

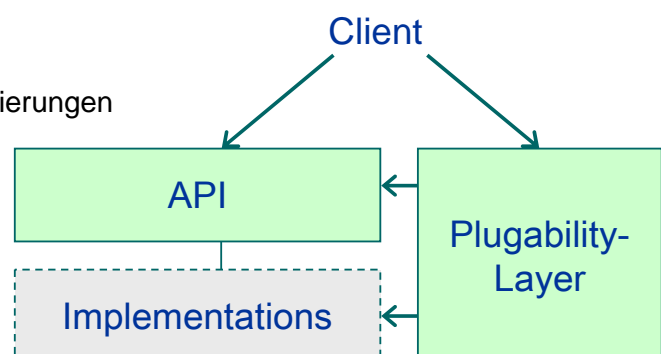
XML

- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API-Auszug



JAXP - XML-Verarbeitung in Java

- JAXP – Java API for XML Processing (<http://java.sun.com/xml/jaxp/index.jsp>)
 - unterstützt Parsen und Transformation von XML-Dokumenten
- JAXP enthält
 - Standards APIs zur XML-Verarbeitung
 - `org.w3c.dom`
 - `org.xml.sax`
 - `java.xml.stream`
 - Implementierungen
 - `javax.xml.parsers`
 - *Plugability*: Bindung der Implementierungen an APIs über Factory-Pattern
 - ➔ Implementierung bleibt verborgen
 - ➔ Implementierung kann ausgetauscht werden!



XML-Dokumente

- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



XML-Dokumente

- XML-Dokumente bestehen aus
 - Deklarationen
 - Baum von Elementen
- jedes Element hat
 - Namen
 - Inhalt
 - Attribute
- jedes Element kann eindeutige ID haben
 - Attribut id
 - Wert eindeutig im Dokument
- Struktur der Elemente

```
<element_name attributes>  
  content  
</element_name>
```

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE addressbook SYSTEM "addressbook.dtd">  
<addressbook owner="p1" date="2005-03-12">  
  <person id="p1">  
    <firstname>Thomas</firstname>  
    <lastname>Kotzmann</lastname>  
    <email>kotzmann@ssw.jku.at</email>  
  </person>  
  <company id="c1">  
    <companyname>Sun</companyname>  
    <url>www.sun.com</url>  
  </company>  
  <person id="p2">  
    <firstname>Markus</firstname>  
    <lastname>Loeberbauer</lastname>  
    <email>loeberbauer@ssw.jku.at</email>  
  </person>  
  <person id="p3">  
    <firstname>Herbert</firstname>  
    <lastname>Praehofer</lastname>  
    <email type="home">praehofer@gmx.at</email>  
  </person>  
</addressbook>
```



Hierarchie und Navigation

- Wurzelement (*root*)

addressbook-Element

- Kinder (*children*)

person-Elemente sind Kinder von addressbook

firstname, lastname und email-Elemente sind Kinder von person

- Vater (*parent*)

person-Elemente sind Väter der firstname, lastname und email-Elemente

- Geschwister (*siblings*)

firstname und lastname-Elemente sind Geschwister von email-Element

- Vorgänger (*ancestors*)

alle Väter und deren Väter

- Nachfolger (*descendants*)

Alle Kinder und deren Kinder

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE addressbook SYSTEM "addressbook.dtd">
<addressbook owner="p1" date="2005-03-12">
  <person id="p1">
    <firstname>Thomas</firstname>
    <lastname>Kotzmann</lastname>
    <email>kotzmann@ssw.jku.at</email>
  </person>
  <company id="c1">
    <companyname>Sun</companyname>
    <url>www.sun.com</url>
  </company>
  <person id="p2">
    <firstname>Markus</firstname>
    <lastname>Loeberbauer</lastname>
    <email>loeberbauer@ssw.jku.at</email>
  </person>
  <person id="p3">
    <firstname>Herbert</firstname>
    <lastname>Praehofer</lastname>
    <email type="home">praehofer@gmx.at</email>
  </person>
</addressbook>
```



Deklarationen

- XML-Deklaration: `<?xml version encoding standalone ?>`

- *version*: Version von XML
- *encoding*: verwendete Zeichenkodierung
- *standalone*: "yes" = ohne Verweis, "no" = mit Verweis auf andere Dokumente

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
```

- Deklaration des verwendeten Schemas (DTD)

- Name des Wurzelements
- verwendetes Schema
z.B. URL der verwendeten DTD-Datei

