

## Übung Netzbasierte Informationssysteme

### Termin 3: XML Processing

**Prof. Dr. Adrian Paschke**

Arbeitsgruppe Corporate Semantic Web (AG-CSW)  
Institut für Informatik, Freie Universität Berlin

[paschke@inf.fu-berlin.de](mailto:paschke@inf.fu-berlin.de)

<http://www.inf.fu-berlin.de/groups/ag-csw/>



# Agenda

---

- Feedback Übungsblatt 2
- Themen des heutigen Übungsblatts:
  - Filehandling mit Java
  - XML Basics
    - DTD
    - Schema
    - Parser
  - eXtensible Stylesheet Language (XSL)
    - XML Path Language (XPath)
    - Extensible Stylesheet Language Transformations (XSLT)
  - XML Query Language (XQuery)
- Übungsblatt 3

# Feedback Übungsblatt 3

---

- **Lernziele des 2. Übungsblattes waren:**
  - Java Servlets: Ergebnis- und Requestweiterleitung
  - JSPs: Basics
- **Wie war`s?**
- **Probleme/Anregungen?**

# Filehandling mit Java

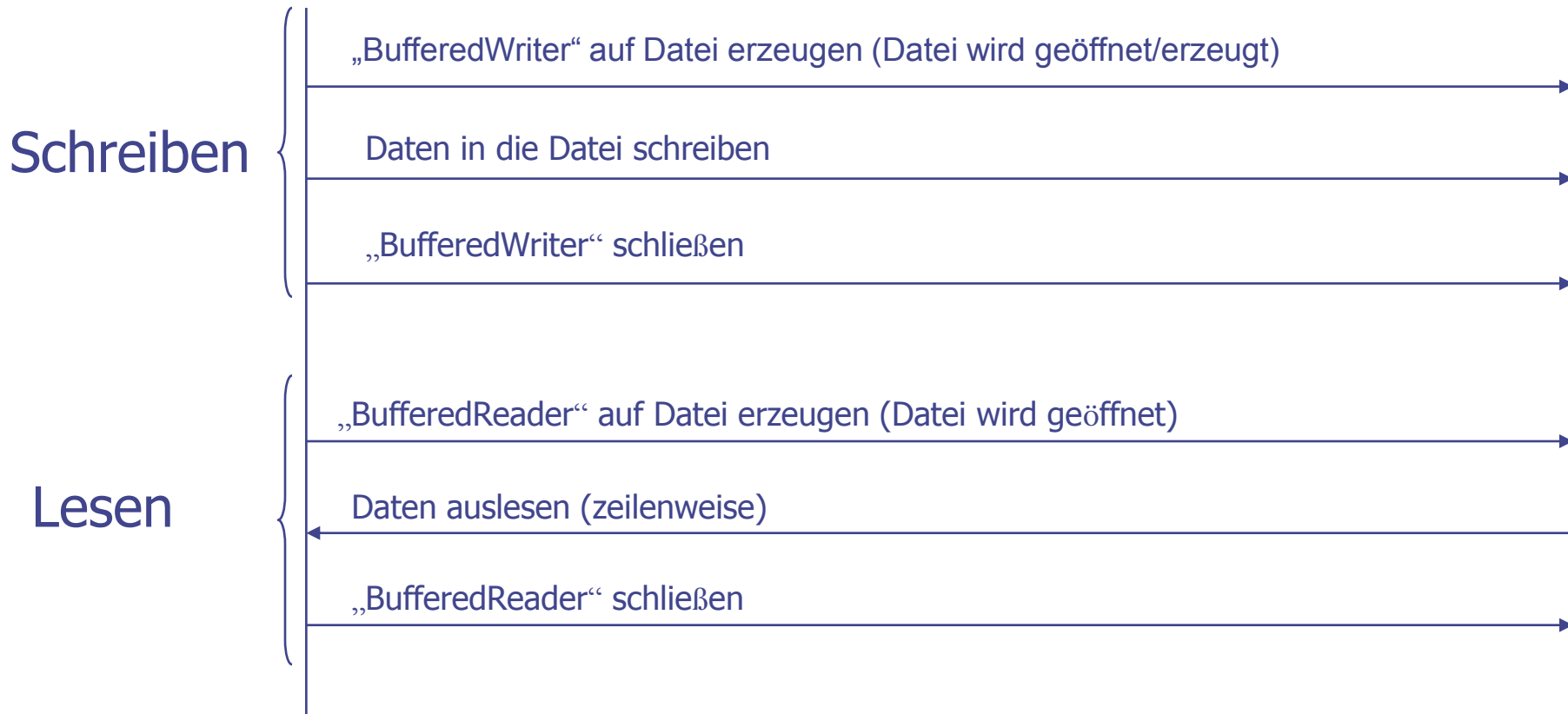
---

- Zugriff auf Daten mittels **Datenstrom** (stream)
  - Eingabestrom – input stream
  - Ausgabestrom – output stream
- „**File**“-**Objekt** liefert Zugang zu Datei und Verzeichnis
  - Verweis wird durch Pfadnamen spezifiziert
  - absolut oder relativ zum aktuellen Verzeichnis
- **Achtung: Plattformabhängigkeit!!**
  - Pfadangabe - Win: “temp\files”; “Unix: temp/files”
  - FileSeparator in Attribut separatorChar der Klasse File auslesbar
  - Wurzelverzeichnis: Unix (“/”), Windows (“C:\”)

# Filehandling - Schematischer Ablauf

Servlet

File



# Filehandling – Datei schreiben

---

- `java.io.*` importieren
- Öffnen/Anlegen der zu schreibenden Datei, „Writer“ erzeugen
  - `BufferedWriter out =`  
`new BufferedWriter(`  
`new OutputStreamWriter(`  
`new`  
`FileOutputStream („<pfad+filename>*“,`  
`true));` -- true steht für „append“
- Daten in Datei schreiben
  - `out.write(<String>);`
- Datei schließen
  - `out.close();`

# Filehandling – Datei lesen

---

- Öffnen der zu lesenden Datei, „Reader“ erzeugen
  - ```
BufferedReader in =  
    new BufferedReader(  
        new InputStreamReader(  
            new FileInputStream(new File(url))));
```
- Daten zeilenweise auslesen
  - ```
String buffer;  
    while (true) {  
        buffer=in.readLine();  
        if (buffer==null) break;  
    }
```
- Datei schließen
  - ```
in.close();
```
- Tipp: Methoden der Klasse `java.util.StringTokenizer` können zum Parsen separierter Daten verwendet werden

