zu erstellende XSD-Dokument wird sich nach dieser Klasse richten



```
1. package model;
2. public class POI {
3.     private int id;
4.     private String name;
5.     private float latitude;
6.     private float longitude;
7.     /*Getter und Setter*/
8. }
```

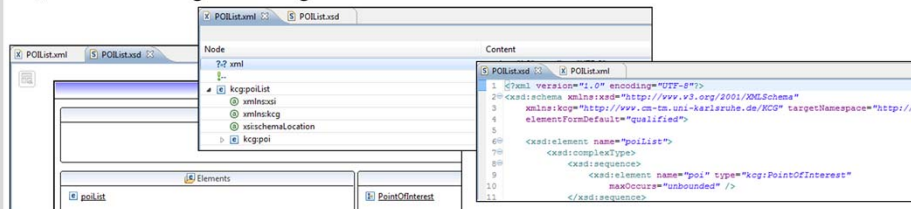
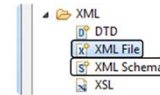
POI.java

(1) Rechtsklick auf den Code und Auswahl von "Source" -> "Generate Getters and Setters ...", um die Getter- und Setter-Methoden in Eclipse automatisch zu generieren.

(2) Es ist wichtig, dass die POI-Klasse mit der POI-XSD übereinstimmt, da eine Abbildung in beiden Richtungen realisiert werden soll. Werden Elemente in der XSD geändert, so sollte sich das auch in der POI-Klasse widerspiegeln und umgekehrt.

PA XML-PROGRAMMIERUNG – Eclipse XML-Editoren/Tools

- (1) Neues XSD/XML-Dokument erstellen
 - (1) "New" -> "File" und Endung ".xml" bzw. ".xsd" hinzufügen oder
 - (2) "New" -> "Other" -> "XML"
- (2) Visuelle- und Quell-Code-Bearbeitung
 - (1) Code-Hervorhebung und automatische Vervollständigung
- (3) Validierung wird angeboten



7 14.11.2013 WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

(1) Eclipse WTP (Web Tools Platform) bietet mit den XML-Editoren/Tools ein umfangreiches Set von Standard-Werkzeugen, um den Entwickler beim Bearbeiten von XML-Dokumenten zu unterstützen. Standardmäßig werden XML/XSD-Dateien mit den passenden Editoren geöffnet. Durch Rechtsklick und Auswahl von "Open With" kann man andere Editoren zum Öffnen der Dateien auswählen.

(2) Die Editoren unterstützen zwei Bearbeitungs-Paradigmen, die visuelle Bearbeitung und die Quell-Code-Bearbeitung. Für XML- und XSD-Dateien wird im Reiter "Design" das visuelle Editieren angeboten und das Dokument wird als Baumstruktur dargestellt. Der Entwickler kann durch "Drag-and-Drop" und Kontextmenüs die Dateien beliebig verändern. Das Editieren des Quelltextes wird im Reiter "Source" angeboten (bietet u.a. Code-Hervorhebung (engl. highlighting) und automatische Vervollständigung (engl. auto-complete)).

(3) Die Validierung kann vom Menü, das sich bei einem Rechtsklicks auf eine XML-, XSD- oder XSL-Datei öffnet, ausgeführt werden. Falls Fehler auftreten, werden diese in der Problem-Ansicht gezeigt. Ohne Schema-Angabe wird nur syntaktisch geprüft, ob ein XML-Dokument wohl-geformt (engl. well-formed) ist. Wenn eine Schemaangabe gemacht wird (in XML durch das Attribut "schemaLocation" und in XSD durch ein "import"-Tag), kann der interne Validierer von Eclipse prüfen, ob das XML-Dokument auch Schema-konform ist oder ob das XSD-Dokument die richtigen Elemente aus anderen Schemata importiert. Dies ist ein sehr nützliches Feature und erspart eine aufwändige und fehleranfällige manuelle Überprüfung.