```
<!DOCTYPE addressbook SYSTEM "addressbook.dtd">
```



- XML-Dokumente
- **Schemadefinition**
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



Schemadefinition

- Definition der erlaubten Elemente und der Struktur des Dokuments
- XML-Dokumente können gegen Schema **validiert** werden
- 2 Arten:
 - XML Schema: Schema eines XML-Dokuments in XML
 - DTDs: älteres Format

- DTD besteht aus

- Deklaration der Elemente: **!ELEMENT**

```
<!ELEMENT element_name (content_model)>
```

- Deklaration der Attribute für die Elemente: **!ATTLIST**

```
<!ATTLIST target_elem attr_name attr_type default ...>
```



!ELEMENT-Deklaration

```
<!ELEMENT element_name (content_model)>
```

- besteht aus

- Name des Elements
- Inhalt des Elements

- Inhalt

- Kinderelemente

```
<!ELEMENT person (firstname, lastname, email)>
```

- Anzahl: * für [0.. ∞], + für [1 .. ∞], ? für [0, 1]

```
<!ELEMENT addressbook (person*)>
```

- Alternativen

```
<!ELEMENT addressbook ((person | company)*)>
```

- Textdaten: PCDATA

```
<!ELEMENT firstname (#PCDATA)>
```



!ATTLIST-Deklarationen

```
<!ATTLIST target_elem attr_name attr_type default ...>
```

- besteht aus

- Name des Elements für die Attributdeklarationen
 - Name des Attributs
 - Typ des Attributs
 - Default-Deklaration
- beliebig viele

```
<!ATTLIST addressbook owner IDREF #REQUIRED  
date CDATA #IMPLIED>
```

```
<!ATTLIST person id ID #REQUIRED>
```

```
<!ATTLIST email type (home | business) #IMPLIED>
```



!ATTLIST-Deklaration: Typ

- CDATA: beliebige Zeichenkette für Attributwert (ohne Leerzeichen)

```
<!ATTLIST addressbook date CDATA #IMPLIED>
```
- ID: Name als eindeutige ID für Element (darf nicht mit Ziffer beginnen !)

```
<!ATTLIST person id ID #REQUIRED>
```
- IDREF (IDREFS): Referenz zu einem Element (mehrerer Elemente) durch Angabe des(r) IDs des Elements (der Elemente)

```
<!ATTLIST addressbook owner IDREF #REQUIRED>
```
- Enumeration: Angabe der Werte in einer Aufzählung

```
<!ATTLIST email type (home | business) #IMPLIED>
```
- NMTOKEN (NMTOKENS): Angabe eines (mehrerer) Namen



!ATTLIST-Deklaration: Default-Deklaration

- #IMPLIED: Angabe des Attributs ist optional

```
<!ATTLIST addressbook date CDATA #IMPLIED>
```
- #REQUIRED: Angabe des Attributs ist gefordert

```
<!ATTLIST person id ID #REQUIRED>
```
- #FIXED "value": Attributwert ist konstant und unveränderlich

```
<!ATTLIST addressbook owner IDREF #FIXED "p1">
```
- "value": Default-Wert

```
<!ATTLIST email type (home | business) "business">
```



Beispiel DTD

```
<!ELEMENT addressbook ((person | company)*)>
<!ATTLIST addressbook owner IDREF #REQUIRED
    date CDATA #IMPLIED>
<!ELEMENT person (firstname, lastname, email)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLIST email type (home | business) "business">
<!ELEMENT company (companyname, url)>
<!ATTLIST company id ID #REQUIRED>
<!ELEMENT companyname (#PCDATA)>
<!ELEMENT url (#PCDATA)>
```

addressbook.dtd

addressbook.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE addressbook SYSTEM "addressbook.dtd">
<addressbook owner="p1" date="2005-03-12">
  <person id="p1">
    <firstname>Thomas</firstname>
    <lastname>Kotzmann</lastname>
    <email>kotzmann@ssw.jku.at</email>
  </person>
  <company id="c1">
    <companyname>Sun</companyname>
    <url>www.sun.com</url>
  ...
</addressbook>
```



XML

- XML-Dokumente
- Schemadefinition
- **DOM-Parser**
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



Parsen von XML-Dokumenten

3 Arten von Parser:

- DOM: Document Object Model
 - Hauptspeicherrepräsentation des XML-Dokuments wird erstellt
 - API: `org.w3c.dom`
 - Implementierung: `javax.xml.parsers`
- SAX: Simple API for XML
 - Ereignisse beim Lesen zeigen Inhalte an
 - API: `org.xml.sax`
 - Implementierung: `javax.xml.parsers`
- StAX: Streaming API for XML
 - Sequentielles Durchlaufen von XML
 - API/Implementierung: `javax.xml.stream`

