5.3. Document Object Model - DOM

- Überblick DOM
- DOM und JAXP
- Node-Interface
- Einige Subinterfaces von Node
- Weitere Interfaces

Überlick DOM

- DOM-Entwicklung
- DOM-Baumstruktur
- Knoten-Eigenschaften
- DOM Interfaces

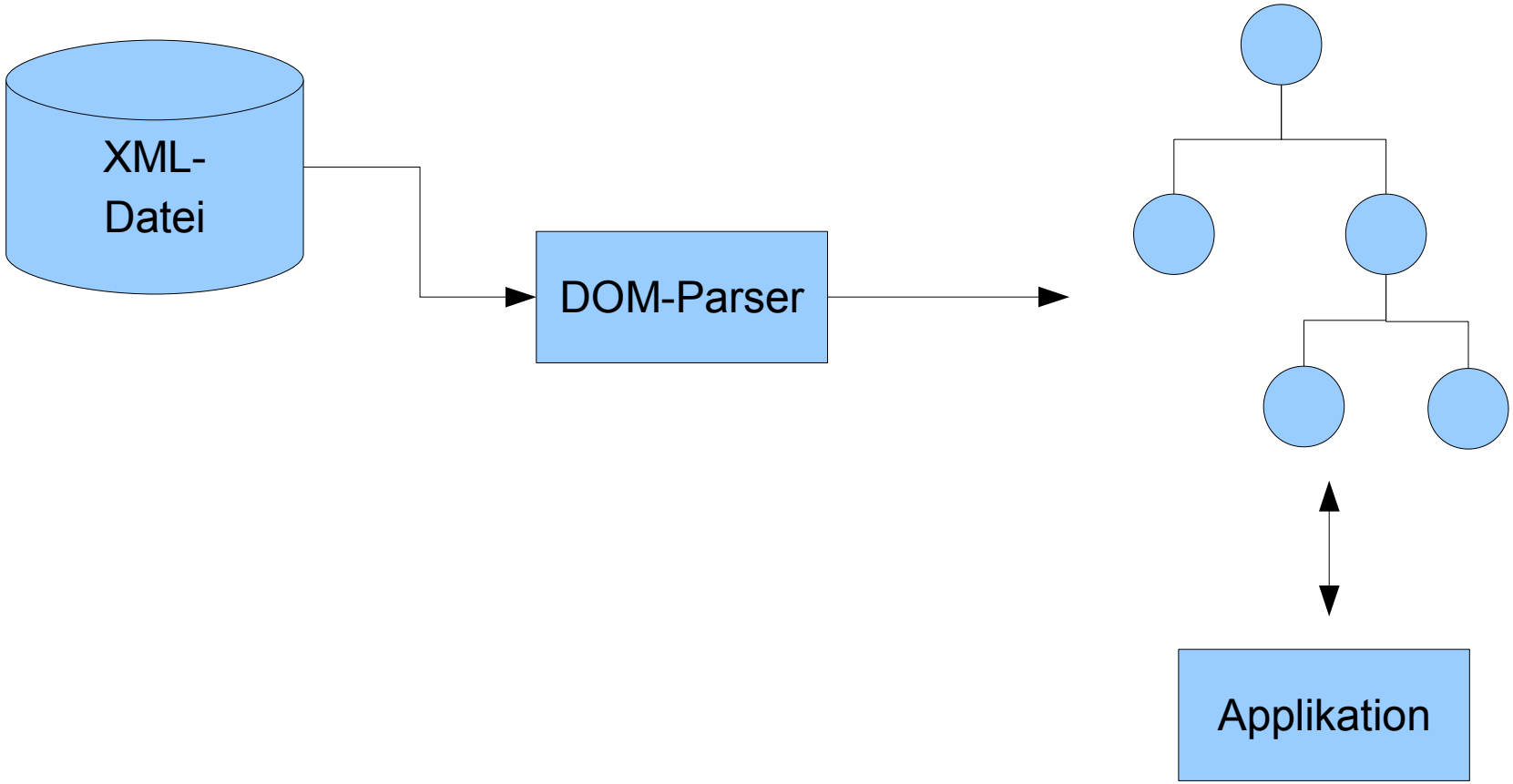
DOM-Entwicklung

- DOM: Document Object Model
- W3C-Recommendation: http://www.w3.org/DOM/
- DOM-Versionen in Form von "Levels":
  - Level 0 (keine Recommendation): nur HTML, für JavaScript in Browsern
  - Level 1: Unterstützung von XML 1.0 und HTML 4.0
  - Level 2: Namespaces im Element/Attribute Interface, erweiterte Baum-Manipulationsmöglichkeiten, etc.
  - Level 3: Laden/Speichern, XPath, etc.
- Programmiersprachen-unabhängig
- DOM definiert:
  - logische Struktur eines XML-Dokuments (als Baum)
  - Methoden zum Navigieren und zum Manipulieren des Baumes

DOM-Baumstruktur

- XML Dokument wird als Baumstruktur dargestellt
  - Dieser Baum ist im allgemeinen detaillierter als im Xpath/XSLT Datenmodell (z.B.: eigene Knoten für Entity, CDATA-Section, etc.)
  - Knoten des Baums sind vom Typ "Node"
- Die verschiedenen Knoten-Arten erben von "Node":
  - Document : hier ist der ganze DOM-Baum "aufgehängt"
  - Element, Attr, Text, ProcessingInstruction, Comment
  - DocumentFragment, DocumentType, Entity, CDATASection, EntityReference, Notation
- Wichtige Knoten-Eigenschaften:
  - nodeName, nodeValue, nodeType, attributes

Arbeitsweise von DOM-Parsern



Beispiel für einen DOM-Baum

```
<sentence>
  The &projectName; <![CDATA[<i>project</i>]]> is
  <?editor: red?><bold>important</bold><?editor: normal?>.
</sentence>
```

Dazugehöriger DOM-Baum:

+ Element: sentence
  + Text: The
  + EntityReference: projectName
    + Text: Eagle