# XML – Extensible Markup Language

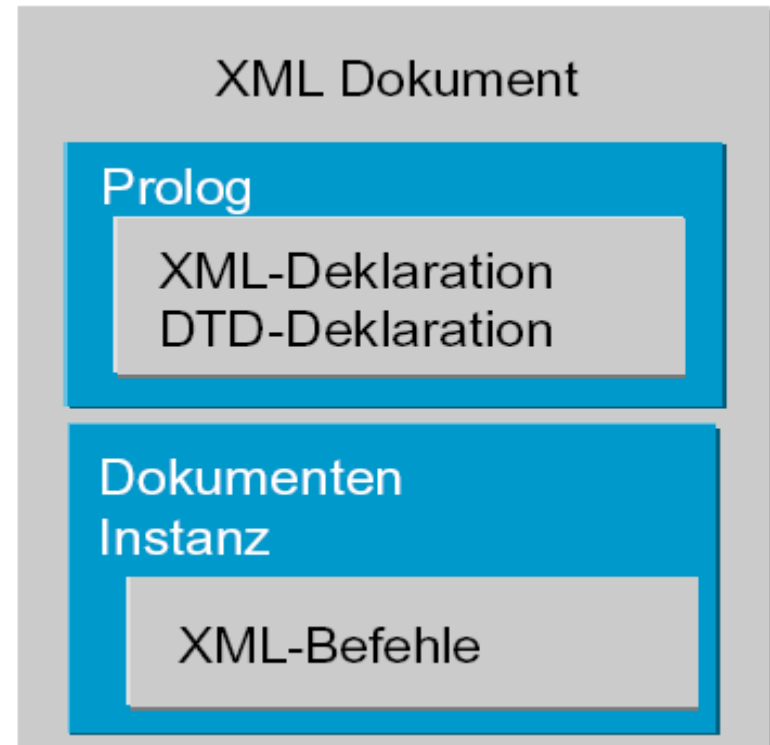
---

- Standard zur Erstellung von Dokumenten in Baumstruktur
- Vom World Wide Web Consortium (W3C) definiert
- Trennung von Datenbasis und Repräsentation
- XML-Dokument ist “wohlgeformt” falls es alle Regeln der Spezifikation einhält (z.B. Tags geschlossen etc)
- Format eines XML-Dokuments kann durch eine Grammatik definiert werden
  - durch DTD (Dokumenttypdefinition) oder ein XML-Schema
  - Falls es einer DTD/Schema entsprechend aufgebaut ist, heißt es “gültig”
- XML-Parser sind Programme, die XML-Daten auslesen und interpretieren (z.B. auf Gültigkeit prüfen)

# Struktur eines XML Dokuments

---

```
<?XML Version="1.0" ENCODING="ISO-8859-1"?>
<!DOCTYPE PersonenDB SYSTEM „person.dtd“>
<PersonenDB>
  <Person id="1">
    <Nachname>Paschke</Nachname>
    <Vorname>Adrian</Vorname>
  </Person>
  <Person id="2">
    <Nachname>Maier</Nachname>
    <Vorname>Hans</Vorname>
  </Person>
</PersonenDB>
```



# DTD – Document Type Definition

- Beschreibt die logische und physikalische Struktur eines XML-Dokuments (Grammatik)
- Nicht XML-Syntax
- Kennt 6 Typen, Hauptsächlich
  - Elemente
  - Attributlisten
  - Kommentare
  - Entities
  - Notationen
  - Prozess-Instruktionen
- Rudimentäre Datentypen
  - PCDATA | CDATA
- In- oder Outline eingebunden

## Beispiel DTD

```
<!ELEMENT Bestelldokument (Lieferadresse, Bestellung, Zahlung)>
<!ATTLIST Bestelldokument datum CDATA #REQUIRED
                                prio ( hoch | niedrig ) "niedrig"
>
<!ELEMENT Lieferadresse (#PCDATA)>

<!ELEMENT Bestellung (Artikel+)>
  <!ELEMENT Artikel (Artikelnummer, Beschreibung, Anzahl)>
    <!ELEMENT Artikelnummer (#PCDATA)>
    <!ELEMENT Beschreibung (#PCDATA)>
    <!ELEMENT Anzahl (#PCDATA)>

<!ELEMENT Zahlung (Zahlungsart | Zusatzangaben?)>
  <!ELEMENT Zahlungsart (#PCDATA)>
  <!ELEMENT Zusatzangaben (#PCDATA)>
```

\* + ?

# XML-Schema

---

- (Mächtiger) Alternative zu XML-DTDs
- ... mit folgenden Unterschieden:
  - Datentypen – Typgenauigkeiten auf Datenbankniveau
  - Schemas sind selbst in XML-Notation
  - Unterstützung von XML-Namespaces
  - Genaue Kardinalitätsrestriktionen
  - Möglichkeiten der OO-Modellierung (Subklassenbildung)
  - SimpleTypes vs ComplexTypes
    - SimpleType z.B. „string“
    - ComplexType entspricht z.B. einem Record
  - Es gibt ein Schema für XML Schema ...
- mehr Infos zu XML-Schema -> Online-Referenzen

# XML Werkzeuge: Parser

---

- XML-Files Auslesen und Verarbeiten in Programmiersprachen wie z.B. Java
- Früher: Zu jedem Parser gehörte eine eigene API
- Heute: Standards SAX, DOM, JAXP
  - SAX: event-basiert
  - DOM: Baum
  - JAXP: Sun's Java API for XML Processing (unterstützt SAX, DOM und XSLT)

# SAX

---

- Simple API for XML Processing
- SAX agiert ereignisgesteuert
  - XML Prozessor parst Dokument, beim Eintritt bestimmter Ereignisse erhält SAX ein Callback vom Parser, wodurch eine Methode der SAX-Klasse aktiviert wird.
  - Diese Methoden sind zu überschreiben; mit gewünschter Funktionalität zu füllen.
- SAX hält nur den gerade bearbeiteten Teil des Dokuments im Arbeitsspeicher (im Gegensatz zu DOM)  
  