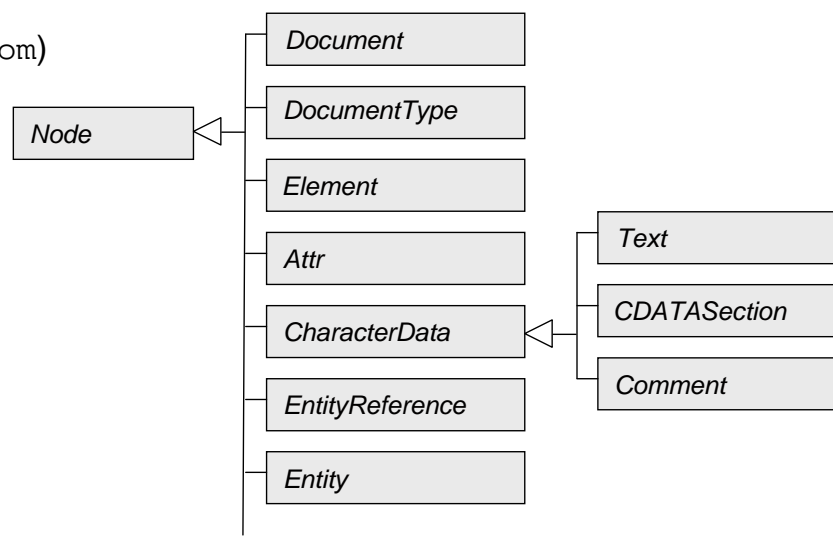


DOM

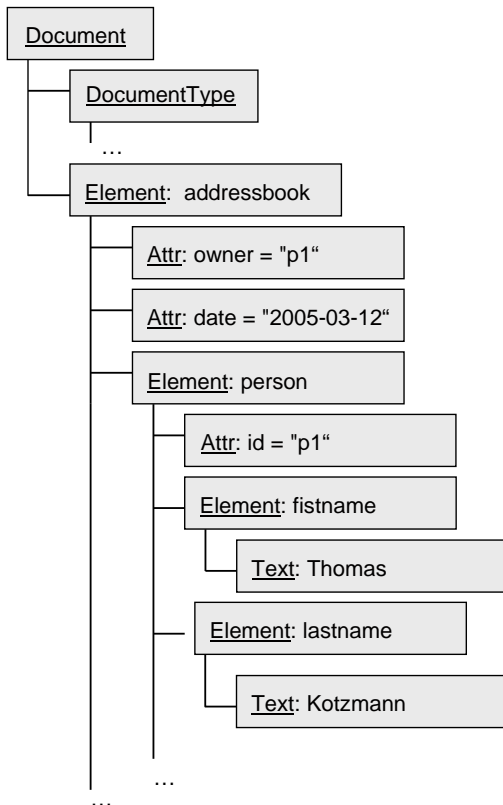
- Hauptspeicherrepräsentation des Inhalts einer XML-Datei
- Knotenstruktur (DOM-Baum)
- objektorientierter Zugriff

Interfaces:

(aus `org.w3c.dom`)



Beispiel DOM-Baum



```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE addressbook SYSTEM "addressbook.dtd">
<addressbook
  owner="p1"
  date="2005-03-12">
  <person
    id="p1">
    <firstname>
      Thomas </firstname>
    <lastname>
      Kotzmann </lastname>
  </person>
</addressbook>
```



DocumentBuilder: Parsen eines XML-Dokuments

- Parsen erfolgt mittels DocumentBuilder
- DocumentBuilder wird durch DocumentBuilderFactory erzeugt

```
DocumentBuilderFactory factory
    = DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setIgnoringElementContentWhitespace(true);

try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    File file = new File("addressbook.xml");
    Document doc = builder.parse(file);
    ...
} catch (ParserConfigurationException e) { ...
} catch (SAXException e) { ...
} catch (IOException e) { ...
}
```

repräsentiert XML-Dokument



Navigieren im DOM-Baum (1)

- Zugriff auf DOM-Baum vom Document-Knoten ausgehend
 - `getDocumentElement` liefert Wurzelement

```
Element addressbook = doc.getDocumentElement();  
System.out.print(addressbook.getNodeName());
```