  + CDATASection: project
  + Text: is
  + ProcessingInstruction: editor: red
  + Element: bold
    + Text: important
  + ProcessingInstruction: editor: normal
  + Text: .

Hierarchie der DOM-Objekte (1)

- Erlaubte Kinder eines Document Objekts
  - DocumentType (max. 1)
  - Element (max. 1)
  - Processing Instruction
  - Comment
- Erlaubte Kinder eines Element Objekts (gilt auch für DocumentFragment, Entity oder EntityReference):
  - Element
  - Processing Instruction
  - Comment
  - Text
  - CDATASection
  - EntityReference

Hierarchie der DOM-Objekte (2)

- Erlaubte Kinder eines Attr Objekts
  - Text
  - EntityReference
- Objekte ohne weitere Kinder
  - DocumentType
  - Processing Instruction
  - Comment
  - Text
  - CDATASection
  - Notation

Knoten-Eigenschaften

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	`"#cdata-section"`	content of the CDATA Section	null
Comment	`"#comment"`	content of the comment	null
Document	`"#document"`	null	null
DocumentFragment	`"#document-fragment"`	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	`"#text"`	content of the text node	null

DOM-Interfaces

- Zentrales Interface: **`Node`**
  - Gemeinsame Methoden der Knoten des DOM-Baums
  - Navigieren im Baum, Manipulieren des Baums (Knoten löschen/ einfügen/ändern), Knoten-Eigenschaften (lesen,schreiben), etc.
- Subinterfaces von Node für jeden Knotentyp
  - Stellen spezifische Methoden zur Verfügung
- Weitere Interfaces:
  - **`NamedNodeMap`**: Sammlung von Knoten, auf die mittels Name oder Index zugegriffen werden kann.
  - **`NodeList`**: Sammlung von Knoten, auf die mittels Index zugegriffen werden kann
  - **`DOMImplementation`**: Erlaubt das Auslesen und Setzen von "Features" der DOM-Implementierung.