→ SAX durchläuft dem Baum sequentiell, beginnt jeden Suchvorgang am Dokumentanfang

# SAX API

---

- Zu überschreibende Java SAX Ereignis-Methoden
  - startDocument()
  - startElement()
  - endElement()
  - ...
- SAX Klasse erweitert DefaultHandler
- SAXParserFactory API stellt Parser bereit
- Siehe SAX-Beispiel auf der Homepage oder auch zahlreiche andere Beispiele im Netz

# SAX

---

## Wie lese ich ein xml-file mit SAX?

```
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.Parser;
import org.xml.sax.ParserFactory;
import org.xml.sax.SAXException;
import java.io.IOException;

...

String xmlFile = "file:///xerces-1_4_4/data/personal.xml";

String parserClass = "org.apache.xerces.parsers.SAXParser";
Parser parser = ParserFactory.makeParser(parserClass);

try {
    parser.parse(xmlFile);
} catch (SAXException se) {
    se.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

# SAX – ContentHandler Interface

```
package org.xml.sax;

public interface ContentHandler {

    public void setDocumentLocator(Locator locator);

    public void startDocument() throws SAXException;

    public void endDocument() throws SAXException;

    public void startPrefixMapping(String prefix, String uri)
        throws SAXException;

    public void endPrefixMapping(String prefix) throws SAXException;

    public void startElement(String namespaceURI, String localName,
        String qualifiedName, Attributes atts) throws SAXException;

    public void endElement(String namespaceURI, String localName,
        String rawName) throws SAXException;

    public void characters(char[] text, int start, int length)
        throws SAXException;

    public void ignorableWhitespace(char[] text, int start, int length)
        throws SAXException;

    public void processingInstruction(String target, String data)
        throws SAXException;

    public void skippedEntity(String name) throws SAXException;