- Attribute eines Knotens mit `getAttributes`

```
NamedNodeMap attrs = addressbook.getAttributes();  
Attr owner = (Attr) attrs.getNamedItem("owner");  
Attr date = (Attr) attrs.getNamedItem("date");
```

- Zugriff auf Unterknoten mit `getChildNodes` und auf Attribute mit `getAttribute`

```
NodeList entries = addressbook.getChildNodes();  
for (int i = 0; i < entries.getLength(); i++) {  
    Element entry = (Element)entries.item(i);  
    if (entry.getNodeName().equals("person")) {  
        String id = entry.getAttribute("id");
```



Navigieren im DOM-Baum (2)

- Zugriff auf bestimmte Unterknoten mit `getElementsByTagName`

```
NodeList fnNodeList= entry.getElementsByTagName("firstname");
```

- Zugriff auf Listenelemente und Unterelemente

```
Element fnElem = (Element) fnNodeList.item(0);  
Text fnText = (Text) fnElem.getFirstChild();  
String fnString = fnText.getNodeValue();
```

- Direktzugriff auf Elemente mit eindeutiger ID (Methode `getElementById` von Document)

```
Element getElementById(String id)
```

```
Element owner =  
doc.getElementById(doc.getDocumentElement().getAttribute("owner"));
```



Beispiel: Ausgabe von addressbook.xml (1)

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setIgnoringElementContentWhitespace(true);

try {
    DocumentBuilder builder =
        factory.newDocumentBuilder();
    File file = new File("addressbook.xml");
    Document doc = builder.parse(file);
    // root element
    Element addressbook = doc.getDocumentElement();
    // attributes of address book
    Attr owner = addressbook.getAttributeNode("owner");
    String date = addressbook.getAttribute("date");
    System.out.println(" " + addressbook.getNodeName() +
        ": " + owner.getNodeName() + " = " + owner.getValue() +
        ", date = " + date);
    // subelements person | company
    NodeList entries = addressbook.getChildNodes();
    for (int i = 0; i < entries.getLength(); i++) {
        Element entry = (Element)entries.item(i);
        // id attribute
        String id = entry.getAttribute("id");
        System.out.println(" " + entry.getNodeName() + ": id = " + id);
        ...
    }
}
```

```
addressbook: owner = p1, date = ..
person: id = p1
  Name: Thomas Kotzmann
  Email: kotzmann@ssw.jku.at
company: id = c1
  Name: Sun
  URL: www.sun.com
person: id = p2
  Name: Markus Loeberbauer
  Email: loeberbauer@ssw.jku.at
person: id = p3
  Name: Herbert Praehofer
  Email: praehofer@gmx.at
```



Beispiel: Ausgabe von addressbook.xml (2)

```
if (entry.getNodeName().equals("person")) {
    // first Name
    NodeList fnNodeList = entry.getElementsByTagName("firstname");
    Element fnElem = (Element)fnNodeList.item(0);
    Text fnText = (Text)fnElem.getFirstChild();
    String fnString = fnText.getNodeValue();
    // lastName
    NodeList lnNodeList = entry.getElementsByTagName("lastname");
    Element lnElem = (Element)lnNodeList.item(0);
    Text lnText = (Text)lnElem.getFirstChild();
    String lnString = lnText.getNodeValue();
    // emails
    NodeList emailNodeList = entry.getElementsByTagName("email");
    Element emailElem = (Element)emailNodeList.item(0);
    Text emailText = (Text)emailElem.getFirstChild();
    String emailString = emailText.getNodeValue();
    System.out.println("  Name: " + fnString + " " + lnString);
    System.out.println("  Email: " + emailString);
} else if (entry.getNodeName().equals("company")) {
    Node cnNode = entry.getElementsByTagName("companyname").item(0);
    Node urlNode = entry.getElementsByTagName("url").item(0);
    System.out.println("  Name: " + cnNode.getFirstChild().getNodeValue());
    System.out.println("  URL: " + urlNode.getFirstChild().getNodeValue());
}
} // end for
} catch (ParserConfigurationException e) { ...
} catch (SAXException e) { ...
} catch (IOException e) { ... }
```



- XML-Dokumente
- Schemadefinition
- DOM-Parser
- **SAX-Parser**
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



SAX-Parser

- SAX-Parser liest XML-Dokument und generiert Ereignisse
 - Anfang und Ende von Dokument,
 - Anfang und Ende von Elementen,
 - Auftreten von Textknoten, etc.
- Ereignisbehandlung in `ContentHandler` (implementiert von `DefaultHandler`)

