XML APIs, DOM und SAX

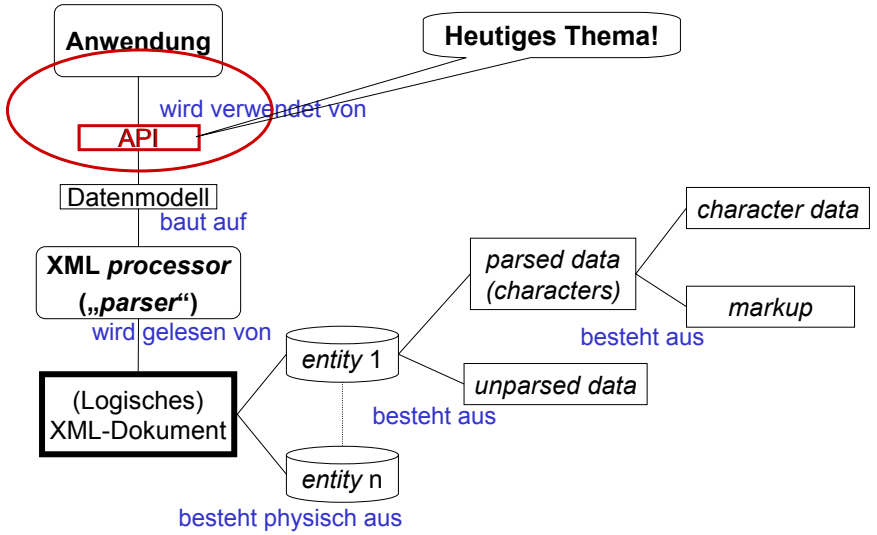
XML aus Sicht der Software-Entwickler

Mit freundlicher Unterstützung von Prof. Weitz
(danke für die Java-Folien zu DOM & SAX!)

XML APIs allgemein

Welche Möglichkeiten bestehen?
Vor- und Nachteile

Erinnerung



Ansprüche an XML APIs

- **Sprachintegration**
  - API sei optimiert für “meine Lieblingssprache”, oder
  - API sei möglichst standardisiert für viele Sprachen
- **R/O oder R/W**
  - Nur lesender Zugriff vs.
  - Auch Ändern vorhandener XML-Dokumente und evtl.
  - Aufbau kompletter eigener XML-Dokumente
- **Random access vs. streaming mode**
  - Sequenzielles Lesen: Schnell, speichersparend
  - Speicherresidentes Modell des Dokuments: Flexibel
- **Interne Lösung oder externe (per Script)**

XML APIs: Klassifikation typischer Lösungen

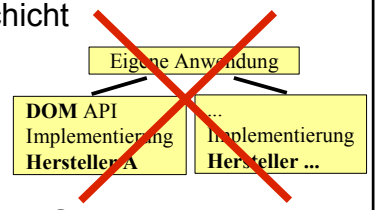
- Ruby's **REXML** - Beispiel für tiefe Sprachintegration
  - Proprietär, perfekt auf Ruby abgestimmt; „alles" möglich.
- **XSLT**
  - Scriptsprache, für externe Lösungen
  - *Random access* möglich, R/O, Aufbau neuer Dokumente
  - Aus vielen Sprachen heraus einsetzbar
- **JAXP** (Java API for XML Processing)
  - Herstellerunabh. Zwischenschicht für SAX, DOM & XSLT
- **SAX**
  - Standard-API, *streaming*, R/O
- **DOM**
  - Standard-API, *random access*, R/W