}
```

# DOM – Document Object Model

---

- Document Object Model (DOM)
  - Standardisierte API zu Inhalten eines XML-Dokuments.
  - W3C Recommendation
  - Sehr beliebt unter Entwicklern
  - Jedes XML-Dokument ist ein Baum
  - Ein Baum besteht aus Knoten
  - Knoten können andere Knoten beinhalten (je nach Art des Knotens)
  - Alle Inhalte liegen in den Blättern (Elemente, Texte, Kommentare, CDATA-Sektionen, Verweise, Attribute, ...)

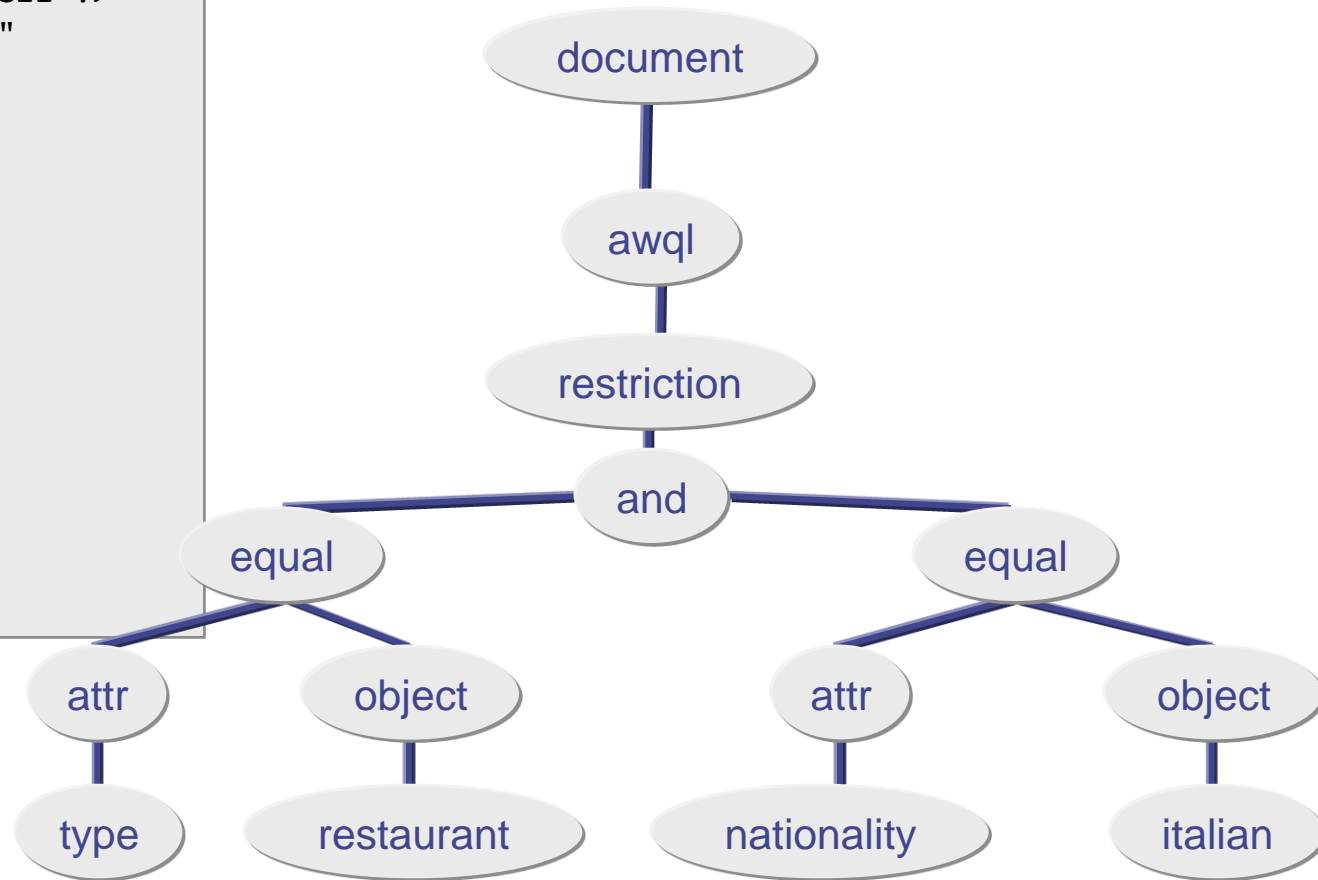
# DOM - Basics

---

- Library-spezifisch wird ein Parser erzeugt
- Dieser parst das Dokument und gibt ein Objekt vom Typ `org.w3c.dom.Document` zurück
- damit liegt das gesamte Dokument im Speicher
- -> Problem bei großen XML-Files!
  
- Jeder “document node” beinhaltet:
  - maximal einen “doctype” Knoten
  - genau einen “root element” Knoten
  - null oder mehr “comment and processing instruction” Knoten
  
- DOM Methoden und Interfaces werden verwendet um Daten aus diesem Objekt zu extrahieren.

# DOM – Baum

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE awql SYSTEM "http:// ... "
<awql>
  <restriction>
    <and>
      <equal>
        <attr name="type"/>
        <object>restaurant</object>
      </equal>
      <equal>
        <attr name="nationality"/>
        <object>italian</object>
      </equal>
    </and>
  </restriction>
  (...)
</awql>
```



# DOM

---

## Wie lese ich ein xml-file mit DOM?

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import java.io.IOException;

...

String xmlFile = "file:///xerces-1_4_4/data/personal.xml";

DOMParser parser = new DOMParser();

try {
    parser.parse(xmlFile);

} catch (SAXException se) {
    se.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}