DOM und JAXP

- Das Java API hält sich streng an die W3C Vorgabe
- Der **`DocumentBuilder`** liest und erzeugt den DOM-Baum
- Der **`DocumentBuilder`** wird via Factory erzeugt
  - Factories ermöglichen es den Parser transparent zu tauschen
- Features werden über die Factory gesetzt (bevor der **`DocumentBuilder`** erzeugt wird).

Java Mindest-Code (1)

- Importe: DOMBuilder/Factory, DOM:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

- Factory-Instanzierung:

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

- Parser-Instanzierung und Parsen:

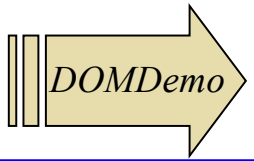
```
DocumentBuilder builder =
  factory.newDocumentBuilder();
Document document = builder.parse(new
  File(filename));
```

Java Mindest-Code (2)

- Einstellungen des Parsers:

```
Document document;
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
factory.setIgnoringComments(true);
factory.setIgnoringElementContentWhitespace(true);


DocumentBuilder builder =
  factory.newDocumentBuilder();
Document document = builder.parse(new
  File(filename));
```



Einige weitere Anweisungen

- Neuen DOM-Baum erzeugen:

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder =
  factory.newDocumentBuilder();

Document document = builder.newDocument();
```

- Neuen Knoten erzeugen und einfügen:

```
Element root = document.createElement("test");
document.appendChild(root);
root.appendChild(document.createTextNode("Some
  Text"));
```

Node-Interface

- Node Properties
- Navigation im Baum
- Manipulation des Baums
- Utilities

Node Properties

```
public String   getNodeName();
public String   getNodeValue() throws DOMException;
public void     setNodeValue(String nodeValue)
                  throws DOMException;
public short    getNodeType();
public String   getNamespaceURI();
public String   getPrefix();
public void     setPrefix(String prefix)
                  throws DOMException;
public String   getLocalName();
```

Navigation im Baum

- `public Node getParentNode();`
- `public boolean hasChildNodes();`
- `public NodeList getChildNodes();`
- `public Node getFirstChild();`
- `public Node getLastChild();`
- `public Node getPreviousSibling();`
- `public Node getNextSibling();`
- `public Document getOwnerDocument();`
- `public boolean hasAttributes();`
- `public NamedNodeMap getAttributes();`

Manipulation des Baums

- `public Node insertBefore(Node newChild, Node refChild) throws DOMException;`
- `public Node replaceChild(Node newChild, Node oldChild) throws DOMException;`
- `public Node removeChild(Node oldChild) throws DOMException;`
- `public Node appendChild(Node newChild) throws DOMException;`

- Beispiele:
  - Die Methode `removeChild` entfernt den Knoten `oldChild` aus dem DOM-Baum und liefert diesen Knoten als Ergebnis zurück.
  - `replaceChild` ersetzt den Knoten `oldChild` durch `newChild` im DOM-Baum und liefert `oldChild` als Ergebnis zurück.

Utilities

- `boolean isEqualNode(Node arg);`
- `boolean isSameNode(Node other);`
- `public Node cloneNode(boolean deep);`

```
// Bei deep = true wird der gesamte Subbaum kopiert.
// Ansonsten nur der einzelne Knoten.
```

- `public void normalize();`

```
// eliminiert leere Text-Knoten und verschmilzt
// benachbarte Text-Knoten im ganzen Subbaum
```

- `public String getTextContent() throws DOMException;`

```
// liefert bei Element-Knoten den gesamten Text-Inhalt
// im Unterbaum. Diesen müsste man sonst aus allen
Text-,
// CDATASection- und EntityReference-Knoten
zusammensuchen.
```

Einige Subinterfaces von Node

- Überblick
- Document Interface
- Element Interface
- Attr Interface
- CharacterData, Text, Comment, CDATASection

Überblick

- Alle Subinterfaces von Node:

Attr	**Element**
CDATASection	**Entity**
CharacterData	**EntityReference**
Comment	**Notation**
Document	**ProcessingInstruction,**
DocumentFragment	**Text**
DocumentType	

Document Interface (1)

- Der DOM-Baum ist an einem Document-Knoten aufgehängt.
- Jeder DOM-Knoten ist einem Document Knoten zugeordnet.
- Das Document Interface bietet Methoden, um neue Knoten (für diesen DOM-Baum) zu erzeugen oder um Knoten aus einem anderen Baum zu importieren.