```
public interface ContentHandler {  
    void startDocument() throws SAXException  
    void endDocument() throws SAXException  
    void startElement( String namespaceURI, String localName,  
                      String qName, Attributes atts)  
        throws SAXException  
    void endElement( String namespaceURI, String localName,  
                    String qName, Attributes atts)  
        throws SAXException  
    void characters(char[] ch, int start, int length)  
        throws SAXException  
    ...  
}
```

```
public class DefaultHandler implements ContentHandler,  
    EntityResolver, DTDHandler, ErrorHandler
```



Vorgehen

- SAXParserFactory erzeugen
- SAXParser erzeugen
- parse mit Datei und Handler-Objekt aufrufen

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
try {
    SAXParser saxParser = factory.newSAXParser();
    // parse ...
    File file = new File("addressbook.xml");
    saxParser.parse(file, new PrintElementsHandler());
} catch (ParserConfigurationException e1) { ...
} catch (SAXException e1) { ...
} catch (IOException e) { ...
}
```

Handler-Objekt



Handler

- Im Handler auf die Ereignisse reagieren und Inhalt auswerten

```
class PrintElementsHandler extends DefaultHandler {
    public void startDocument() throws SAXException {
        // Anfang von Dokument behandeln
    }

    public void startElement(String uri, String localName, String qName,
        Attributes attributes) throws SAXException {
        // Element behandeln
    }

    public void characters(char[] ch, int start, int length)
        throws SAXException {
        // Textknoten behandeln
    }

    public void endElement(String uri, String localName, String qName)
        throws SAXException {
        // Ende von Element behandeln
    }

    public void endDocument() throws SAXException {
        // Ende von Dokument behandeln
    }
}
```

Klasse DefaultHandler erweitern!



Beispiel: Ausgabe von addressbook.xml (1)

- Ausgabe der Elemente, Attribute und Textknoten
- mit Einrückung und Tiefe des Knotens

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

public class SAXParserTest {

    public static void main(String[] args) {

        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setValidating(true);
        SAXParser saxParser;
        try {
            saxParser = factory.newSAXParser();
            // parse ...
            File file = new File("addressbook.xml");
            saxParser.parse(file, new PrintElementsHandler());
        } catch (ParserConfigurationException e1) {
        } catch (SAXException e1) {
        } catch (IOException e) {
        }
    }
}
```



Beispiel: Ausgabe von addressbook.xml (2)

```
class PrintElementsHandler extends DefaultHandler {

    StringBuffer blanks = new StringBuffer(); // leading blanks
    int depth = 0; // depth of node

    public void startDocument() throws SAXException {
        System.out.println("start document -----");
    }

    public void startElement(String uri, String localName, String qName,
        Attributes attributes) throws SAXException {
        blanks.append(" "); // add 3 blanks
        depth++; // increase depth
        System.out.print(blanks.toString() + depth + " - " + qName + ": ");
        for (int i = 0; i < attributes.getLength(); i++) {
            System.out.print(attributes.getQName(i) + " = " +
                attributes.getValue(i) + " ");
        }
        System.out.println();
    }

    public void characters(char[] ch, int start, int length)
        throws SAXException {
        String s = new String(ch, start, length);
        System.out.println(blanks.toString() + " " + (depth+1) + " - " + s);
    }
    ...
}
```



Beispiel: Ausgabe von addressbook.xml (3)

```
public void endElement(String uri, String localName, String qName)
    throws SAXException {
    blanks.delete(0, 3);    // delete 3 blanks
    depth--;               // decrease depth
}

public void endDocument() throws SAXException {
    System.out.println("end document -----");
}
}

start document -----
1 - addressbook: owner = p1 date = 2005-03-12
2 - person: id = p1
  3 - firstname:
    4 - Thomas
  3 - lastname:
    4 - Kotzmann
  3 - email: type = business
    4 - kotzmann@ssw.jku.at
2 - company: id = c1
  3 - companyname:
    4 - Sun
  3 - url:
    4 - www.sun.com
2 - person: id = p2
...

end document -----
```



XML

- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- **StAX-Reader**
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



StAX-Reader (Java 6.0)

- XML-Dokument lesen mit StAX
 - Sequentielles Lesen eines Dokuments
 - PULL-Methode, d.h. Verwender liest Elemente

- Vorgehen

- Erstellen einer XMLInputFactory
- XMLStreamReader erzeugen
- Sequentielles Lesen des XML-Dokuments mit XMLStreamReader

Im Gegensatz zur PUSH-Methode eines SAX-Parsers !!