Document document = parser.getDocument();
```

# XSL – Extensible Stylesheet Language

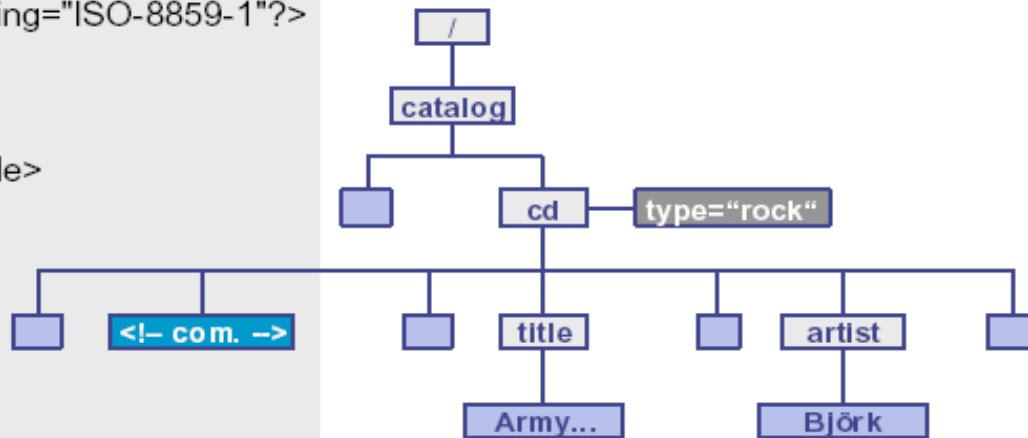
---

- XSL umfasst **drei Technologien**:
  - XML Path Language (XPath)
  - XSLT
  - XSL-FO (Formatting Objects)
- XPath ermöglicht den **Zugriff auf bestimmte Elemente** eines XML-Files mittels spezieller Ausdrücke
- XSLT **transformiert Inhalte** eines XML-Files in anderes XML-Format oder HTML
- XSL-FO ermöglicht die **Dokumentformatierung** (nicht sehr weit verbreitet, daher bei uns nicht genauer)

# XPath - Basics

- **XPath** modelliert XML-Dokument als **Baum**, der aus Knoten besteht.
- verschiedene **Knotentypen**, unter anderem Elementknoten, Attributknoten und Textknoten.
- XPath verwendet **Lokalisierungspfade** wie in einem Dateisystem, um Knoten auszuwählen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd type="rock">
    <!-- -comment-->
    <title>Army Of Me</title>
    <artist>Björk</artist>
  </cd>
  .
  .
  .
</catalog>
```



Beispiel: Wähle alle „Artisten“ aus dem Dokument:

`/child::catalog/child::cd/child::artist` oder `/catalog/cd/artist`

# XSLT – Extensible Stylesheet Language

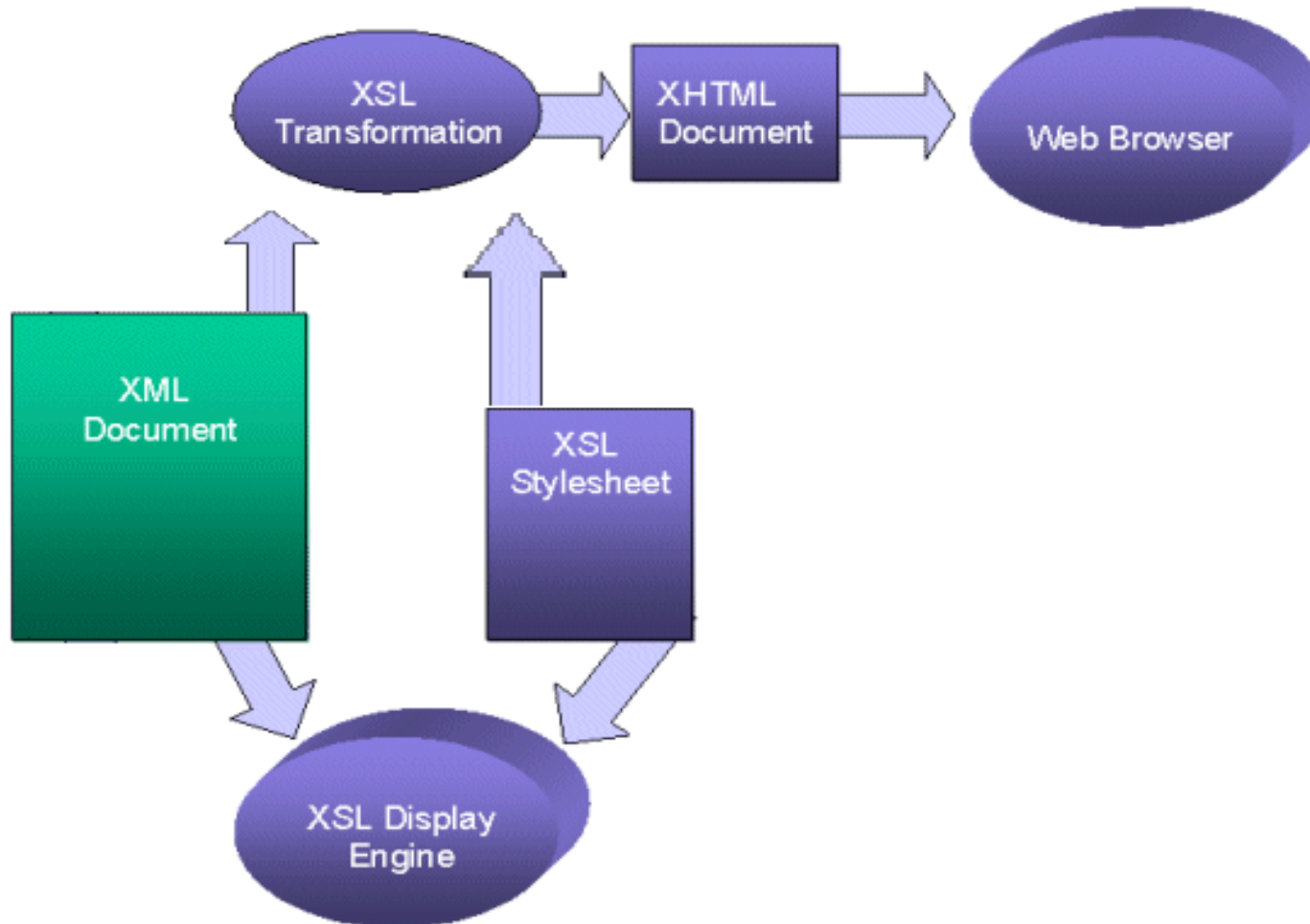
## Transformations

---

- XSLT transformiert (Teile von) XML-Dokumente(n) in eine andere Form (Syntax).  
(z.B. von XML in anderes XML-Format, HTML, ...)
- Um eine **Transformation** durchzuführen werden
  - ein XML Dokument
  - eine Stylesheet-Datei benötigt
- Die **Stylesheet-Datei** enthält **Transformationsvorschriften**
  - Transformationsprozess beginnt immer beim Wurzelement und verarbeitet dann das Dokument.

# XSLT - Ablauf einer Transformation

---



# XSLT - Beispiel

## XML-file:

```
<catalog>
  <cd>
    <title>Heroes</title>
    <artist>Bowie</artist>
  </cd>
</catalog>
```

## stylesheet:

```
<xsl:stylesheet>
  <xsl:template match="/">
    <html><body><h1>
      <xsl:value-of select="/catalog/cd/title"/>
    </h1></body></html>
  </xsl:template>
</xsl:stylesheet>
```



## output:

```
<html><body><h1>
  Heroes
</h1></body></html>
```

1

2

3

4

5

6

# XSLT – Werkzeug Xalan

---

- Xalan
  - Vollständige Implementierung von XSLT und XPath (Version 1.0)