PA XML-PROGRAMMIERUNG – Das Schema für die POI-Liste

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3.   xmlns:KIT-SC="http://www.cm-tm.uni-karlsruhe.de/KIT-SC"
4.   targetNamespace="http://www.cm-tm.uni-karlsruhe.de/KIT-SC"
5.   elementFormDefault="qualified">
6.
7.   <xsd:element name="poiList">
8.     <xsd:complexType>
9.       <xsd:sequence>
10.        <xsd:element name="poi" type="KIT-SC:PointOfInterest"
11.          maxOccurs="unbounded" />
12.      </xsd:sequence>
13.    </xsd:complexType>
14.  </xsd:element>
15.
16.  <xsd:complexType name="PointOfInterest">
17.    <xsd:sequence>
18.      <xsd:element name="name" type="xsd:string" />
19.      <xsd:element name="latitude" type="xsd:float" />
20.      <xsd:element name="longitude" type="xsd:float" />
21.    </xsd:sequence>
22.    <xsd:attribute name="id" type="xsd:integer" />
23.  </xsd:complexType>
24. </xsd:schema>
```

POIList.xsd

8

14.11.2013 WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

(1. <?xml ...) Das ist der Standard XML-Deklarations-Tag.

(2. <xsd: ...) Durch das Attribut "xmlns" kann ein Namensraum deklariert werden, der jeweils einem Präfix (hier: "xsd" und "KIT-SC") zugewiesen wird.

Hinweis: Ein Namensraum, der durch "xmlns=..." definiert ist, bezeichnet den "Default Namespace", in dem alle Elemente enthalten, die ohne ein Präfix angegeben sind.

Der Namensraum mit dem Präfix "xsd" für die "XML Schema Description" (XSD) wird deklariert, damit die Elemente zur Beschreibung von XSDs benutzt werden können. Das eigentliche Schema muss mit einem Import angegeben werden, damit auch der Validierer weiß, wo es liegt und Daten daraus beziehen kann. Das Standard-Schema "xsd" ist dem Validierer bekannt, da es von jeder XSD-Datei referenziert wird.

(3.) Der "KIT-SC"-Namensraum ist für die eigenen Elemente ("poiList" und "poi") und Typen ("PointOfInterest") vorgesehen.

(4.) Das "targetNamespace"-Attribut gibt an, zu welchem der Namensräume die neu definierten Elemente (z.B. Liste der POIs und das POI) gehören.

(5.) Wenn der Namensraum qualifiziert ist, dann müssen alle Elemente im XML-Dokument immer qualifiziert werden. Wenn der Wert des Attributes auf "unqualified" gesetzt wird, muss nur das Wurzelement qualifiziert werden.

(6.) Deklaration des Elementes, das die POI-Liste darstellt. Dieses Element ist vom Typen "complexType" und beinhaltet eine Sequenz von "poi"-Elementen.

(9.) Das Element POI soll den Inhalt der POIListe darstellen. Mit "KIT-SC:" wird ausgedrückt, dass dieses Element im "KIT-SC"-Namensraum enthalten ist.

Durch das Attribut "maxOccurs" kann die maximale Anzahl von Elementen, die in der Sequenz auftreten dürfen, angegeben werden. Der Wert "unbounded" (unbegrenzt) besagt, dass es keine obere Grenze gibt.

(14.-18.) POI-Element als "complexType" mit "simpleType"-Elementen "name" als "xsd:string", "latitude" und "longitude" als "xsd:float" (Gleitkommazahl). Da der "KIT-SC"-Namensraum auch "targetnamespace" ist, wird bei der Deklaration in (13.) angenommen, dass das POI-Element dem "KIT-SC"-Namensraum angehört.

(19.) Die Definition für Attribute erfolgt am Ende der Element-Deklaration vor. Das Attribut "id" wird als "xsd:integer" deklariert und soll zur eindeutigen Identifikation des POI dienen.


