



# Semistrukturierte Daten

## Sommersemester 2008

### Teil 5: DOM

- 5.1. Überblick
- 5.2. Überblick DOM
- 5.3. Node-Interface
- 5.4. Einige Subinterfaces von Node
- 5.5. weitere Interfaces
- 5.6. Benutzung von DOM in JAXP
- 5.7. DOM-Ausgabe



## 5.1. Überblick

---

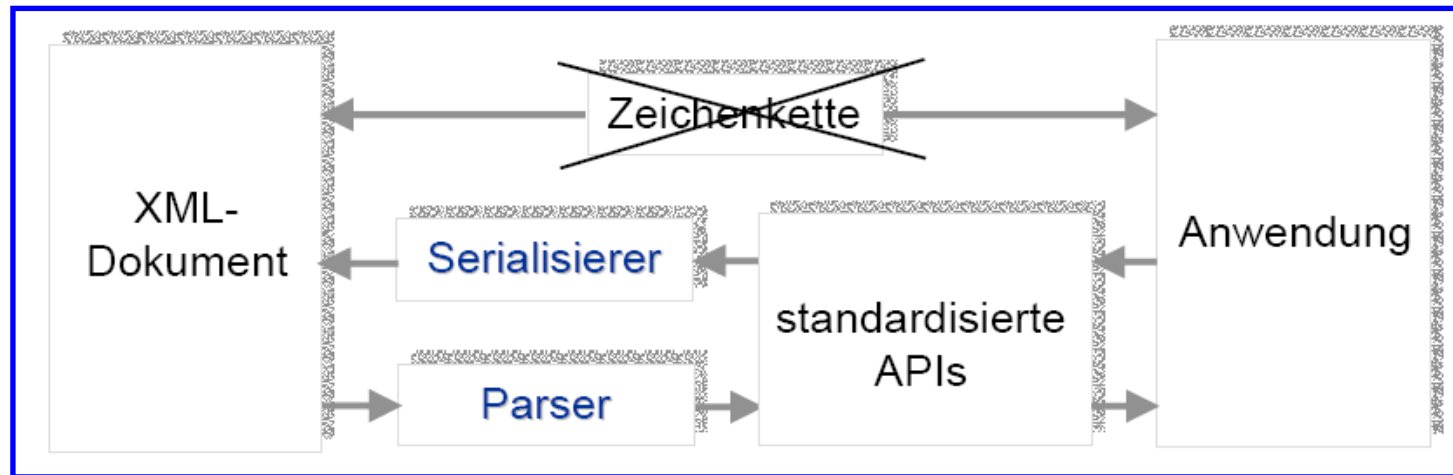
- XML-APIs und Java
- XML-Prozessor (Parser)
- Parser-Modelle

# XML-APIs und Java

- Die wichtigsten XML-APIs (DOM und SAX) sind eigentlich **programmiersprachen-unabhängig**. In der SSD-Vorlesung werden aber nur die **Java-Realisierungen** behandelt.
- JAXP (Java API for XML-Processing):
  - Teil von JDK ab Version 1.4
  - Enthält Java-Realisierung von DOM und SAX sowie ein XSLT-API (TrAX: Transformation API for XML)
- JAXP-Packages:
  - javax.xml.parsers (gemeinsames Interface für SAX und DOM Parser von unterschiedlichen Herstellern).
  - org.w3c.dom
  - org.xml.sax
  - javax.xml.transform

# Idee eines XML-Prozessors (Parsers)

- Stellt der Applikation eine **einfache Schnittstelle** zum **Zugriff** auf ein XML-Dokument (plus eventuell zum **Manipulieren** und wieder **Abspeichern** des XML-Dokuments) zur Verfügung.
- Ermöglicht die Konzentration auf die **logische Struktur bzw. den Inhalt** des XML-Dokuments (und nicht auf die exakte Syntax) => wesentlich flexibler/robuster als Text-Parsen.



# Aufgaben eines XML-Prozessors

## Parser:

- Überprüfung der Wohlgeformtheit und (optional) der Gültigkeit
- Unterstützung unterschiedlicher Encodierungen
- Ergänzung von default/fixed-Werten
- Auflösen von internen/externen Entities und Character References
- Normalisierung von whitespace
- Generiert aus dem XML-Dokument eine interne Datenstruktur

## Serialisierer:

- Generiert aus der internen Datenstruktur ein XML-Dokument
- Kümmert sich dabei um Wohlgeformtheit des XML-Dokuments

# Parser-Modelle

## ■ Objektmodell Parser:

- Baut gesamten XML-Baum im Speicher auf => wahlfreier Zugriff
- Beispiele: DOM, JDOM

