XML-basierter Datenaustausch in Java (2)

PROG 2: Einführung in die Programmierung für Wirtschaftsinformatiker

---

Steffen Helke
Technische Universität Berlin
Fachgebiet Softwaretechnik

13. Mai 2013

Übersicht

- Wiederholung: XML & DTD
- XML-Verarbeitung mit SAX
- XML-Verarbeitung mit DOM
- XML-Transformer

Teil III der Vorlesung PROG 2

XML und Java

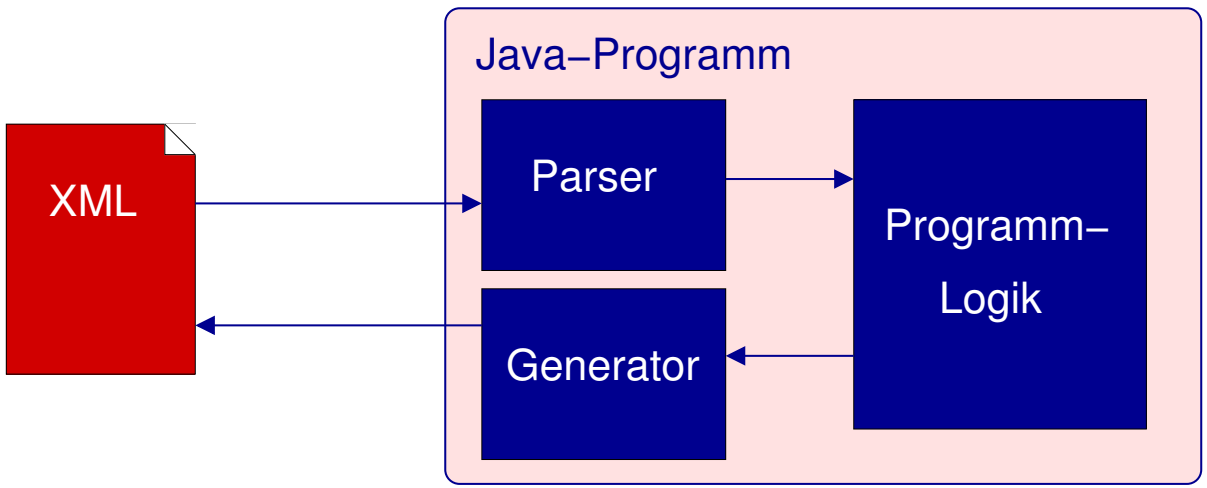
Grundlegende Einführung: Extensible Markup Language

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

XML: Extensible Markup Language

Einordnung

- Basis für Dokumentenaustausch
- hierarchisch strukturierte Daten in Textform



Beispiele

- Beschreibung von GUI-Layouts
- Speicherformat für Textdokumente, wie .docx

Aufbau einer XML-Kodierung

Grundelemente

1. Prolog: Version und Kodierungsart
2. Elemente in Tags

```
<Tag> Elemente </Tag>
```

Beispiel: *Buch.xml*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Buch>
 <Titel>Wilhelm Tell</Titel>
 <Autor>
  <Vorname>Friedrich</Vorname>
  <Nachname>Schiller</Nachname>
  <!-- Nachfolgend erscheint ein Bild von Schiller -->
  <Img quelle="Schiller.jpg" hoehe="200" breite="100" />
 </Autor>
</Buch>
```

Trennung in Struktur und Inhalt

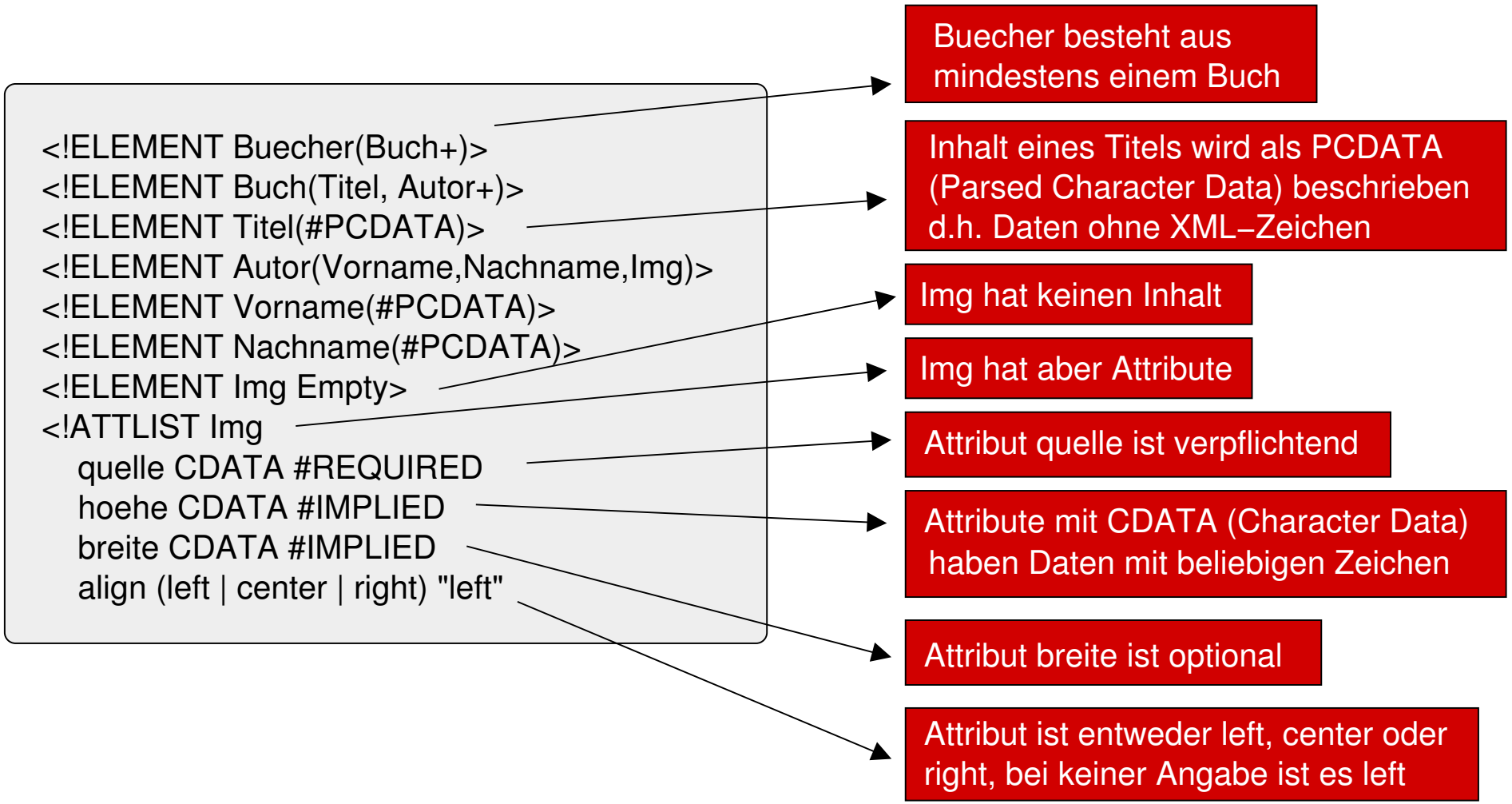
Buecher.dtd