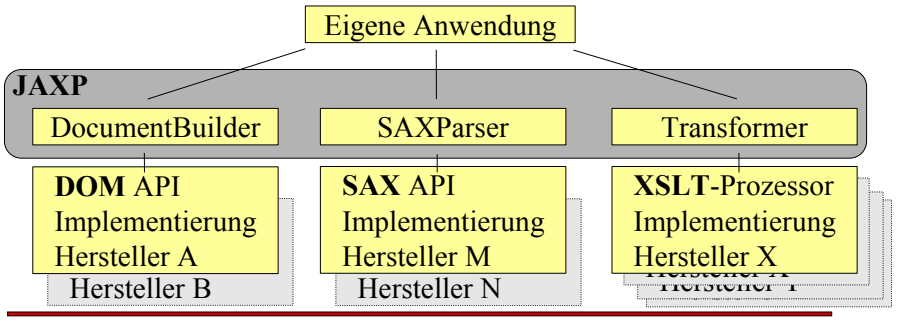
Erinnerung: REXML-Beispiel

```
#!/usr/bin/env ruby
#
require "rexml/document"
include REXML        # Vermeidet Präfix „REXML::"

# XML-Dokument als Datenstruktur in den Speicher laden:
doc = Document.new File.new( "08-bestell.xml" )

# Liste aller Belegnummern:
doc.elements.each(
   "/Bestellungen/Bestellung/Bestellkopf") do |element|
       puts element.elements["Belegnummer"].text
end

# Rollen der Handelspartner:
doc.elements.each("//Bestellkopf/Handelspartner") { |element|
       puts element.attributes["Rolle"]
}
```

JAXP – Java API for XML Processing

- **Herstellerunabhängige** Zwischenschicht für XML-Dienste
- Herstellereigene Implementierungen „einstöpselbar“, **austauschbar**
- So:
  Unabhängigkeit / **Übertragbarkeit** der Gesamtanwendung
- Mehr dazu unter http://java.sun.com/xml/jaxp/index.html





SAX

Simple **A**PI for **X**ML

http://www.saxproject.org

SAX

- **Herkunft**
  - Ein echtes Produkt des Internet
    entstanden aus Diskussionen in der Usenet-Liste *xml-dev*
    Keine Gremienlösung, nicht vom W3C koordiniert
  - Ca. 85 Urheber (*contributors*), herstellerneutral
  - SAX 1.0 11. 05. 1998
  - SAX 2.0.1 29. 01. 2002
  - SAX 2.0.2 27. 04. 2004 aktuelle Version
- **Ziele & Eigenschaften**
  - Einheitliches, einfaches, schnelles Java API für XML
  - Ereignisgesteuert, nur lesender Zugriff
  - Speichereffizient: Keine interne Repräsentation des gesamten Dokuments erforderlich (aber möglich!)

SAX XML-Parser: Idee

XML-Eingabe

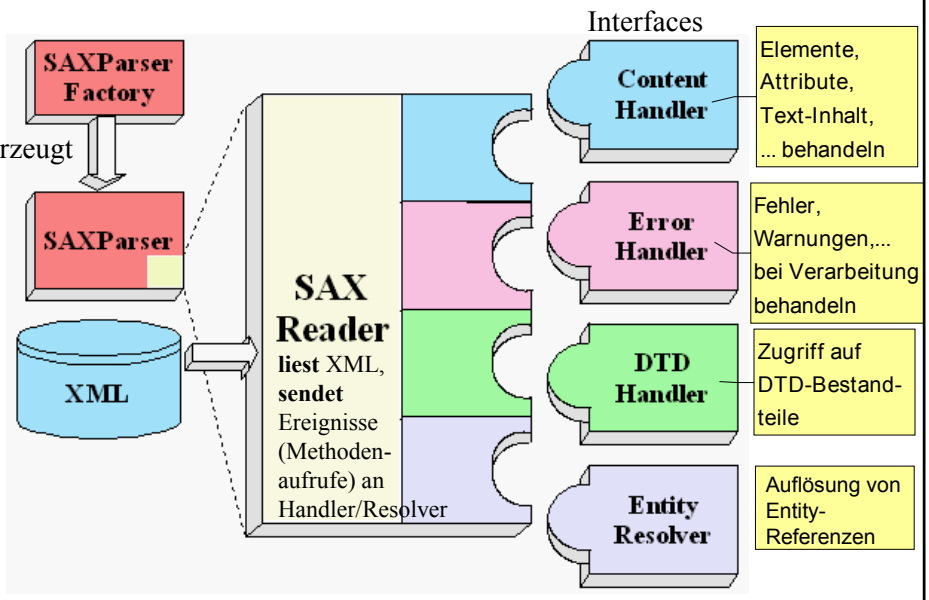
```
<?xml version="1.0"?>
<telefonbuch>
  <!-- erster Eintrag -->
  <eintrag art="mobil">
    <name>Otto</name>
    <nr>0171/12345</nr>
  </eintrag>
</telefonbuch>
```

