XML-basierter Datenaustausch in Java

PROG 2: Einführung in die Programmierung für Wirtschaftsinformatiker

---

Steffen Helke
Technische Universität Berlin
Fachgebiet Softwaretechnik

6. Mai 2013

Übersicht

- Wiederholung: Streams für I/O in Java
- XML, DTD und XML-Schemata
- XML-Verarbeitung mit SAX
- XML-Verarbeitung mit DOM

Teil II der Vorlesung PROG 2

Ein- und Ausgabe mit Dateien

Dateiverarbeitung in Java

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

Dateiverarbeitung in Java

Verarbeitungslevel

- Java setzt *keine Struktur* voraus
- Dateien werden als Folgen von Zeichen interpretiert
- Verarbeitung von Zeichenfolgen (*Streams*)

Vorgehen

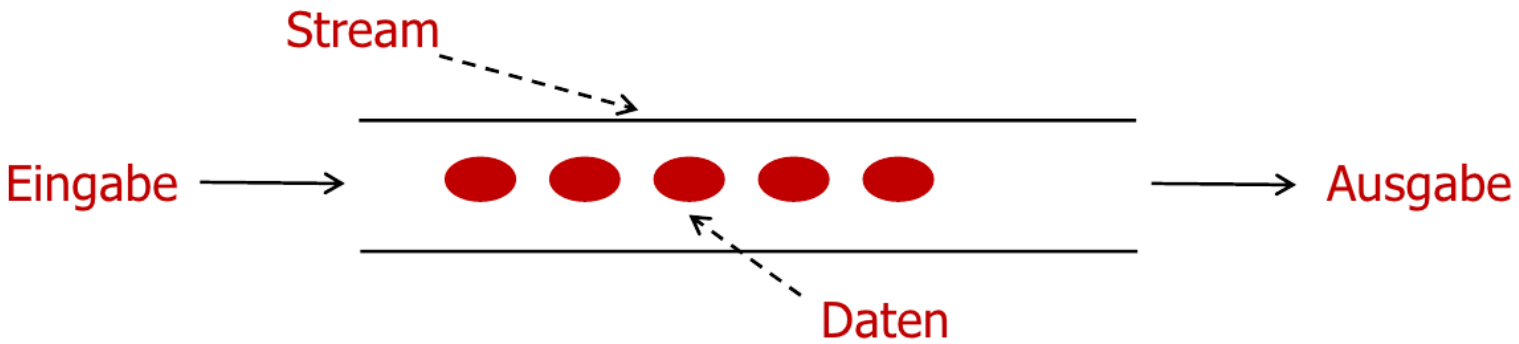
1. Öffnen der Datei

   ⇒ Verbindungsaufbau zwischen Datei und Programm
2. Bearbeiten der geöffneten Datei

   ⇒ Streams werden gelesen oder geschrieben
3. Schließen der Datei

   ⇒ Herstellung eines definierten Zustands

Streams in Java



Varianten

1. Byte-basierte Streams
   - Daten werden als Bytes gelesen/geschrieben
   - geeignet *für binäre Dateien*
2. Charakter-basierte Streams
   - Daten werden als alpahnumerische Zeichen (16 Bit Unicode-Zeichen) gelesen/geschrieben
   - geeignet *für textuelle Dateien*

Lesen und Schreiben von Dateien

Herausforderung

- Auswahl der richtigen Werkzeuge
- *mehr als 20 Dateizugriffsklassen* verfügbar

Unterscheidung

1. Sequentieller Zugriff (*stream access*): Elemente als Zeichenfolge, Bearbeitung erfolgt streng nacheinander
2. Wahlfreier Zugriff (*random access*): Dynamische Bearbeitung an gewählten Positionen möglich

Komfortable Funktionen in java.io.*

- Lesen/Schreiben aus Datei: *FileInputStream/FileOutputStream*
- Zeichenbasierte Ein- und Ausgabe: *FileReader/FileWriter*