```
<!ELEMENT Buecher(Buch+)>
<!ELEMENT Buch(Titel,
                Autor+)>
<!ELEMENT Titel(#PCDATA)>
<!ELEMENT Autor(Vorname,
      Nachname,Img)>
<!ELEMENT Vorname(#PCDATA)>
<!ELEMENT Nachname(#PCDATA)>
<!ELEMENT Img EMPTY>
<!ATTLIST Img
 quelle CDATA #REQUIRED
 hoehe CDATA #IMPLIED
 breite CDATA #IMPLIED
 align (left|center|right)
 "left">
```

Buecher.xml

```
 <?xml version="1.0"
       encoding="ISO-8859-1"
       standalone="no" ?>
 <!DOCTYPE Buecher >
<Buecher>
 <Buch>
  <Titel>Wilhelm Tell</Titel>
  <Autor>
   <Vorname>Friedrich</Vorname>
   <Nachname>Schiller</Nachname>
   <!-- Nachfolgend erscheint
        Bild von Schiller -->
   <Img quelle="Schiller.jpg"
        hoehe="200"
        breite="100"
        align="center"/>
  </Autor>
 </Buch>
</Buecher>
```

Syntax für Dokumenttyp-Definition (DTD)



DTD: Dokumenttyp-Definition

Bestandteile der DTD-Datei

- Bücher = mehrere Elemente vom Typ Buch
- Buch besteht aus Titel, Autor und Bild des Autors
- Autor besteht aus Vor- und Nachname
- Bild besteht aus Quelle und ggf. Dimensionen/Ausrichtung

Grenzen der DTD bei komplexen Datenmodellen

- Datentypprüfung, Namensräume und Kardinalität
- Beschreibungen (Annotationen)
- Objektorientierung

$\Rightarrow$ Lösung: Verwendung von XML-Schemata

Bestandteile von XML-Schemata

1. Version, Zeichenkodierung und Namensraum

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

2. einzelne Elemente

```
<xs:element name = "Elementname">
<xs:attribute name = "Attributname" type="xs:string"
                                    use="required>
```

Syntax

- Einführung Namensraum xs mit <xs:schema xmlns:xs=URI>
- einfache/komplexe Typen mit <simpleType>/<complexType>
- 44 vordefinierte Datentypen, wie string , **int** , date , anyURI ,...
- eigene Datentypen mit Namen und zulässigen Werten, eingerahmt mit < restriction >

Syntax für XML-Schemata

Notationen

- <sequence>: Festlegung der Reihenfolge für Elemente
- <all>: alle Elemente notwendig, Reihenfolge beliebig
- <choice>: nur ein Element darf angegeben werden
- minOccurs: Mindestanzahl von Elementen
- maxOccurs: Maximalanzahl von Elementen
- unbounded: beliebige Anzahl von Elementen
- required: Attribut notwendig
- optional: Attribut freiwillig
- prohibited: Attribut nicht erlaubt

Beispiel

```
<xs:element name="abc" minOccurs="1" maxOccurs="unbounded"/>
```

Beispiel: XML Schema Buecher.xsd (1)

1. Definition Kopplung zum Schema und Namensraum