```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <KIT-SC:poiList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns:KIT-SC="http://www.cm-tm.uni-karlsruhe.de/KIT-SC"
4. xsi:schemaLocation="http://www.cm-tm.uni-karlsruhe.de/KIT-SC POIList.xsd">
5.   <KIT-SC:poi id="2323">
6.     <KIT-SC:name>Mensa</KIT-SC:name>
7.     <KIT-SC:latitude>49.011924</KIT-SC:latitude>
8.     <KIT-SC:longitude>8.41678</KIT-SC:longitude>
9.   </KIT-SC:poi>
10.  <KIT-SC:poi id="12314">
11.    ...
12.  </KIT-SC:poi>
13.  ...
14. </KIT-SC:poiList>
```

POIList.xml

Schon bekannte XML-Elemente werden hier nicht weiter erläutert.

(2.) Deklaration des Wurzelements mit Namensraum-Angabe, weil in der XSD definiert wurde dass alle Elemente von POIList.xsd-Instanzen qualifiziert sein müssen.

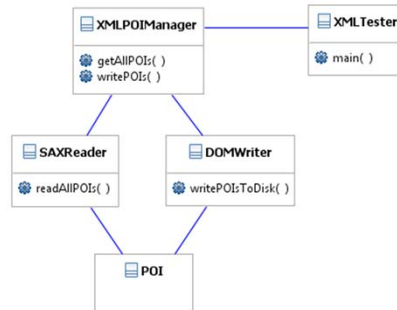
(3.) Die Namensraumangabe für den Namensraum des KIT-Smart-Campus. Dies ist nur ein Indikator, die eigentliche Referenz auf die XSD-Datei wird in (4.) angegeben.

(4.) Mit dem "schemaLocation"-Attribut kann man den Ort der XSD-Datei angeben, damit der Validierer prüfen können, ob die XML-Datei gültig (engl. valid) ist. In dem String, der als Wert dem Attribut zugewiesen wird, ist der Identifikator für den Namensraum an erster Stelle angegeben und durch Leerzeichen getrennt die eigentliche URL, wo die XSD-Datei zu finden ist.

(5.-12.) Die POIs werden nacheinander im Dokument aufgeführt. Hier ist die Auto-Complete-Funktion des Eclipse-XML-Editors hilfreich.

(14.) Nach der Beschreibung des letzten POIs wird die POI-Liste abgeschlossen.

- (1) Die im zu entwickelnden Java-Programm auftretenden Klassen und deren Assoziationen lassen sich als ein Klassendiagramm modellieren, um einen besseren Überblick zu erhalten
- (2) Erstellung mit den Eclipse UML2 Tools



(XMLTester) Diese Klasse enthält die "main"-Methode und ist der Startpunkt der Anwendung. Sie wird benutzt, um die Funktionalität zu testen.

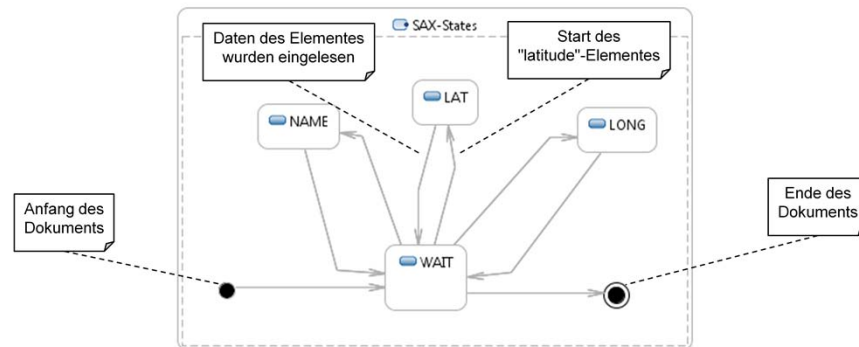
(XMLPOIManager) Diese Klasse bietet Methoden "getAllPOIs()" und "writePOIs()" an, um einem nachfragenden Klienten (Aufrufer der Methoden), wie z.B. dem "XMLTester", die Informationen zu liefern.

(SAXReader) Eine JAXP-SAX-Implementierung, um Daten aus einer XML-Datei auszulesen und sie als POI-Objekte in einer Liste auszugeben.

(DOMWriter) Eine JAXP-DOM-Implementierung, um aus einer Liste mit POI-Objekten ein XML-Dokument zu erstellen und es dann auf die Festplatte zu schreiben.

PA XML-PROGRAMMIERUNG – Funktion des "SAXReader"-Automaten

- (1) Innerhalb der Zustände "NAME", "LAT" und "LONG" wird der Inhalt des Elements ausgelesen und danach wird wieder zu "WAIT" gewechselt
- (2) Beim Finden eines "poi"-Elements wird kein Zustand geladen, sondern es wird lediglich ein neues POI-Objekt erstellt



11

14.11.2013

WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

(1) Dem Parsen von XML mit SAX liegt das Konzept der endlichen Automaten zugrunde. Da SAX eine Ereignis-basiertes API ist, kann man durch das Definieren von Zuständen die Informationen richtig zuordnen. Es werden Ereignisse (engl. events) gefeuert, wenn Elemente anfangen und enden. Separat werden Ereignisse gefeuert, die das Vorhandensein von Text angeben.

(WAIT) Im WAIT-Zustand wird der Text einfach übersprungen. Der Text-Event feuert z.B. auch bei XML-Kommentaren, nicht nur wenn Text im Inhalt des Elementes gefunden wird.

(NAME, LAT, LONG) In diesen Zuständen wird der Inhalt des jeweiligen Elementes ausgelesen und in das POI-Objekt gespeichert.

(2) Für das Starten des Elements "poi" und das Einlesen des Attributs "id" braucht man keinen Zustand, weil im selben Ereignis alle Informationen bereitgestellt werden.