Stromarten (Streams) in Java

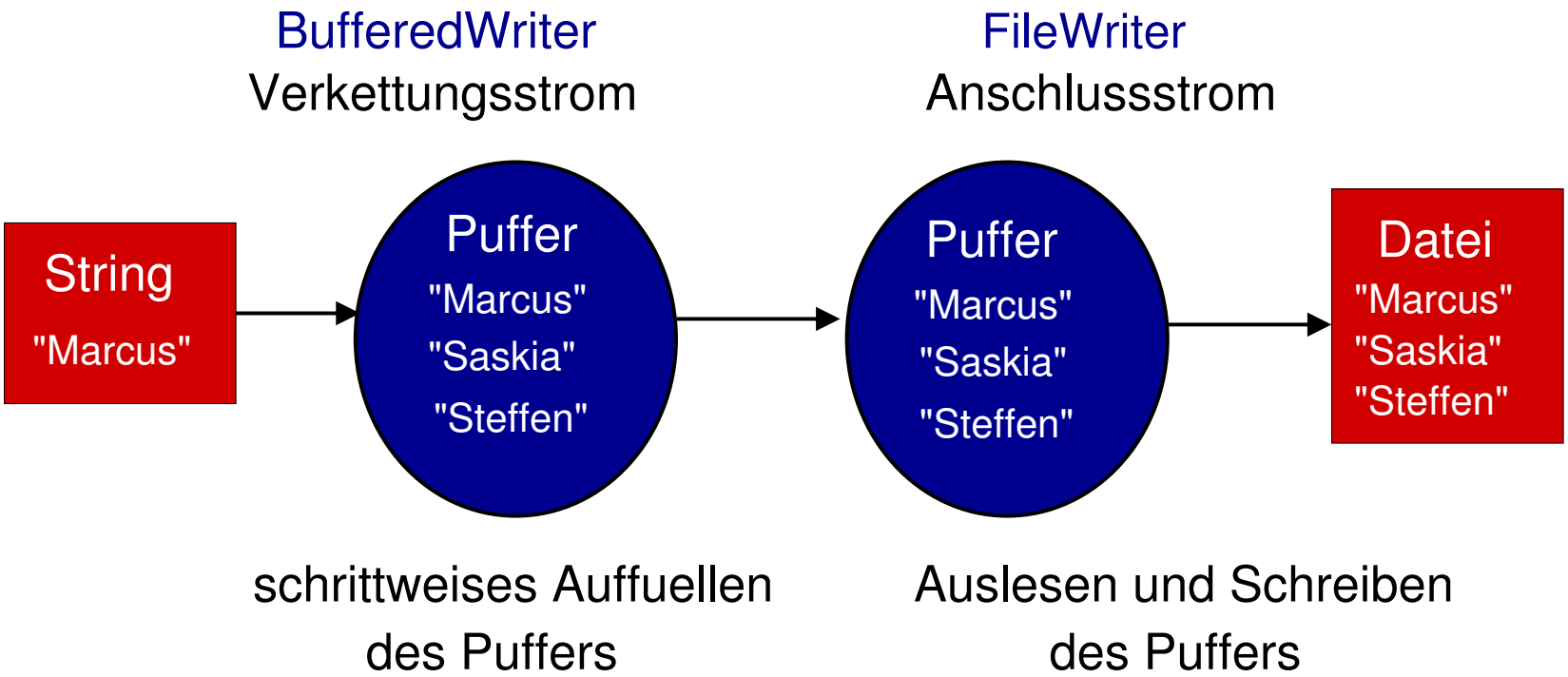
Anschlussstrom

- bildet Verbindung zu Dateien
- bietet *low-level* Methoden zum Lesen/Schreiben von Bytes

Verkettungsstrom

- bildet Verbindung zum Objekt
- bietet *high-level* Methoden zum Lesen/Schreiben
- muss mit anderen Strömen verkettet werden, z.B. *BufferedWriter* mit *FileWriter*

Beispiel: Puffern beim Speichern



Teil III der Vorlesung PROG 2

XML und Java

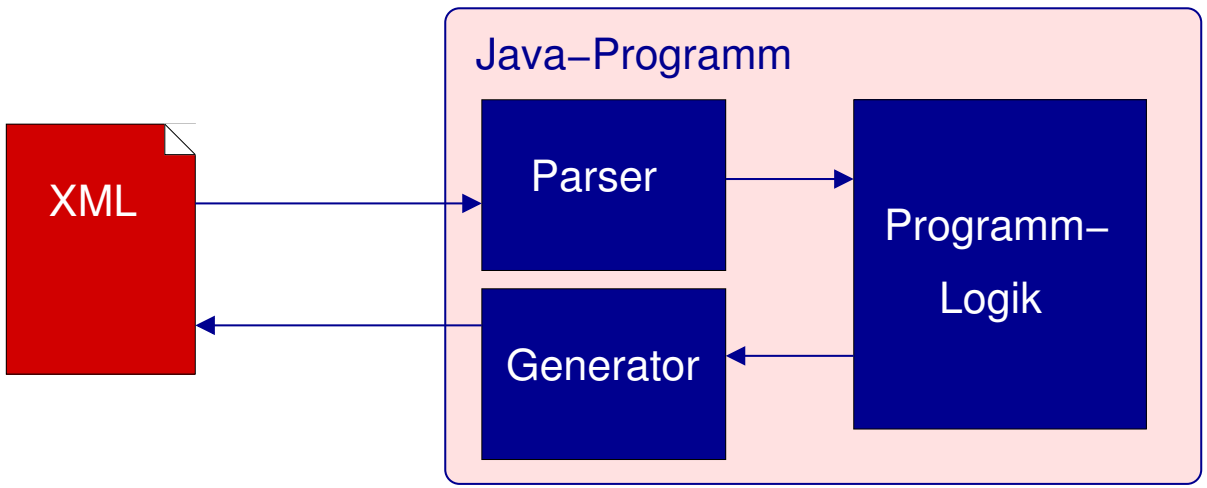
Grundlegende Einführung: Extensible Markup Language