```
<?xml version="1.0"?>
<xschema:schema
 xmlns:xschema="http://www.w3.org/2001/XMLSchema">
```

⇒ Namensraum ist in diesem Beispiel xschema

⇒ Kodierung nach Standard http://www.w3.org/2001/XMLSchema

2. Definition Datentyp align mit drei möglichen Werten

```
<xschema:simpleType name="align">
 <xschema:restriction base="xschema:string">
  <xschema:enumeration value="left" />
  <xschema:enumeration value="center" />
  <xschema:enumeration value="right" />
 </xschema:restriction>
</xschema:simpleType>
```

Beispiel: XML Schema Buecher.xsd (2)

3. Definition Attribute für die Bilddarstellung

```
<xschema:attributeGroup name="img_attribute">
 <xschema:attribute name="quelle"
                    type="xschema:string"
                    use="required" />
 <xschema:attribute name="hoehe"
                    type="xschema:string"
                    use="optional" />
 <xschema:attribute name="breite"
                    type="xschema:string"
                    use="optional" />
 <xschema:attribute name="ausrichtung"
                    type="align"
                    use="optional" />
</xschema:attributeGroup>
```

⇒ Typdefinition von align wurde hier wiederverwendet!

Beispiel: XML Schema Buecher.xsd (3)

4. Definition Datentyp Person mit Vorname, Nachname, Bild

```
<xschema:complexType name="Person">
 <xschema:all>
  <xschema:element name="Vorname"
                   type="xschema:string" />
  <xschema:element name="Nachname"
                   type="xschema:string" />
  <xschema:element name="Img">
   <xschema:complexType>
     <xschema:attributeGroup ref="img_attribute" />
   </xschema:complexType>
  </xschema:element>
 </xschema:all>
</xschema:complexType>
```

⇒ Attributdefinition von img_attribute wurde hier wiederverwendet!

Beispiel: XML Schema Buecher.xsd (4)

5. Definition Datentyp Buecher: mindestens ein Buch usw.

```
<xschema:element name="Buecher">
 <xschema:complexType>
  <xschema:sequence>
   <xschema:element name="Buch" minOccurs="1"
                    maxOccurs="unbounded">
    <xschema:complexType>
     <xschema:sequence>
      <xschema:element name="Titel" type="xschema:string"/>
      <xschema:element name ="Autor" type="Person"
                       minOccurs="1" maxOccurs="unbounded"/>
     </xschema:sequence>
     <xschema:attribute name="seiten" type="xschema:int"
                        use="optional" />
    </xschema:complexType>
   </xschema:element>
  </xschema:sequence>
 </xschema:complexType>
</xschema:element>
```

⇒ Typdefinition von Person wurde hier wiederverwendet!

Beispiel: XML Schema Buecher.xml (5)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Buecher xmlns:xsi=
            "http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="Buecher.xsd" >
 <Buch seiten="240">
  <Titel>Wilhelm Tell</Titel>
  <Autor>
   <Vorname>Friedrich</Vorname>
   <Nachname>Schiller</Nachname>
   <Img quelle="Schiller.jpg" hoehe="200"
        breite="100" ausrichtung="center" />
  </Autor>
 </Buch>
 <Buch>
  <Titel>Faust 1.Teil</Titel>
  <Autor>
   <Vorname>Johann Wolfgang</Vorname>
   <Nachname>von Goethe</Nachname>
   <Img quelle="Goethe.jpg" ausrichtung="right" />
  </Autor>
 </Buch>
</Buecher>
```

Teil III der Vorlesung PROG 2

XML und Java

Java API for XML Processing (SAX)

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

JAXP: Java API for XML Processing

Intention

- Unterstützung der Verarbeitung von XML-Dokumenten durch Parsen, Validieren und Transformieren

Zugehörige Packages

1. *SAX* (Simple API for XML): API zum Parsen von XML-Dokumenten
2. *DOM* (Document Object Model): Parsen von XML-Dokumenten, Zugriff/Änderung auf/von XML-Daten
3. *TrAX* (Transformation API for XML): Transformation von XML-Dokumenten in andere Formate, z.B. HTML oder Text

Unterschiede zwischen SAX und DOM

SAX

- Simple API for XML
- schrittweise Interpretation des Dokuments
- Daten stehen nach dem Parsen nicht mehr zur Verfügung
- ist schnell und einfach, ideal für einmaliges Parsen

DOM

- Document Object Model
- Erzeugen einer Baumstruktur im Speicher
- dynamischer Datenzugriff nach dem Parsen möglich
- langsamer als SAX, aber besser bei mehrfachem Zugriff

SAX: Simple API for XML

Funktionsweise

- Ereignisgesteuerte, sequenzielle Abarbeitung von XML-Dateien
- Erzeugen von Ereignissen für alle gelesenen Elemente
- Aktivieren zugeordneter Ereignismethoden

Ereignisse beim Lesen einer XML-Datei

- Beginn/Ende des Dokuments
- Anfang/Ende des Elements
- Inhalt eines Elements

Einbettung von SAX in Java

Vorgehen

1. Erzeugen eines Objekts vom Typ *org.xml.sax.XMLReader*
   z.B. durch *XMLReaderFactory*
2. Übergeben eines Parsers als Parameter
   kein Parameter $\Rightarrow$ Verwendung eines Default-Parsers
3. Parsen einer XML-Datei durch Aufruf der Methode *parse*
4. Exception-Behandlung für *SAXExceptions*