```
1. public class SAXReader {
2.     ...
3.     private List<POI> poiList;
4.     private POI myPoi;
5.     private State state;
6.
7.     public void read(String url) {
8.         try {
9.             // init parser
10.            factory = SAXParserFactory.newInstance();
11.            parser = factory.newSAXParser();
12.            parser.parse(url, new MyHandler());
13.        } catch (Exception e) {
14.            ...
15.        }
16.    }
17.
18.    private class MyHandler extends DefaultHandler {
19.        @Override
20.        public void startDocument() throws SAXException {
21.            state = State.WAIT;
22.            poiList = new LinkedList<POI>();
23.            super.startDocument();
24.        }
25.    }
26.}
```

SAXReader.java

12

14.11.2013

WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

Die Klasse "SAXReader" übernimmt die Aufgabe des Auslesens aus der XML-Datei.

(3. - 5.) Deklaration der Variablen, die zum KIT-SC-Szenario gehören.

(6.) Die "read"-Methode erhält als Parameter den Pfad der zu lesenden Datei.

(9. - 11.) Die Initialisierung des Parsers geschieht durch das Aufrufen des dafür verantwortlichen Fabrik-Objekts (engl. factory object).

(15.) Die interne Klasse "MyHandler" überschreibt die Methoden des "DefaultHandler". Diese werden durch den Callback-Mechanismus vom Parser gerufen.

(16.) Die Annotation, durch die dem Übersetzer mitgeteilt wird, dass diese Methode überschrieben wird.

(17.) Die Methode, die beim Starten des Dokuments aufgerufen wird.

(18.) Der Zustand wird auf "WAIT" initialisiert.

(19.) Die Liste, die die POIs enthalten soll, wird initialisiert.

(20.) Dies ist ein Aufruf auf den Konstruktor des "DefaultHandler". Dieser Aufruf erfolgt standardmäßig.