  - Benutzt Apache Xerces als Parser
  - Kann z.B. in Servlet benutzt werden um XML-Dokumente in HTML zu transformieren
  - damit in einem Browser übersichtlich darstellbar

Anwendung:

```
C:\jdk1.5...\bin\java -cp "C:\...\xalan.jar;
```

```
C:\...\xerces.jar" org.apache.xalan.xslt.Process  
-IN sourceFile.xml -XSL stylesheet.xslt  
-OUT outputFile.xml
```

# XSLT – Beispiel 2

## stylesheet:

```
<xsl:stylesheet version="1.0">
<xsl:template match="/">
  <HTML>
    <BODY>
      <TABLE BORDER="2">
        <TR>
          <TD>Name</TD>
          <TD>Address</TD>
          <TD>Tel</TD>
          <TD>Fax</TD>
          <TD>Email</TD>
        </TR>
        <xsl:for-each select="PEOPLE/PERSON">
          <TR><TD>
            <xsl:value-of select="NAME"/>
          </TD><TD>
            <xsl:value-of select="ADDRESS"/>
          </TD><TD>
            <xsl:value-of select="TEL"/>
          </TD><TD>
            <xsl:value-of select="FAX"/>
          </TD><TD>
            <xsl:value-of select="EMAIL"/>
          </TD>
        </TR>
        </xsl:for-each>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>
</xsl:stylesheet>
```