Hinweis

- Häufig erfolgt eine SAX-Einbettung auch mit Hilfe von *javax.xml.parsers.SAXParserFactory*, um die Abstraktionsschicht von JAXP auszunutzen

Beispiel: Einbettung von SAX in Java

```
1  import org.xml.sax.XMLReader;
2  import org.xml.sax.helpers.XMLReaderFactory;
3
4  public class EinbettungSax {
5   public EinbettungSax() {
6    try {
7      // SAX Parser erzeugen
8      XMLReader xr = XMLReaderFactory.createXMLReader();
9      xr.parse("Buch.xml");
10     System.out.println("Datei gefunden und bearbeitet");
11   }
12   catch ( Exception e ) {
13     System.out.println("Datei nicht gefunden: "
14                        + e.getMessage());
15     e.printStackTrace();}
16  }
17  public static void main( String[] argv ) {
18    System.out.println( "Einbettung von SAX in Java" );
19    new EinbettungSax();}
20 }
```

Alternative Einbettung von SAX in Java

```
1  import java.io.File;
2  import javax.xml.parsers.SAXParserFactory;
3  import javax.xml.parsers.SAXParser;
4
5  public class SAXExample {
6   public static void main( String[] argv ){
7    try {
8     // SAX Parser erzeugen
9     SAXParserFactory factory = SAXParserFactory.newInstance();
10    // Namesraeume behandeln
11    factory.setNamespaceAware(true);
12    SAXParser xr = factory.newSAXParser();
13    // Eingabedatei parsen
14    xr.parse( new File ("Students.xml"), ... );
15   }
16   catch ( Exception e ) {
17     e.printStackTrace();}
18   }
19 }
```

Handler zur Ereignisbehandlung in SAX

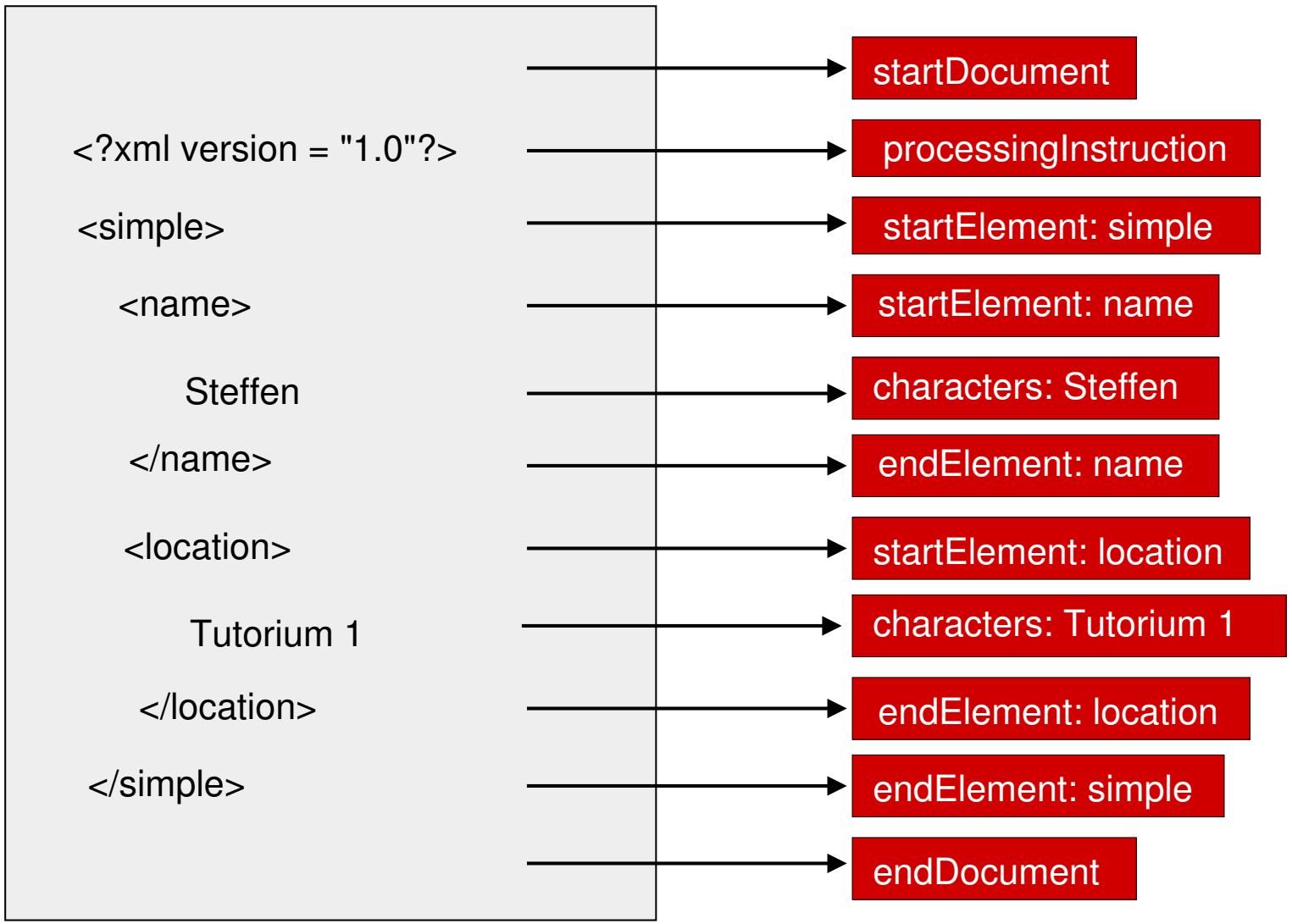
Idee

- SAX-Handler definieren Callback-Methoden zur Bearbeitung von Ereignissen (analog zum Listener-Konzept)