```
1.      public void startElement(String uri, String localName, String qName,  
2.      Attributes attributes) throws SAXException {  
3.  
4.          if (qName.equals("KIT-SC:poi")) {  
5.              myPoi = new POI();  
6.              myPoi.setId(Integer.parseInt(attributes.getValue("id")));  
7.          }  
8.          if (qName.equals("KIT-SC:name")) {  
9.              state = State.NAME;  
10.         }  
11.         ...  
12.     }  
13.     super.startElement(uri, localName, qName, attributes);  
14. } //startElement  
  
15.     public void characters(char[] ch, int start, int length)  
16.     throws SAXException {  
17.         switch (state) {  
18.             case NAME:  
19.                 myPoi.setName(new String(ch, start, length));  
20.                 break;  
21.             case LAT:  
22.                 ...  
23.             }  
24.             state = State.WAIT;  
25.             super.characters(ch, start, length);  
26.         }  
27.     }  
28. }
```

SAXReader.java

13 14.11.2013 WASA - INFORMATION

Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

(1.) Wenn ein Element gefunden wird, dann wird die "startElement"-Methode aufgerufen und es werden ihr als Parameter die Informationen über das Element übergeben.

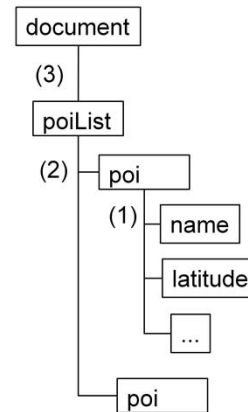
(4.-10.) Das String-Objekt "qName" (von qualified name) enthält den Namen des Elementes, der in den "if"-Klauseln benutzt wird, um zu entscheiden, welcher Zustand zu laden ist. Beim Auftreten des qualifizierten Namens "KIT-SC:poi" wird kein Zustand geladen, sondern es wird ein neues "POI"-Objekt erstellt und da die Attribute der Elemente auch als Parameter dieser Methode übergeben werden, kann der Wert des "id"-Feldes des "POI"-Objekts gesetzt werden.

(15.) Wird eine Reihe von Zeichenketten gefunden, wird die "characters"-Methode aufgerufen. Der erste Parameter enthält das gesamte Dokument als Charakter-Array "char[] ch". Nur durch die Variablen "start" und "length" kann der nützliche Teil extrahiert werden.

(17.) Je nach Zustand wird das geeignete Feld des "POI"-Objektes "myPoi" auf den Wert gesetzt, den der Parser übergibt.

(24.) Am Ende des "switch"-Blocks wird der Zustand auf "WAIT" gesetzt, um eine fehlerhafte Veränderung der schon gesetzten Felder zu vermeiden, wenn ein "character"-Ereignis gefeuert wird.

- (1) Die Blätter (z.B. "name" oder "latitude") werden zuerst an die "poi"-Knoten gehängt
- (2) Die "poi"-Knoten werden dann an den Wurzelknoten "poiList" angehängt
- (3) "poiList" wird dann an das Dokument angehängt
- (4) Zum Speichern des im Speicher liegenden DOM-Dokuments in eine Datei als XML-Dokument ist die Transformation in ein passendes Format notwendig
 - (1) Dies übernimmt die von JAXP angebotene XSL-API



(1) Im vorliegenden Fall sind die Blätter die Attribute, die in den Instanzen der Klasse "POI" gespeichert sind. Aus diesen Attributen werden DOM-Elemente, d.h. jedes Attribut wird auf ein DOM-Element abgebildet. Der "poi"-Knoten ist auch ein DOM-Element, dessen Attribut "id" gesetzt werden muss.

(2) Das erstellte DOM-Dokument soll eine Darstellung der internen POI-Liste in XML sein. Die einzelnen POIs werden der Reihe nach in DOM-Elemente abgebildet und angehängt.

(3) Es darf in einem XML-Dokument nicht mehr als einen Wurzelknoten geben. Das Element "poiList" ist komplett, es muss aber noch dem erstellten Dokument hinzugefügt werden, da es im Speicher als unabhängige Datenstruktur abgelegt ist und erst im Zusammenhang mit dem Dokument die fertige Repräsentation der XML-POI-Liste darstellt.

(4) Das Dokument kann nicht direkt ausgegeben werden, da es nicht als XML, sondern als baumartige Datenstruktur im Speicher vorliegt.

(4.1) Mit wenigen an die von JAXP-XSL-API gerichteten Methodenaufrufe wird das Dokument serialisiert und in die Datei geschrieben.