„SAX-Ereignisse“

startElement „telefonbuch“
Kommentar „erster Eintrag“
startElement „eintrag“
Attribut „art“=“mobil“
startElement „name“
Text „Otto“
endElement „name“
startElement „nr“
Text „0171/12345“
endElement „nr“
endElement „eintrag“
endElement „telefonbuch“

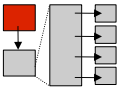
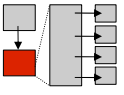
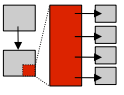
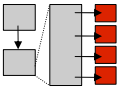
- XML-Eingabestrom „von oben nach unten“ durchlesen, dabei:
- für jedes erkannte XML-Konstrukt (Element, Kommentar, usw.) ein „Ereignis“ an *handler*-Objekt senden
- Ereignis = Methodenaufruf

SAX: Bestandteile



Wichtige SAX-APIs

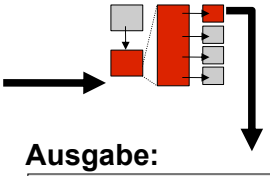
- **SAXParserFactory**
  - **erzeugt** konkreten SAXParser,
  - abhängig von Einstellungen (Property „javax.xml.parsers.SAXParserFactory“)
- **SAXParser**
  - stellt **parse()**-Methode bereit,
  - bedient SAXReader
- **SAXReader**
  - sendet **Ereignisse** an Handler-Objekt(e)
- **DefaultHandler**
  - „tue nichts“-Implementierung der Handler-/Resolver-Interfaces;
  - **Ausgangspunkt** für eigene Handler

SAX-Beispiel: Das Ziel

Eingabe „daten.xml“:

```
<?xml version="1.0"
      encoding="ISO-8859-1"?>

<telefonbuch>
  <eintrag art="firma">
    <!-- erster Eintrag -->
    <name>Willi Wusel</name>
    <nr>0171 / 123456</nr>
  </eintrag>

  <eintrag art="familie">
    <name>Lisa Lustig</name>
    <nr>0161 / 987654</nr>
  </eintrag>

  <!-- und so weiter-->
</telefonbuch>
```



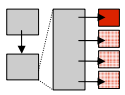
Ausgabe:

```
Element telefonbuch
Element eintrag
  Attr art=firma
Element name
Element nr
Element eintrag
  Attr art=familie
Element name
Element nr
```


MeinHandler.java

```
import org.xml.sax.helpers.*;
import org.xml.sax.*;

public class MeinHandler extends DefaultHandler {

  public void startElement(String namespaceURI,
              String localName,
              String qualifiedName,
              Attributes attrs) throws SAXException
  {
     System.out.println("Element " + localName);
     if (attrs != null) {
          for (int i=0; i < attrs.getLength(); i++) {
               String aname = attrs.getLocalName(i);
               String awert = attrs.getValue(i);
               System.out.println
                    ("  Attr "+aname+"="+awert);
          }
     }
  }
  /* evtl. weitere Methoden... */
}
```

Die „Tue nichts“-Implementierung



Vorgehen bei SAX-Nutzung

- **SAXParserFactory** erzeugen, Namespace-Support an:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
```

- **Handler** erzeugen:

```
DefaultHandler meinHandler = new MeinHandler();
```

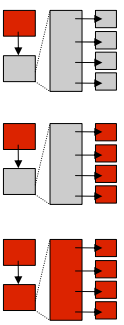
- **Parser erzeugen,** mit Datenquelle und Handler **aufrufen**:

```
try {
    SAXParser parser = factory.newSAXParser();
    parser.parse(new File("daten.xml"), meinHandler);
}
catch (Exception e) {
  e.printStackTrace();
}
```


MiniSAX.java

```
import java.io.File;
import javax.xml.parsers.*;
import org.xml.sax.helpers.*;
import org.xml.sax.*;
import MeinHandler.*;

public class MiniSAX {
  public static void main(String[ ] args) {
    SAXParserFactory factory=SAXParserFactory.newInstance();
    factory.setNamespaceAware(true);

    DefaultHandler meinHandler = new MeinHandler();

    try {
      SAXParser parser = factory.newSAXParser();
      parser.parse(new File("daten.xml"), meinHandler);
    } catch (Exception e) {  e.printStackTrace(); }
  }
}
```

Imports für SAX-Verwendung

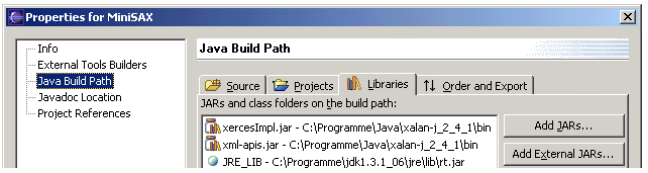


Einbindung von Xerces2-J

- Xerces2-J ist eine kostenlose Java-SAX-Implementierung aus dem Apache-XML-Projekt
  - Homepage: http://xml.apache.org

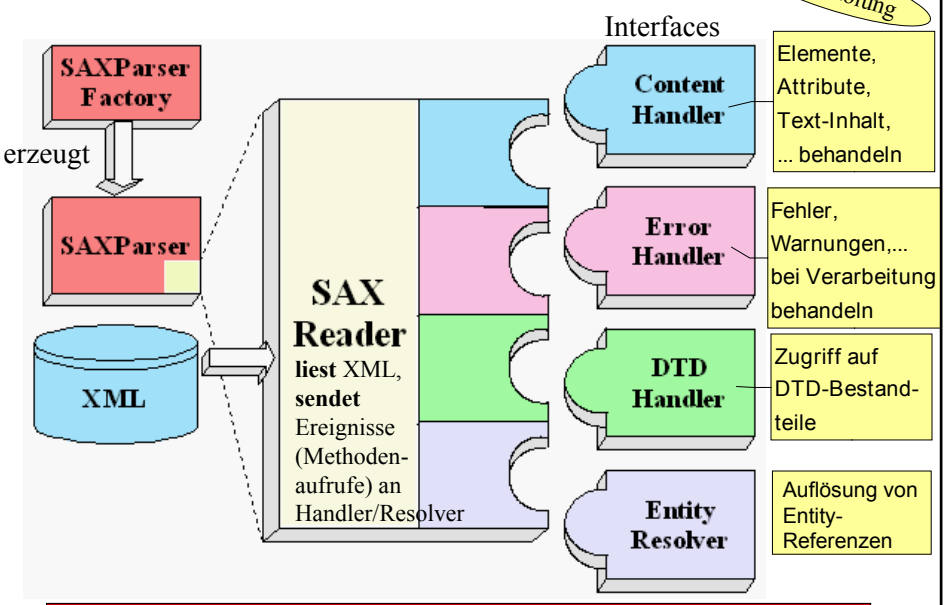
- Dokumentation der Schnittstellen (APIs): http://xml.apache.org/xerces2-j/javadocs/api/index.html

- Folgende JAR-Dateien müssen eingebunden werden: xercesImpl.jar, xmlParserAPIs.jar

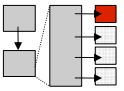


SAX: Bestandteile

Wiederholung

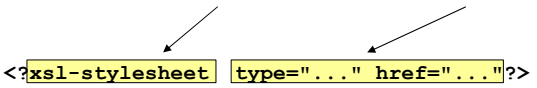


Mehr zu ContentHandler-Events



- `void startDocument()`
- `void endDocument()`
- `void startElement(String namespaceURI, String localName, String qualifiedName, Attributes attrs)`
  - `attrs.getLength()`
  - `attrs.getLocalName(int i), attrs.getQName(int i),`
  - `attrs.getValue(int i), attrs.getValue(String qName)`
- `void endElement(String namespaceURI, String localName, String qName)`
- `void characters(char[ ] ch, int start, int length)`

  Tipp: Konvertierung in String mit
  `"String s = new String(ch, start, length);"`
- `void processingInstruction(String target, String data)`



SAX: Online-Demo

- LongestSpeech.java
  - Das XSLT-Beispiel zur Ermittlung der längsten Rede im Shakespeare-Drama “The Tempest”, nun mittels SAX
  - Verallgemeinert: Längste Rede pro Redner
- Zum Vergleich
  - Ruby-Lösung: Kein Fehlerbehandlungscode, Ausgabe schon sortiert
- Testdaten stehen bereit als (Pseudo-) Übung 12
  - Aufrufe:

```
$ java -classpath /path/to/xercesImpl.jar
       LongestSpeech 10-tempest.xml
$ ruby longest_speech.rb 10-tempest.xml
```

    (falls Ruby installiert ist)
- Ergebnisse
  - Java: Schnell, ca. 140 Zeilen Sourcecode
  - Ruby: Langsamer, ca. 20 Zeilen Sourcecode, Ergebnis sortiert

Vorteile/Nachteile von SAX

- **Vorteile:**
  - SAX-Parser sind **klein** und **schnell**
  - Geringer **Speicherverbrauch**
  - **Flexibilität** bei Fehlerbehandlung
  - SAX-Eventerzeugung ist **einfach**
- **Nachteile:**
  - Reine „von-oben-nach-unten-Verarbeitung“, **kein freier Zugriff** auf XML-Struktur

DOM

Document **O**bject **M**odel

http://www.w3.org/DOM

http://www.w3.org/DOM/DOMTR

DOM

- **Herkunft**
  - DOM entstand aus dem Bedarf der HTML-Browser nach einem API zur dynamischen Veränderung des dargestellten Dokuments (DHTML)
  - Proprietäre Vorläufer, dann Standardisierung durch W3C
    - DOM Level 1 01. 10. 1998
    - DOM Level 2 Core 13. 11. 2000
    - DOM Level 3 Core 07. 04. 2004
- **Ziele & Eigenschaften**
  - Sprach- und herstellerunabhängiges, objekt-orientiertes API für HTML & XML. Primär für JavaScript entstanden.
  - Wahlfreier Zugriff auf alle Bestandteile eines Dokuments
  - Auch zum Verändern und Erzeugen von Dokumenten

DOM in JavaScript: Online-Demo

- **dom_demo1.html**
  - Enthält JavaScript-Code, der Daten aus XML-Datei “unicode.xml” einliest, daraus eine HTML-Tabelle baut und diese in die aktuelle Seite dynamisch einfügt.
  - Quelle der Vorlage: http://www.quirksmode.org, importXML.html
  - createTable() ist das zentrale “Arbeitspferd” (s.u.) und enthält zahlreiche DOM-Methoden (blau markiert).
- **Bemerkungen zur Demo:**
  - Nur ein erster Eindruck, kein Versuch einer systematischen Einführung in DOM
  - Methodennamen sind selbsterklärend, wenn man an XML-Datenmodelle und XPath gewöhnt ist.

DOM: Online-Demo / Quellcode-Ausschnitt