Arten

1. ContentHandler: Ereignisse zum logischen Inhalt
2. DTDHandler: Ereignisse zu DTD
3. EntityResolver: Aktivierung bei Aufruf externer Ressourcen
4. ErrorHandler: Reaktion auf Fehler beim Parsen

Ereignisse eines ContentHandler in SAX



ContentHandler-Methoden

- Beginn/Ende eines Dokumentes

```
void startDocument() throws SAXException
```

- Beginn/Ende eines Elementes

```
void startElement(String namespaceURI,
                  String localName,
                  String qName,
                  Attributes atts)
```

- Verarbeitung des Inhalts eines Elementes

```
void characters(char[] ch,int start,int length)
```

- Überspringen nicht druckbarer Zeichen

```
void ignorableWhitespace(char[] ch,int start,int length)
```

- ... und noch weitere ...

Beispiel Implementierung ContentHandler

```
1  public class MySAXContentHandler extends DefaultHandler {
2
3   public List<Student> students = new ArrayList<Student>();
4   private Student currentStudent;
5   private String currentContent = "";
6
7   public void endElement( String namespaceURI,
8                           String localName,
9                           String qName ) throws SAXException{
10    if ( localName.equals( "name" ) ) {
11          currentStudent = new Student(currentContent); }
12    if ( localName.equals( "location" ) ) {
13          currentStudent.location = currentContent;
14          students.add(currentStudent);
15          currentStudent = null;}}
16
17  public void characters( char[] ch, int start, int length )
18          throws SAXException {
19      currentContent = new String(ch, start, length);}
20 }
```

Aktivieren eines SAX-ContentHandlers

```
1  import java.io.File;
2  import javax.xml.parsers.SAXParserFactory;
3  import javax.xml.parsers.SAXParser;
4
5  public class SAXExample {
6   public static void main( String[] argv ){
7    try {
8     // SAX Parser erzeugen
9     SAXParserFactory factory = SAXParserFactory.newInstance();
10    SAXParser xr = factory.newSAXParser();
11    // Namesraeume behandeln
12    factory.setNamespaceAware(true);
13    // ContentHandler erzeugen und aktivieren
14    MySAXContentHandler ch = new MySAXContentHandler();
15    // Eingabedatei parsen
16    xr.parse( new File ("Students.xml"), ch);
17   }
18   catch ( Exception e ) {
19     e.printStackTrace();}
20   }
21 }
```

Teil III der Vorlesung PROG 2

XML und Java

Java API for XML Processing (DOM)

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

DOM: Document Object Model

Intention

- Plattform- und Sprachenunabhängige Schnittstelle für den dynamischen Zugriff und die Änderung von Struktur und Inhalt

Vergleich mit SAX

- verwendet auch die *parse*-Methode, aber anderer Effekt
- erzeugt keine Ereignisse während des Parsens
- erstellt Kopie der XML-Datei im Speicher als Baumstruktur
- repräsentiert Elemente als Knoten mit Referenzen auf über und untergeordnete Knoten
- Zugriffe mit üblicher Baumtraversierung

Beispiel: Einbettung von DOM in Java

```
1  import org.w3c.dom.Document;
2  import org.w3c.dom.NodeList;
3  import org.apache.xerces.parsers.DOMParser;
4
5  public class EinbettungDOM {
6    private Document doc;
7    public EinbettungDOM() {
8      DOMParser dom = new DOMParser();
9      try {
10       dom.parse("Students.xml");
11       doc = dom.getDocument();
12       System.out.println("Datei gefunden und bearbeitet");
13     }
14     catch ( Exception e ) { ... }
15   }
16
17   public static void main( String[] argv ) {
18       System.out.println( "Einbettung von DOM in Java" );
19       EinbettungDOM dom = new EinbettungDOM();
20       dom.showNode(); }
21
22   void showNode() { ... }          }
```

Knoten in der DOM-Baumdatenstruktur

Allgemeines

- alle Baumknoten sind Objekte von Klassen mit in *org.w3c.dom.Node** definierten Interfaces
- Interfaces für unterschiedliche Knotentypen

Objekte der Klasse *Document*

- Wurzel der Baumstruktur und Eintrittspunkt
- Hinweis auf verwendete DTD (ist Null falls keine benutzt)

Speicherung der Konten

- *NodeList*: nummeriertes Feld mit Knoten, Zugriff mit *item*
- *NamedNodeMap*: analoge Speicherung, zusätzlich Adressierung über Namen möglich