```
1. public int write(List<POI> poiList) {  
2.     try {  
3.         factory = DocumentBuilderFactory.newInstance();  
4.         builder = factory.newDocumentBuilder();  
5.         doc = builder.newDocument();  
6.         doc.createElement("poiList");  
  
7.         Element poiList = doc.createElementNS(  
8.             "http://www.cm-tm.uni-karlsruhe.de/KIT-SC", "KIT-SC:poiList");  
9.         poiList.setAttribute("xmlns:xsi",  
10.             "http://www.w3.org/2001/XMLSchema-instance");  
  
11.         for (POI p : poiList) {  
12.             Element poi = doc.createElement("KIT-SC:poi");  
13.             poi.setAttribute("id", String.valueOf(p.getId()));  
14.             Element name = doc.createElement("KIT-SC:name");  
15.             name.setTextContent(p.getName());  
16.             ...  
17.             poiList.appendChild(poi);  
18.         }  
19.         doc.appendChild(poiList);  
20.         ...  
21.         outputToFile(path);  
22.     } //write
```

DOMWriter.java

(3. - 6.) Ein "DocumentBuilderFactory"-Objekt kann "DocumentBuilder" erstellen, die dann neue DOM-Dokumente liefern. Bei SAX hat dies genauso funktioniert, da beide durch die JAXP-API den darunterliegenden Parser aufrufen.

(7.) Das Wurzelement (Wurzelknoten) "poiList" wird erstellt.

(8.) Zum Wurzelement müssen die Namensräume hinzugefügt werden, was mittels "doc.createElementNS" erfolgt.

(9.) Der Namensraum der "XMLSchema-instance" lässt sich als normales Attribut angeben.

(11.) Es wird durch alle Elemente in der Liste iteriert und die entsprechenden DOM-Knoten werden erstellt.

(12.) Ein neues DOM-Element wird erstellt, das im XML-Dokument einem "poi"-Tag entspricht. DOM schließt die Elemente automatisch, so dass dem Entwickler die Arbeit abgenommen wird und die Fehlerquellen reduziert werden.

(13. - 15.) Die Daten aus dem derzeitigen "POI"-Objekt einlesen. Die "id" wird als Attribut geschrieben, der Inhalt des Namens wird als Textinhalt (engl. text content) geschrieben.

(17.) Jedes aus einem POI-Objekt erhaltene Element (Knoten) wird dem Element "poiList" angehängt.

(19.) Das Element "poiList" wird zuletzt an das DOM-Dokument "doc" angehängt, womit dieses Element der Wurzelknoten wird.

(21.) Die Helfer-Methode "outputToFile" übernimmt das Transformieren und das Schreiben der Datei.

```
1. private void outputToFile(String path) {  
2.     try {  
3.         TransformerFactory transformerFactory = TransformerFactory  
4.             .newInstance();  
5.  
6.         Transformer transformer = transformerFactory.newTransformer();  
7.  
8.         transformer.setOutputProperty(OutputKeys.INDENT, "yes");  
9.  
10.        Result result = new StreamResult(new File(path));  
11.        DOMSource source = new DOMSource(doc);  
12.        transformer.transform(source, result);  
13.    }  
14. }
```

DOMWriter.java

(3. - 5.) Es wird ein neues Transformer-Objekt "transformer" erstellt. Dies wird auch durch die JAXP-API gewährleistet.

(6.) Um ein ansprechend formatiertes Dokument am Ende zu erhalten, kann diese Option eingeschaltet werden, indem man die Eigenschaft "OutputKeys.INDENT" mit dem Wert "yes" dem "transformer" übergibt.

(7. - 9.) Hier wird die eigentliche Transformation durchgeführt. Das "result"-Objekt ist ein Stream, mit dem man in die Datei schreibt. Das "source"-Objekt ist eine DOM-Quelle, die mit den Daten aus dem gerade erstellten DOM-Baum gefüllt wird.