## ■ Push Parser (ereignisorientiert, Parser kontrolliert Ablauf):

- XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
- Applikation kann für die möglichen "Events" (d.h. syntaktische Einheiten) callback Funktionen bereitstellen
- Beispiel: SAX

## ■ Pull Parser (ereignisorientiert, Applikation kontrolliert Ablauf):

- XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
- Applikation fordert Analyse der nächsten syntaktischen Einheit an
- Beispiel: StAX (mittlerweile Teil von Java als SJSXP, "Sun Java Streaming XML Parser")

## 5.2. Überblick 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.
- Plattform- und programmiersprachen-unabhängig;
  - Implementierung muss nicht einmal in OO-Sprache erfolgen.
- 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

# Beispiel für einen DOM-Baum

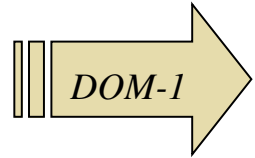
<sentence>

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: <i>project</i>
  - + 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 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 weitere 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 Java-Interface Hierarchie

interface org.w3c.dom.Node

- \* interface org.w3c.dom.Attr
- \* interface org.w3c.dom.CharacterData
  - o interface org.w3c.dom.Comment
  - o interface org.w3c.dom.Text
    - + interface org.w3c.dom.CDATASection
- \* interface org.w3c.dom.Document
- \* interface org.w3c.dom.Element
- ... ..
- \* interface org.w3c.dom.ProcessingInstruction

interface org.w3c.dom.NamedNodeMap

interface org.w3c.dom.NodeList

interface org.w3c.dom.DOMImplementation

## 5.3. Node-Interface

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

# Node Types

## Konstanten-Definitionen im Node Interface:

```
public static final short ELEMENT_NODE = 1;
public static final short ATTRIBUTE_NODE = 2;
public static final short TEXT_NODE = 3;
public static final short CDATA_SECTION_NODE = 4;

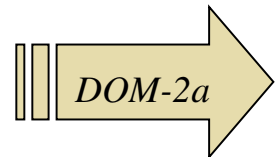
ENTITY_REFERENCE_NODE = 5; ENTITY_NODE = 6;
PROCESSING_INSTRUCTION_NODE = 7; COMMENT_NODE = 8;
DOCUMENT_NODE = 9; DOCUMENT_TYPE_NODE = 10;
DOCUMENT_FRAGMENT_NODE = 11; NOTATION_NODE = 12;
```

# 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.
```

## 5.4. Einige Subinterfaces von Node

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

# Überblick

## Alle Subinterfaces von Node:

**Attr**

**CDATASection**

**CharacterData**

**Comment**

**Document**

**DocumentFragment**

**DocumentType**

**Element**

**Entity**

**EntityReference**

**Notation**

**ProcessingInstruction,**

**Text**

# 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 "ownerDocument" 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)

etc.

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)

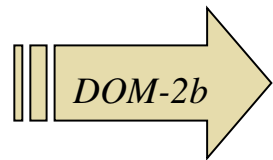
```
String getAttribute(String name)
    // liefert den Wert des Attributes "name"
void setAttribute(String name, String value)
    throws DOMException
    // erzeugt oder überschreibt Attribut mit diesem Wert.
void removeAttribute(String name) throws DOMException
    // löscht das Attribut mit diesem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

## 5.5. 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 bestimmten Namen:

```
public Node findSubNode(String name, Node node) {  
    if (node.getNodeType() != Node.ELEMENT_NODE) return null;  
    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;  
        }  
    }  
    return null;  
}
```

# 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

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

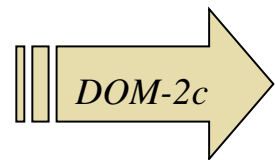
# NamedNodeMap (2)

```
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.
```

# Listen-Bearbeitung des DOM-Baums

Die Knotenlisten (vom Typ NodeList oder NamedNodeMap) zur Bearbeitung des DOM-Baums sind fast immer live, d.h.:

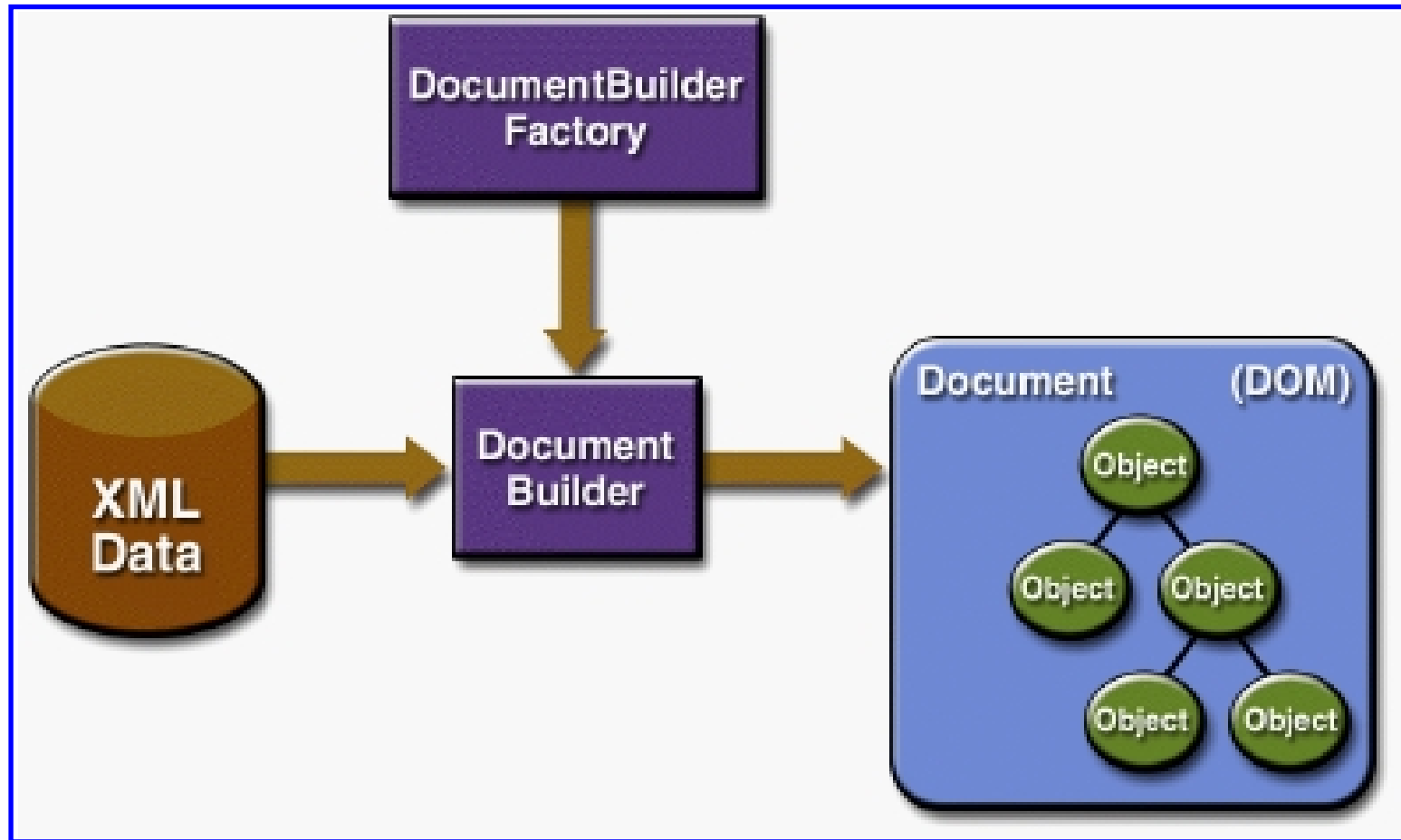
- Veränderungen der Knoten in so einer Knotenliste verändern auch den DOM-Baum (z.B.: Element-Knoten in childNode-Liste verändern, Attribut in attributes-NamedNodeMap hinzufügen...)
- Veränderungen des DOM-Baums (z.B. durch andere Threads) verändern auch diese Knotenlisten.



## 5.6. Benutzung von DOM in JAXP

- Überblick
- Mindest-Code
- Einige weitere Anweisungen

# Benutzung von DOM in JAXP



# 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-Instanziierung:

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

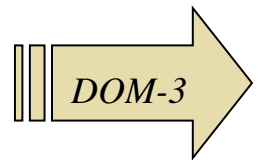
## ■ Parser-Instanziierung und Parsen:

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

# Mindest-Code (2)

## ■ Einstellungen des Parsers:

```
Document document;  
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
factory.setValidating(true);  
factory.setNamespaceAware(true);  
factory.setIgnoringComments(true);  
try {  
    DocumentBuilder builder = factory.newDocumentBuilder();  
    builder.setErrorHandler(new MyErrorHandler());  
    document = builder.parse( new File(filename) );  
} catch (SAXParseException spe)
```



# Einige weitere Anweisungen

## ■ Neuen DOM-Baum erzeugen:

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
try {  
    DocumentBuilder builder = factory.newDocumentBuilder();  
    document = builder.newDocument();  
} catch (ParserConfigurationException pce)
```

## ■ Neuen Knoten erzeugen und einfügen:

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

## 5.7. DOM-Ausgabe

---

- Überblick
- Ausgabe mittels Transformer
- Ausgabe mittels LSSerializer

# Überblick

## DOM-Baum als XML-Dokument ausgeben:

### ■ mittels Transformer

- ☐ 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

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

# 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();
```