*Anmerkung: W3C steht für *World Wide Web Consortium*

Methoden: Zugriff/Modifikation von Knoten

```
// neuen Knoten am Ende der Liste hinzufuegen
Node appendChild(Node newChild)
// Liste aller Kindknoten
NodeList getChildNodes()
// Erster Kindknoten
Node getFirstChild()
// Elternknoten
Node getParentNode()
// Knotenname (abhaengig vom Knotentyp)
String getNodeName()
// Wert neu setzen
void setNodeValue(String nodeValue)
// Knotenbezeichnung
short getNodeType()
//  Wert des Knotens
string getNodeValue()
//  Attribute vorhanden?
boolean hasAttributes()
// Kinderknoten vorhanden?
boolean hasChildNodes()
// Knoten entfernen
Node removeChild(Node oldChild)
```

Beispiel: Auslesen von XML-Daten mit DOM

Struktur: Students.dtd

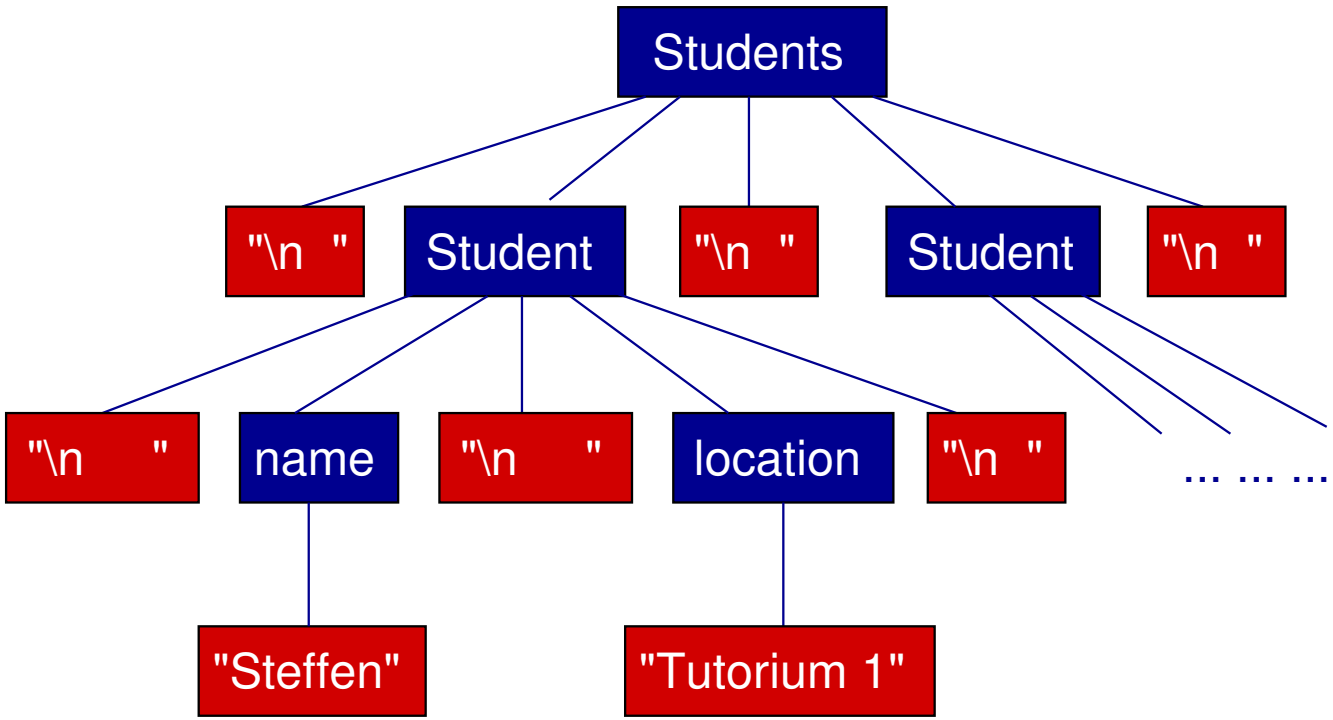
```
<!ELEMENT Students (Student+)>
  <!ELEMENT Student (name, location)>
    ...
```