```
Node adoptNode(Node source) throws DOMException
   // gibt dem Knoten "source" ein neues "ownerDocumnet"
   // und entfernt ihn von der Kinder-Liste seines Parent.
```

```
Node importNode(Node importedNode, boolean deep)
        throws DOMException
   // erzeugt eine Kopie des importierten Knoten sowie
   // (bei deep = true) des gesamten Subbaums.
```

Document Interface (2)

- `Attr createAttribute(String name) throws DOMException`
- `Element createElement(String tagName) throws DOMException`
- `Text createTextNode(String data) ...`

- `DocumentType getDoctype()`
  `// direkter Zugriff zum DocumentType Kind-Knoten`
- `Element getDocumentElement()`
  `// direkter Zugriff zum einzigen Element Kind-Knoten`
- `Element getElementById(String elementId)`
  `// sucht nach dem Element mit diesem ID-Attribut`
- `NodeList getElementsByTagName(String tagname);`
  `// liefert alle Elemente mit diesem Namen im Dokument`

Element Interface (1)

- Einige wesentliche Funktionen sind nicht Element-spezifisch sondern werden vom Node Interface geerbt, z.B.: getAttributes(), getName(), Navigation im Baum
- Das Element Interface bietet einige nützliche Zusatz-Funktionen.

```
NodeList getElementsByTagName(String name)
    // liefert alle Element mit diesem Namen im Subbaum
boolean hasAttribute(String name)
    // hat das Element ein Attribut mit diesem Namen?
```

Element Interface (2)

- `String getAttribute(String name)`
  `// liefert den Wert des Attributes "name"`
- `void setAttribute(String name, String value) throws DOMException`
  `// erzeugt oder überschreibt Attribut mit diesem Wert.`
- `void removeAttribute(String name) throws DOMException`
  `// löscht das Attribut mit diesem Wert.`

- `Attr getAttributeNode(String name)`
  `// liefert den Knoten des Attributes "name"`
- `Attr setAttributeNode(Attr newAttr) throws DOMException`
  `// fügt ein neues Attribut ein (oder ersetzt ein altes mit`
  `// dem selben Namen.`
- `Attr removeAttributeNode(Attr oldAttr) throws DOMException`

Attr Interface

- Attribute gelten nicht als Kinder eines Elements.
- Sie müssen entweder mit getAttributes() oder (falls der Name bekannt ist) mit getAttribute() geholt werden.
- Das Attr Interface bietet nur wenige Erweiterungen gegenüber dem Node Interface, z.B.:

```
Element getOwnerElement()
    // liefert zugehörigen Element Knoten
boolean isId()
    // hat das Attribut den Typ ID?
getValue(), setValue(String value), getName():
analog zu getNodeValue, setNodeValue, getNodeName
```

CharacterData Interface

- Bildet eine "Zwischenebene" zwischen Node Interface und den Interfaces **`CDATASection, Comment, Text`**
- Bietet die "typischen" String-Manipulationen, z.B.:

```
String getData()throws DOMException
   // liefert die Text-Daten des Knotens
void setData(String data) throws DOMException
int getLength()
String substringData(int offset, int count)throws DOMException
void appendData(String arg) throws DOMException
void insertData(int offset, String arg) throws DOMException
ähnlich: deleteData, replaceData (mit offset+count)
```

Text, Comment, CDATASection

- Comment erbt von CharacterData ohne Erweiterungen
- CDATASection erbt von Text ohne Erweiterungen
- Text erbt von CharacterData und bietet einige wenige Erweiterungen, z.B.:
- `String getWholeText()`

```
// liefert Text von diesem Knoten plus von allen
// benachbarten Text-Knoten
```

- `boolean isElementContentWhitespace()`