```
1. package test;
2. import java.util.List;
3. import model.POI;
4. import poiManagement.XMLPOIManager;
5. public class XMLTester {
6.     private static String file = "./Data/POIList.xml";
7.     public static void main(String[] args) {
8.         XMLPOIManager manager = new XMLPOIManager();
9.         List<POI> poiList = manager.getAllPOIs(file);
10.        for (POI p : poiList) {
11.            p.setName(p.getName().toUpperCase());
12.        }
13.        manager.writeToDisk(poiList);
14.    }
15. }
```

XMLTester.java

Die "XMLTester"-Klasse ist der Einstiegspunkt der Anwendung und beinhaltet Code zum Testen der XML-Bearbeitungsfunktionalitäten des Managers.

(6.) Das String-Objekt "file" speichert den Pfad, der auf die XML-Datei zeigt.

(8.) Ein neuer "XMLPOIManager" wird erstellt, der die benötigten XML-Bearbeitungsfunktionalitäten bereitstellt.

(9.) Die "getAllPOIs"-Methode wird aufgerufen und als Parameter wird das "file"-Objekt übergeben. Der Rückgabewert wird in "poiList", einer Liste von POIs ("List<POI>") gespeichert.

(10.) Um auf alle POIs zuzugreifen, wird mit einer "for"-Schleife durch die "List" iteriert.

(11.) Um eine einfache Operation auf den POIs zu veranschaulichen, wird jeder Name eines POIs in Großbuchstaben geschrieben.

(13.) Die "writeToDisk"-Methode wird aufgerufen und als Parameter wird die Liste mit den veränderten POIs übergeben.

PA XML-PROGRAMMIERUNG – Ausführung des XMLPOIManagers

The screenshot displays an IDE with the following components:

- Package Explorer (Left):** Shows the project structure with packages: `data`, `model`, `poiManagement`, and `test`. The `test` package contains `XMLTester.java`.
- Source Editor (Top Right):** Shows the `POI.java` file with imports for `java.util.List`, `model.POI`, and `poiManagement.XMLPOIManager`.
- Console (Bottom Right):** Displays the execution output of `XMLTester`. The output is divided into two sections:
 - Parsen des ersten POI-Elements:** Shows the parsing of the first POI element, including state transitions (WAIT, LONG) and processing of name, latitude, and longitude.
 - Parsen des zweiten POI-Elements:** Shows the parsing of the second POI element, including state transitions (WAIT, LONG) and processing of name, latitude, and longitude.

At the bottom of the slide, the following information is displayed:

18 14.11.2013 WASA - INFORMATION Cooperation & Management (C&M, Prof. Abeck)
Institut für Telematik, Fakultät für Informatik

Die Ausführung erfolgt durch einen Rechtsklick auf die Klasse "XMLTester.java" und Auswahl von "Run As"-> "Java Application".

(Projektstruktur) Zeigt die angelegten vier zum Projekt angelegten Pakete und die darin enthaltenen Klassen (Pakete "model", "poiManagement" und "test") bzw. XML-/XSD-Dokumente (Paket "data").

(Programmablauf) Die Ausgaben auf der Konsole resultieren aus Testausgaben, die in den Java-Klassen vorgesehen wurden, um den Ablauf des Programms verfolgen zu können.

(Parsen ...) Der größte Teil der Testausgaben stammt aus der Klasse "SAXReader" und dem darin realisierten endlichen Automaten, durch den das XML-Dokument in die einzelnen XML-Elemente zerlegt (engl. parse) wird und die Werte Java-Objekte der Klasse "POI" zugewiesen werden.