Inhalt: Students.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Students SYSTEM "Students.dtd">
<Students>
  <Student>
    <name>Steffen</name>
    <location>Tutorium 1</location>
  </Student>
  <Student>
    <name>Saskia</name>
    <location>Tutorium 2</location>
  </Student>
</Students>
```

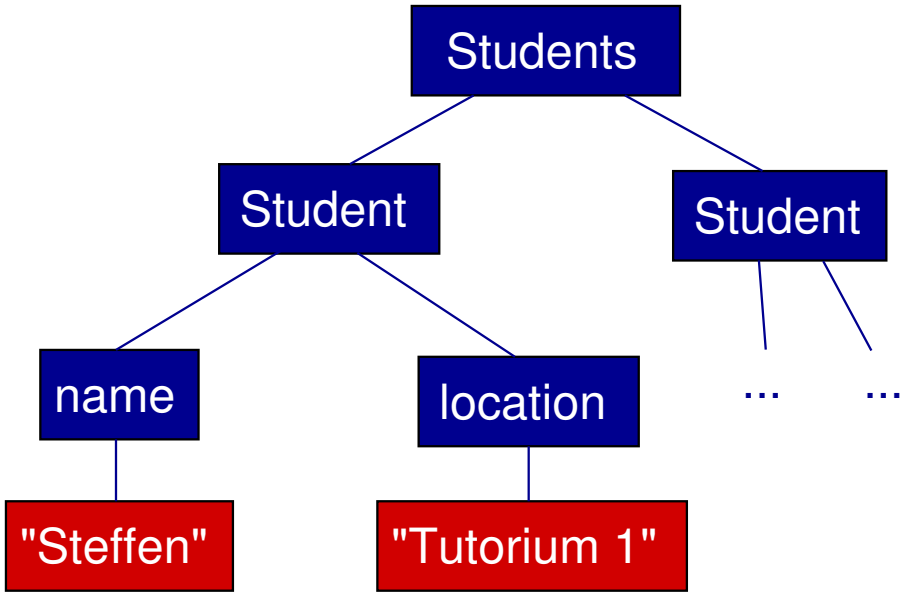
Wie viele direkte Unterknoten hat der Wurzelknoten Students? 5

DOM-Baumdatenstruktur für students.xml



- explizite Repräsentation von White-Spaces (nicht druckbarer Zeichen) als Knoten im DOM-Baum
- erschwert häufig die Bearbeitung der XML-Daten

Überspringen nicht druckbarer Zeichen



Unterbindung von White-Space-Knoten im DOM-Baum

```
DOMParser dom = new DOMParser();
String URL = "http://apache.org/xml/features/dom/";
String whitespace = "include-ignorable-whitespace";
dom.setFeature(URL + whitespace, false)
dom.parse("Students.xml");
doc = dom.getDocument();
```

Behandlung eines Knoten im DOM-Baum

1. Überprüfen, ob der Knoten Attribute hat

```
if( node.hasAttributes() ) { ... }
```

2. Falls ja, Attributliste zurückgeben lassen

```
NamedNodeMap attributes = node.getAttributes();
```

3. Ausgeben einzelner Attribute

```
for(int i=0; i<attributes.getLength(); i++) {
   System.out.print(attribute.item(i).getNodeName() +
   "=" + attribute.item(i).getNodeValue());
 }
```

4. Überprüfen, ob der Knoten Kinder hat

```
if( node.hasChildNodes(); ) { ... }
```

5. Falls ja, Liste mit Nachfolgeknoten zurückgeben lassen

```
NodeList childNodes = node.getChildNodes();
```

Behandlung aller Knoten im DOM-Baum

```
1  public static void main(String[] args) {
2     EinbettungDOM dom = new EinbettungDOM();
3     dom.anzeigen(doc.getDocumentElement(), 1);
4  }
5  public void anzeigen(Node node, int stufe) {
6     for(int i = 1; i < stufe; i++)
7        System.out.print("  ");
8
9     System.out.print(node.getNodeName() + " ");
10    if(node.getNodeValue() != null)
11      System.out.print("= "+ node.getNodeValue()+ " ");
12
13    if(node.hasAttributes()) { ... }
14
15    if(node.hasChildNodes()) {
16      stufe++;
17      NodeList childs = node.getChildNodes();
18      for(int i = 0; i < childs.getLength(); i++)
19        this.anzeigen(childs.item(i), stufe); }
20    else
21      stufe--;
22 }
```

Bearbeitung komplexer XML-Dokumente

- Idee: Ausführung spezialisierter Auswertungsmethoden in Abhängigkeit vom Knotentyp (mit *switch*-Konstrukt)

Knotentyp	Beschreibung	Kinder
Document	Gesamtes Dokument (Wurzel des DOM-Baumes)	max. ein Element, ProcessingInstruction, Comment, DocumentType
Entity, Entity Reference	Entität bzw. Verweis darauf	Element, ProcessingInstruction, Comment, Text, EntityReference
Element	Element	identisch zu EntityReference
Attr	Attribute	Text, EntityReference
Text	Textuelles Element oder Attribut	keine
CDATASection	Texte, die nicht vom Parser bearbeitet werden	keine
Comment	Kommentar	keine
Notation	Notation aus DTD	keine

Konstanten zur Abfrage von Knotentypen

Vorgehen

- Abfrage des Knotentyp mit *node.getNodeType()*
- Rückgabewerte sind definierte Konstanten
- Definition des auszuführenden Codes pro Knotentyp