```
// d.h. whitespace wo Element-Inhalt stehen müsste,
// häufig als "ignorable whitespace" bezeichnet
```

Weitere Interfaces

- NodeList
- NamedNodeMap

NodeList

- Praktisch zum Abarbeiten von Knotenmengen (z.B. alle Kinder eines Knoten, alle Elemente mit einem bestimmten Namen,..)
- Lesender Zugriff
- hat nur 2 Methoden:

```
int getLength()
    // liefert Anzahl der Knoten in der NodeList
Node item(int index)
    // wahlfreier Zugriff mittels index
    // (der bei 0 beginnt)
```

Beispiel

- Suche nach dem ersten Subelement mit einem betimmten Namen:

```
public Node findSubNode(String name, Node node) {
   if (! node.hasChildNodes()) return null;
   NodeList list = node.getChildNodes();
   for (int i=0; i < list.getLength(); i++) {
      Node subnode = list.item(i);
      if (subnode.getNodeType() == Node.ELEMENT_NODE) {
         if (subnode.getNodeName().equals(name)) return subnode;
               else {
                     Node tmp = findSubNode(name, subnode);
                     if(tmp!=null) return tmp;}}
   }  }  }
   return null;
}
```

Beispiel (Alternative)

- Suche nach dem ersten Subelement mit einem betimmten Namen:

```
public Node findSubNode(String name, Node node) {
        NodeList nl;
        if(node.getNodeType()==Node.ELEMENT_NODE) {
                nl = ((Element)node)).getElementsByTagName(name);
        } else if (node.getNodeType()==Node.DOCUMENT_NODE) {
                nl = ((Document)node).getElementsByTagName(name);
        }
        if(nl!=null && nl.getLength() > 0) // nicht unb. nötig
                return nl.item(0);
        return null;
}
```

Beispiel (Alternative 2)

- Suche nach dem ersten Subelement mit einem bestimmten Namen:

```
public Node findSubNode(String name, Node node) {
        return null;
}
public Node findSubNode(String name, Element n) {
        NodeList nl=n.getElementsByTagName(name);
        return nl.item(0);
}
public Node findSubNode(String name, Document n) {
        NodeList nl=n.getElementsByTagName(name);
        return nl.item(0);
}
```

NamedNodeMap (1)

- Zugriff auf Knoten in einer NamedNodeMap:
  - Entweder mittels Name oder mittels Index. Im Gegensatz zu NodeList haben die Knoten einer NamedNodeMap keine definierte Reihenfolge
- Schreibender oder lesender Zugriff
- Wird vorwiegend für Attribute verwendet
- `int getLength()`

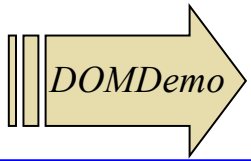
  `// liefert Anzahl der Knoten in der NamedNodeMap`
- `Node item(int index)`

  `// wahlfreier Zugriff mittels index`

  `// (der bei 0 beginnt)`

NamedNodeMap (2)

- `Node getNamedItem(String name)`
  `// liefert Knoten mit diesem Namen`
- `Node setNamedItem(Node arg) throws DOMException`
  `// fügt einen neuen Knoten ein (oder ersetzt einen alten`
  `// mit dem selben Namen.`
- `Node removeNamedItem(String name) throws DOMException`
  `// entfernt den Knoten mit diesem Namen und liefert`
  `// den Knoten zurück.`



dbai

5.4. XML-Ausgabe

- Überblick
- Ausgabe mittels Transformer
  - SAX
  - DOM
- Ausgabe mittels LSSerializer
  - DOM

Überblick

- **XML-Dokument ausgeben/schreiben:**
- mittels Transformer (SAX, DOM)

  Eigentlich für XSLT-Transformationen gedacht
  Erlaubter Input: StreamSource, DOMSource oder SAXSource
  Erlaubter Output: StreamResult, DOMResult oder SAXResult
  Ohne Angabe eines XSLT-Stylesheets wird ein XML-Dokument einfach von einem der Input-Formate in eines der Output-Formate umgewandelt.
- mittels LSSerializer (DOM)

  In DOM Level 3 wurde "Load and Save" Modul ergänzt
  Eigenes Package: org.w3c.dom.ls
  Interface LSSerializer: zur "Umwandlung" in ein XML-Dokument.

SAX-Ausgabe mittels "Transformer" (1)

- Output des XMLReaders als XML-Dokument
  - mittels "transformer", analog zur DOM-Ausgabe, d.h.:
    - Erzeuge mittels XMLReader eine SAXSource
    - Aufruf des transformers mit dieser SAXSource als Input

- Importe
  - wie bei DOM:

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
```