## XML-File:

```
<?xml version="1.0"?>
<PEOPLE>
  <PERSON PERSONID="p1">
    <NAME>Thomas Cook</NAME>
    <ADDRESS>Hauptstr. 9, 80845 Alp</ADDRESS>
    <TEL>012/43 56 43</TEL>
    <FAX>012/43 56 44</FAX>
    <EMAIL>thomascook@example.com</EMAIL>
  </PERSON>
  <PERSON PERSONID="p2">
    <NAME>Mia Wilson</NAME>
    <ADDRESS>Mid 1, 86323 HelpCity</ADDRESS>
    <TEL>02721/289 9090</TEL>
    <FAX>02721/289 9091</FAX>
    <EMAIL>miawilson@example.com</EMAIL>
  </PERSON>
</PEOPLE>
```

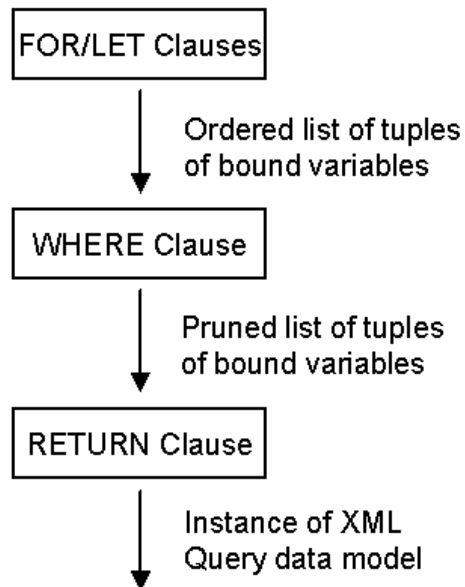
# XQuery - Basics

---

- **XML Query Language**
  - XML-Anfragesprache für den **Zugriff auf XML-Daten**
  - Soll ähnliches leisten wie SQL für Zugriff auf relationale DBMS
  - Verwendet XPath und XML Schema
- **Anfragen an:**
  - XML-Dokumente
  - XML-Datenbanken
- **Eigenschaften**
  - flexibel: Anfrage an unterschiedliche XML Informationsquellen
  - vom Menschen lesbarer Anfragesyntax
  - XML-basierte Anfragesyntax (XQuery X-Syntax)

# XQuery – Eine Beispielabfrage

## ■ FLWR Anfragen:



```
FOR $b
IN document('bib.xml')//book
LET $name:=$b/author/lastname $title:=$b/title
WHERE $b/publisher/name = 'Addison-Wesley'
RETURN
<result>
  <author><lastname>{$name}</lastname></author>
  <title> {$title} </title>
</result>
```

Ergebnis:

```
<result>
  <author><lastname>Gray</lastname></author>
  <title> Web Server Programming </title>
</result>
```

# Sourcen

---

- **Zahlreiche Literatur und Beispiele im Netz**
  - **XSL**  
the Extensible Stylesheet Language Family (XSL): <http://www.w3.org/Style/XSL/>
  - **XPath**  
XML Path Language (XPath): <http://www.w3.org/TR/xpath>  
XML Path Language (XPath) 2.0: <http://www.w3.org/TR/xpath20/>
  - **XSLT**  
Wikipedia – Page: <http://en.wikipedia.org/wiki/XSLT>  
Tutorial: <http://www.topxml.com/xsl/tutorials/intro/default.asp>
  - **XQuery**  
XQuery 1.0: An XML Query Language: <http://www.w3.org/TR/xquery/>  
Tutorial: <http://www.w3schools.com/xquery/default.asp>

# Übungsblatt 3

---

- **Lernziele**

- Filehandling mit Java
- XML-Basics
  - DTD
  - Schema
  - Parser
- XPath
- XQuery
- XSLT

- **Termine**

- Ausgabe: 11.11.2008 Abgabe bis: 18.11.2008, 16.00 Uhr



**Viel Spaß und Erfolg !**