```
case Node.TEXT_NODE:
  ... // Textknoten
  ... // auswerten
break;
```

Konstanten
ELEMENT_NODE
ATTRIBUTE_NODE
TEXT_NODE
CDATA_SECTION_NODE
ENTITY_REFERENCE_NODE
ENTITY_NODE
PROCESSING_INSTRUCTION_NODE
COMMENT_NODE
DOCUMENT_NODE
COMMENT_TYPE_NODE
DOCUMENT_FRAGMENT_NODE
NOTATION_NODE

Speicherung von Daten mit XML

Intention

- temporäre oder dauerhafte Speicherung von Daten, die in einer Anwendung erzeugt werden

Vorgehen Speicherung

1. Erstellen eines Objekts der Klasse *Document*
2. Erzeugen eines XML-Baums im Speicher
3. Eintragen von Attributen/Werten in den Knoten
4. Ausgabe in XML-Datei schreiben, z.B. mit *FileWriter*

Vorgehen Modifikation

1. Einlesen des aktuellen XML-Dokuments
2. Modifikation der Inhalte durch den Benutzer
3. Speicherung der neuen Version

1. Erstellen Objekt der Klasse Document

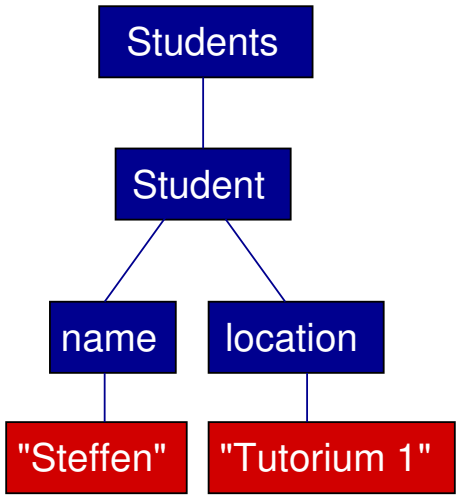
1. Erzeugen eines *Document*-Knotens als Wurzel des XML-Dokuments
2. optionale Erzeugung von Document-Type-Knoten für die Verwendung von DTD

```
DOMImplementationImpl di = new DOMImplementationImpl();
// z.B. Dokumentname="Students"
Document doc = di.createDocument(Namensraum,
                                 Dokumentname,
                                 Dokumenttyp);
// z.B. systemID="Students.dtd"
DocumentType dt = di.createDocumentType(Dokumentname,
                                        publicID,
                                        systemID);
doc.appendChild(dt);
```

2. Erstellen von Elementen im Baum

1. Ermitteln des Wurzelknoten

```
Element students = doc.getDocumentElement()
```



2. Erzeugen/Hinzufügen neuer Knoten

```
Element el_2 = doc.createElement("Student");
// Knoten der zweiten Ebene
students.appendChild(el_2);
```

3. Hinzufügen weiterer Knoten mit Attributen/Texten

```
Element el_31 = doc.createElement("name");
// Knoten der dritten Ebene
el_2.appendChild(el_31);
Text text = doc.createTextNode();
// Knoten der vierten Ebene
el_31.appendChild(text);
Element el_32 = doc.createElement("location");
el_2.appendChild(el_32); ...
```

3. Eintragen von Attributen/Werten

Behandlung von Textknoten

```
Element el_31 = doc.createElement("name");
el_2.appendChild(el_31);
Text text = doc.createTextNode("Steffen");
el_31.appendChild(text);
Element el_32 = doc.createElement("location");
el_2.appendChild(el_32);
text = doc.createTextNode("Tutorium 1");
el_32.appendChild(text);
```

Hinzufügen von Attributen

```
// Variante 1
el_x.setAttribute(AttributName, Attributwert);
```

```
// Variante 2
Attr at = doc.createAttribute(AttributName);
at.setNodeValue(Attributwert);
el_x.setAttributeNode(at);
```

4. Ausgabe in XML-Datei

Beispiel mit XML-Serializer aus org.apache.xml.serialize.*

- Serialisieren: Überführen der Baumstruktur in Zeichenfolgen
- Formatierung: Strukturierung der Ausgabe (Einrückungen)

```
1  // Datei festlegen
2  FileWriter fw = new FileWriter("Ausgabe.xml");
3  // Formatierung notwendig
4  OutputFormat format = new OutputFormat(doc);
5  // Einrückung und Zeilenumbruch erwünscht
6  format.setIndenting(true);
7  // zwei Zeichen pro Einrückung
8  format.setIndent(2);
9  // Steuerzeichen für neue Zeile
10 format.setLineSeparator("\r\n");
11 // Zeichenkodierung
12 format.setEncoding("ISO-8859-1");
13 // Erzeugen und Starten der Serialisierung
14 XMLSerializer serializer = new XMLSerializer(fw,format);
15 serializer.serialize(doc);
```

4. Ausgabe in XML-Datei (Alternative)

Beispiel mit XML-Transformer aus javax.xml.transform.*

```
1  // Datei festlegen
2  FileWriter fw = new FileWriter("Ausgabe.xml");
3  // Transformer-Factory erzeugen
4  TransformerFactory f = TransformerFactory.newInstance();
5  // Transformer erzeugen
6  Transformer t = f.newTransformer();
7  // Typ der Datei festlegen
8  t.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,
9                      "Student.dtd");
10 // zwei Zeichen pro Einrückung
11 t.setOutputProperty(OutputKeys.INDENT, "yes");
12 t.setOutputProperty("{http://xml.apache.org/xslt}" +
13                      "indent-amount","2");
14 // Erzeugen und Starten der Serialisierung
15 t.transform(new DOMSource(doc), new StreamResult(fw));
```

XML-Daten in DOM-Bäumen verändern

Vorgehen

- Baum traversieren, bis gewünschter Knoten gefunden
- Auslesen von Attributen/Werten
- Setzen/Löschen von Attributen/Werten

```
// Attribut löschen
void removeAttribute(String name)
// Werte neu setzen
void setNodeValue(String nodeValue)
// Kopieren eines Knotens
// "deep=true" Kopieren auch aller Kindknoten
Node cloneNode(boolean deep);
// Löschen des übergebenen Kindknotens
Node removeChild(Node oldChild)
```