  - SAX-spezifische Importe:
    - `import javax.xml.transform.sax.SAXSource;`
    - `import org.xml.sax.InputSource;`

SAX-Ausgabe mittels "Transformer" (2)

- Transformer-Instanzierung (wie bei DOM):

```
TransformerFactory tFactory =
 TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer();
```

- Ausgabe (d.h.: erzeuge SAXSource mittels XMLReader)

```
SAXSource source =
 new SAXSource(reader, new InputSource(quelle));
StreamResult result = new StreamResult(new File(ziel));
transformer.transform(source, result);
```

DOM-Ausgabe mittels "Transformer"

- Importe:

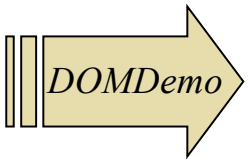
```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

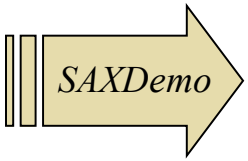
- Transformer instanzieren:

```
TransformerFactory tFactory =
   TransformerFactory.newInstance();
Transformer transformer =
   tFactory.newTransformer();
```

DOM-Ausgabe mittels "Transformer"

- Ausgabe:
  - ```
    DOMSource source = new DOMSource(document);
    StreamResult result =
       new StreamResult(new File("output.xml"));
    transformer.transform(source, result);
    ```




    ```

DOM-Ausgabe mittels "LSSerializer"

- Voraussetzung:
  - Die verwendete DOM-Implementierung muss DOM 3.0 unterstützen.
  - In diesem Fall implementiert die DOMImplementation das erweiterte Interface DOMImplementationLS.
  - DOMImplementationLS bietet eine Factory zum Erzeugen von einem LSSerializer.

- Importe:

```
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSSerializer;
```

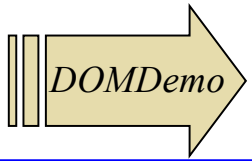
DOM-Ausgabe mittels "LSSerializer"

- LSSerializer instanzieren:

```
DOMImplementation di =
   document.getImplementation();
if (di.hasFeature("Core","3.0")){
  DOMImplementationLS diLS =
   (DOMImplementationLS) di;
  LSSerializer lss = diLS.createLSSerializer();
  …
};
```

- Ausgabe:

```
FileWriter fw = new FileWriter("output.xml");
fw.write(lss.writeToString(document));
fw.flush();
fw.close();
```



5.5. Epilog

- DOM vs. SAX
- Literatur

DOM vs. SAX

DOM:

- Baut gesamten XML-Baum im Speicher auf => wahlfreier Zugriff
- Manipulation des Baums möglich
- Hoher Speicherbedarf, langsamer

SAX:

- XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
- "streaming" möglich (d.h.: Bearbeiten und Weiterreichen, bevor das ganze Dokument übertragen ist).
- Geringerer Speicherbedarf, höhere Geschwindigkeit
- Falls mehrmaliger Zugriff auf Knoten erforderlich: Applikation ist selbst für das Puffern verantwortlich.
- Low level (aktuelle DOM-API benutzt SAX-API)

Literatur

- Spezifikationen:
  - http://www.w3.org/DOM/
  - http://www.saxproject.org/
- (Online) Bücher und Artikel:
  - Elliotte Rusty Harold: "Processing XML with Java"
    - http://www.cafeconleche.org/books/xmljava/
  - J2EE Tutorial (Kap. 4-7):
    - http://java.sun.com/j2ee/1.4/docs/tutorial/doc/