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

Grundprinzipien von XML

Einfachheit

- Bearbeitung mit Texteditoren
- Menschen-lesbarer Text
- keine Binärdaten

Portierbarkeit

- Verwendung von Unicode-Zeichen
- Speicherung als ASCII-Text

Erweiterbarkeit

- Erstellung eigener Dokumententypen
- Hinzufügen neuer Elemente

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

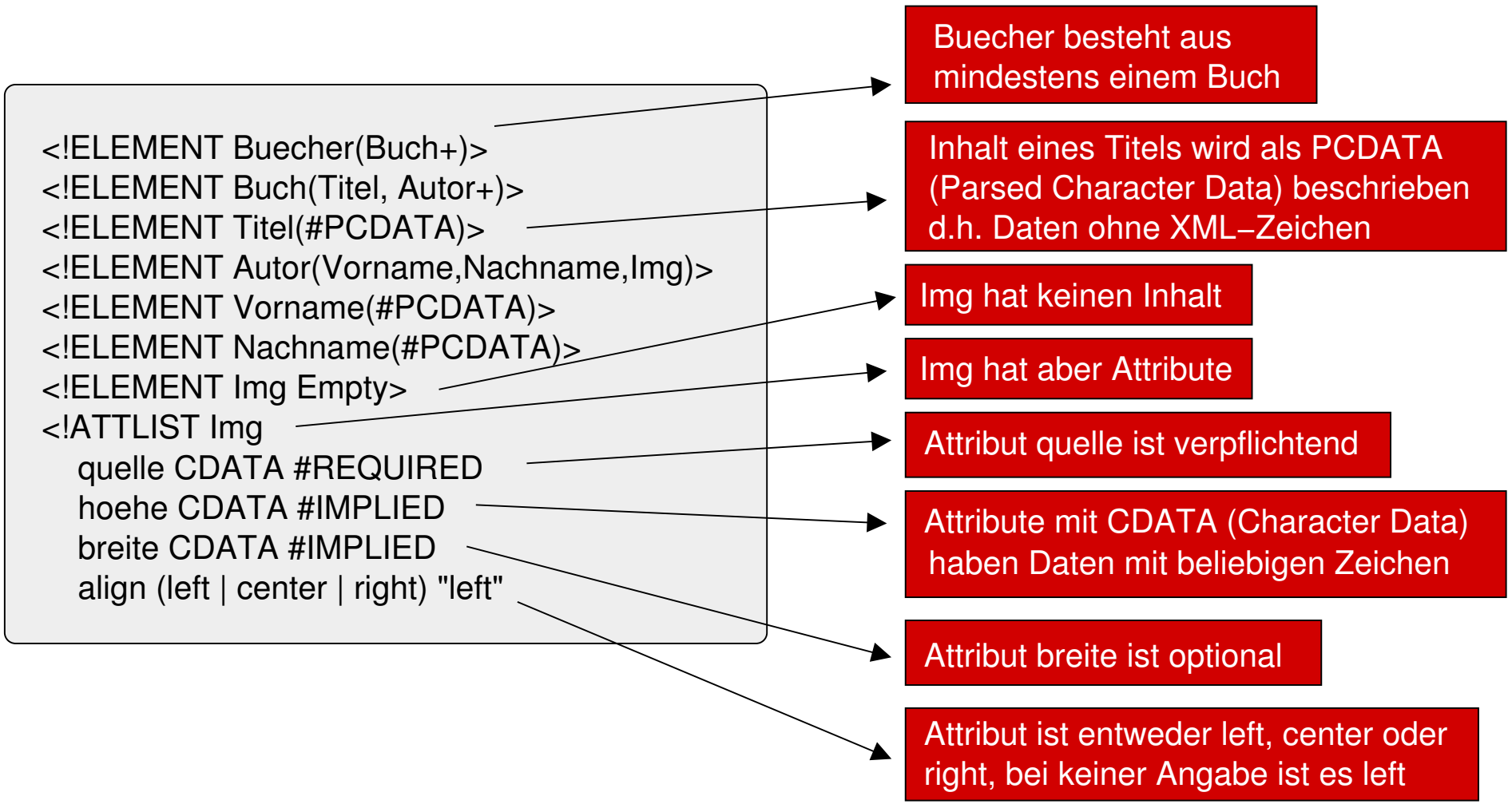
Buecher.dtd

```
<!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

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

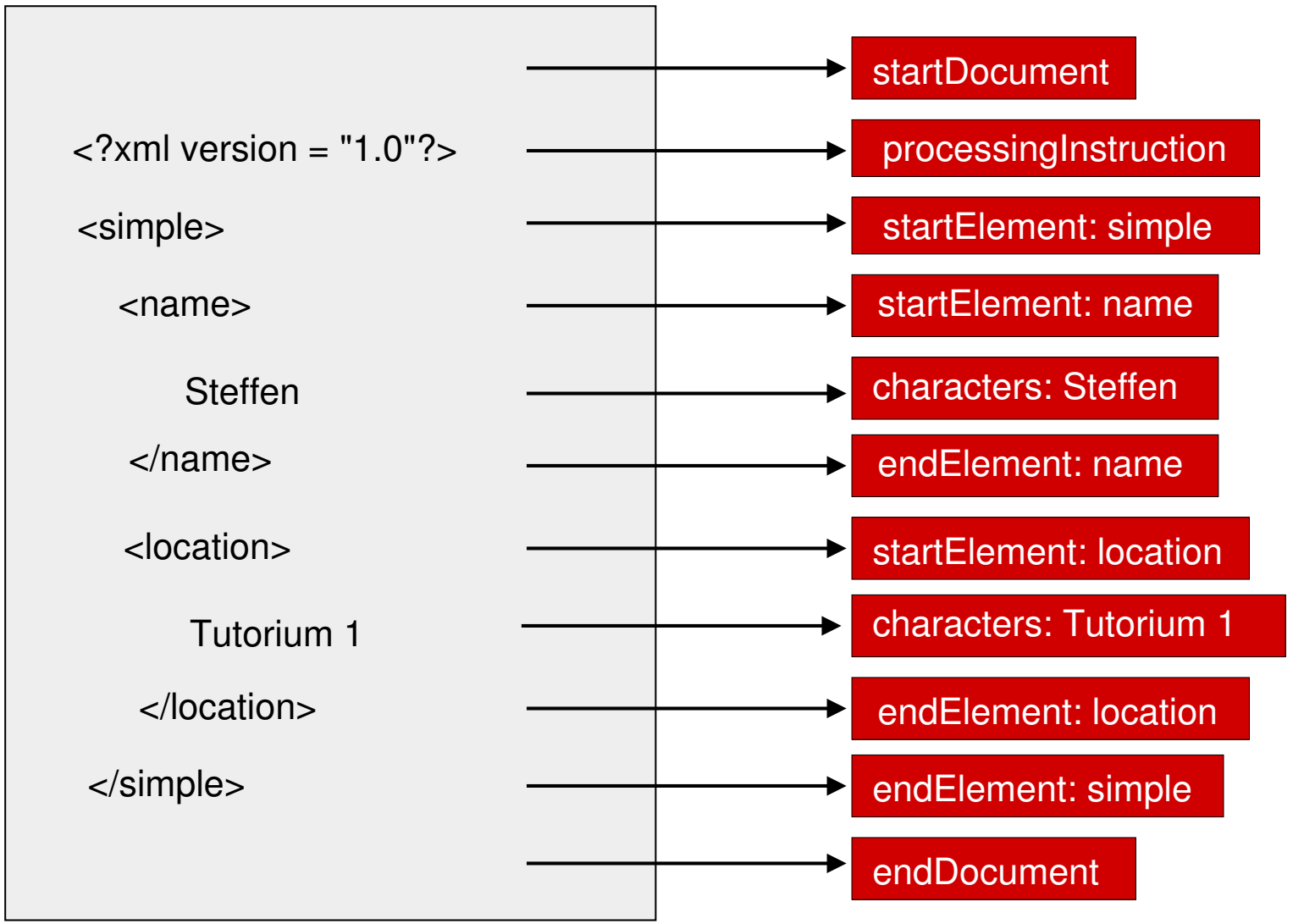
Idee

- 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 }
```

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