```
function createTable()
{
   var x = xmlDoc.getElementsByTagName('Eintrag');
   var newEl = document.createElement('TABLE');
   newEl.setAttribute('cellPadding',5);
   var tmp = document.createElement('TBODY');
   newEl.appendChild(tmp);
   var row = document.createElement('TR');
   for (j=0;j<x[0].childNodes.length;j++)
   {
       if (x[0].childNodes[j].nodeType != 1) continue;
       var container = document.createElement('TH');
       var theData = document.createTextNode
                          (x[0].childNodes[j].nodeName);
       container.appendChild(theData);
       row.appendChild(container);
   }
```


DOM: Online-Demo / Quellcode-Ausschnitt (2)

```
   tmp.appendChild(row);      // Fortsetzung createTable()
   for (i=0;i<x.length;i++)
   {
       var row = document.createElement('TR');
       for (j=0;j<x[i].childNodes.length;j++)
       {
         if (x[i].childNodes[j].nodeType != 1) continue;
         var container = document.createElement('TD');
         var theData = document.createTextNode
               (x[i].childNodes[j].firstChild.nodeValue);
         container.appendChild(theData);
         row.appendChild(container);
       }
       tmp.appendChild(row);
   }
   document.getElementById('writeroot').appendChild(newEl);
} // Ende createTable()

// Bem.: Alle blau gefärbten Methoden sind DOM-Methoden.
```

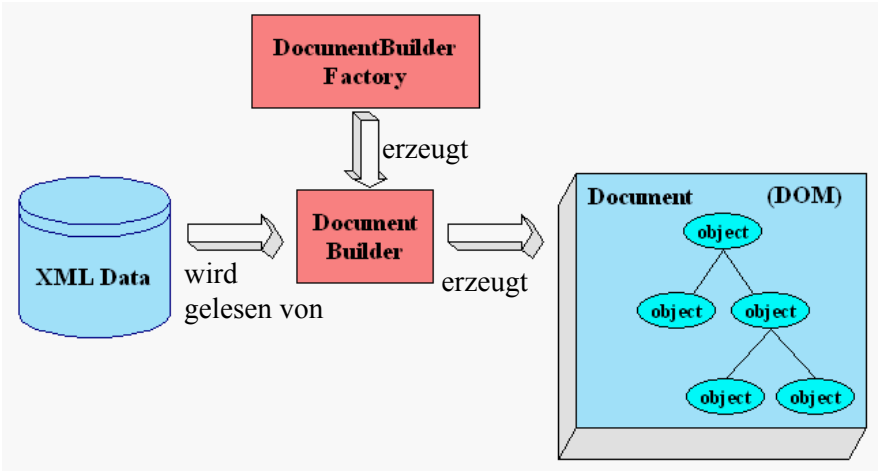
Vorteile/Nachteile von DOM

- **Vorteil:**
  - **Freier Zugriff auf Baumstruktur**
  - **insbesondere auch zur Veränderung der Struktur**

- **Nachteil:**
  - Ressourcenverbrauch (Rechenzeit, Speicher...)
  - Beispiel: EDI, *salesreport*-Daten von 200 Kaufhof-Filialen zu ca. 150 Artikeln eines Sortiments
    - Übertragen: ca. 0.15 MB (UN/EDIFACT, bzip2)
    - Ausgepackt: ca. 5 MB (UN/EDIFACT)
    - Als XML-Dok.: ca. 75 MB
    - Im Speicher, mit DOM: ca. 750 MB (!)
    - Mit gleich großem Zieldokument schließlich für nur einen (!) Mapping-Prozess: ca. 1500 MB Hauptspeicher

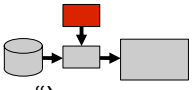
Ergänzung: DOM in Java

DOM in Java: Übersicht

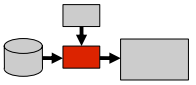


Wichtige DOM-Bausteine

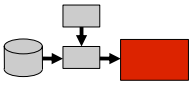
- ***DocumentBuilderFactory***
  - erzeugt DocumentBuilder,
  - abhängig von Einstellungen (Property „javax.xml.parsers.SAXParserFactory“)



- ***DocumentBuilder***
  - stellt u.a. parse()-Methode bereit,
  - erzeugt Document-Objekte



- ***Document***
  - repräsentiert ein ganzes (XML-)Dokument
  - entspricht dem Wurzelknoten „/“



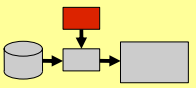
Nützliche Imports für DOM

- import **javax.xml.parsers.*;**
  - für DocumentBuilder (-Factory), ...
- import **org.xml.sax.*;**
  - für SAXException, SAXParseException, ...
- import **org.w3c.dom.*;**
  - für Document, DOMException, ...

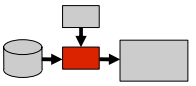
Vorgehen bei DOM-Nutzung

- **DocumentBuilderFactory** erzeugen, Namespace-Support einschalten:

```
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
```



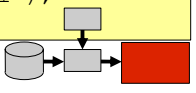
- **DocumentBuilder** erzeugen, ggf. Fehler-Handler (Implementierung des Interface „ErrorHandler“) hinzufügen:



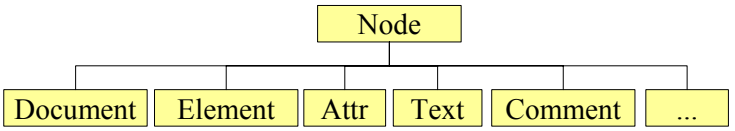
```
DocumentBuilder docBuilder=factory.newDocumentBuilder();
docBuilder.setErrorHandler(new MyErrorHandler());
```

- **Dokument erzeugen (aus XML-Eingabestrom oder „neu“):**

```
Document doc1 = docBuilder.parse("file:datei.xml");
Document doc2 = docBuilder.newDocument();
```



Rund um „Node“ (Auswahl)

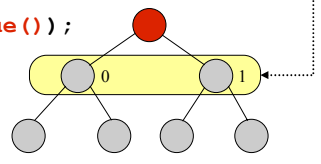


- **Navigation im Baum:**
  - `getParentNode()`, `getFirstChild()`, `getLastChild()`
  - `getNextSibling()`, `getPreviousSibling()`
- **Werte auslesen und setzen:**
  - `getNodeName()`, `getNodeType()`, `getAttributes()`
  - `getNodeValue()`, `setNodeValue()`
- **Baum verändern:**
  - `removeChild()`, `replaceChild()`
  - `appendChild()`, `insertBefore()`

http://xml.apache.org/xerces2-j/javadocs/api/index.html

Umgang mit Kindern

```
// Rekursiv alle Textknoten unter "node" ausgeben
void printTexts(Node node) {
  if (node.hasChildNodes()) {
    NodeList lst = node.getChildNodes();
    for (int i=0; i < lst.getLength(); i++) {
      printTexts(lst.item(i));
    }
  } else if (node.getNodeType()==Node.TEXT_NODE){
    System.out.print(node.getNodeValue());
  }
}
```



Zugriff auf Attribute

```
void printNode(Node node) {
  // alle Attribute des Knotens 'node' ausgeben
  if (node.hasAttributes()) {
    NamedNodeMap attMap = node.getAttributes();
    for (int i=0; i < attMap.getLength(); i++) {
      Node attr = attMap.item(i);
      System.out.println(attr.getNodeName()
        + "=" + attr.getNodeValue());
    }
  }
}
```

```
...
if (node.getType() == Node.ELEMENT_TYPE) {
  Element ele = (Element) node;
  System.out.println(ele.getAttribute("testattr"));
}
...
```


Bäume von Hand bauen

```
Document doc = docBuilder.newDocument();
Element wurz = doc.createElement("wurzel");
Element sohn = doc.createElement("sohn");
Element tochter = doc.createElement("tochter");
Text gruss = doc.createTextNode("Ein Textbeispiel");
sohn.appendChild(doc.createTextNode("Sohn hier"));
tochter.appendChild(
  doc.createTextNode("Tochter da"));
doc.appendChild(wurz);
wurz.appendChild(sohn);
wurz.appendChild(gruss);
wurz.appendChild(tochter);
```