```
public interface XMLStreamReader {
    int next() throws XMLStreamException;
    boolean hasNext() throws XMLStreamException;
    int getEventType();
    boolean isStartElement();
    boolean isEndElement();
    String getText();
    String getLocalName();
    String getNamespaceURI();
    int getAttributeCount();
    String getAttributeName(int i);
    String getAttributeValue(int i);
    String getAttributeValue(String namespaceURI, String localName);
    String getAttributeType(int i);
    void close()
    // ...
}
```

Konstante:
ATTRIBUTE
CDATA
START_DOCUMENT
START_ELEMENT
END_DOCUMENT
END_ELEMENT
SPACE
...



StAX Beispiel MovieDB lesen 1/2

```
List<Movie> readMovies() {
    List<Movie> movies = new ArrayList<Movie>();
    FileInputStream movieFile = null;
    try {
        movieFile = new FileInputStream("movies.xml");
        XMLInputFactory xmlInFact = XMLInputFactory.newInstance();
        XMLStreamReader movieReader = xmlInFact.createXMLStreamReader(movieFile);
        readMovies(movieReader, movies);
    } catch (IOException exc) {
        // ... log ...
    } catch (XMLStreamException exc) {
        // ... log ...
    } finally {
        if (movieFile != null) {
            try {
                movieFile.close();
            } catch (IOException exc) {
                // ... log ...
            }
        }
    }
    return movies;
}
```

```
<?xml version="1.0" ?>
- <movielist>
- <movie name="Return of the Tutors ... ">
  <actor name="Kurt P." />
</movie>
<movie name="A Story without Actors." />
- <movie name="The Lecture of Horror!">
  <actor name="Markus Loeberbauer" />
  <actor name="Herbert Praehofer" />
</movie>
</movielist>
```



StAX Beispiel *MovieDB* lesen 2/2

```
void readMovies(XMLStreamReader movieReader, List<Movie> movies)
    throws XMLStreamException {
    Movie m = null;
    while (movieReader.hasNext()) {
        movieReader.next();
        if (movieReader.getEventType() != XMLStreamReader.START_ELEMENT) {
            continue;
        }
        if (Movie.ID.equals(movieReader.getLocalName())) {
            m = new Movie();
            m.name = movieReader.getAttributeValue(null, Movie.NAME_PROP);
            movies.add(m);
        }
        else if (Actor.ID.equals(movieReader.getLocalName())) {
            Actor a = new Actor();
            a.name = movieReader.getAttributeValue(null, Actor.NAME_PROP);
            m.actors.add(a);
        }
    }
}
```

NamespaceURI

Lokaler Name

```
class Movie {
    public static final String ID = "movie";
    public static final String NAME_PROP = "name";
    public String name;
    public List<Actor> actors = new ArrayList<Actor>();
}
class Actor {
    public static final String ID = "actor";
    public static final String NAME_PROP = "name";
    public String name;
}
```



XML

- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- **Erzeugen von XML-Dokumenten aus DOM**
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



Erzeugen von XML-Dokumenten

- XML-Dokumente werden erzeugt durch
 - Aufbauen eines DOM-Baums
 - Schreiben über XSL-Transformation
- Vorgehen beim Aufbau eines DOM-Baums
 - Erzeugen eines neuen Document-Objekts mittels DocumentBuilder

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

- Erzeugen neuer Elemente für Document-Objekt

```
Element root = doc.createElement("addressbook");  
Element person1 = doc.createElement("person");
```

- Anfügen der Elemente bei Vaterknoten

```
doc.appendChild(root);  
root.appendChild(person1);
```

- Anfügen von Attributen bei Elementen

```
root.setAttribute("date", "2005-03-22");  
person1.setAttribute("id", "p1");
```



Beispiel: Erzeugen des addressbook-Dokuments (1)

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
DocumentBuilder builder;  
try {  
    builder = factory.newDocumentBuilder();  
    Document doc = builder.newDocument();  
    Element addressbook = doc.createElement("addressbook");  
    addressbook.setAttribute("owner", "p1");  
    addressbook.setAttribute("date", "2005-03-22");  
    doc.appendChild(addressbook);  
  
    Element person = doc.createElement("person");  
    person.setAttribute("id", "p1");  
    addressbook.appendChild(person);  
    Element firstName = doc.createElement("firstname");  
    firstName.appendChild(doc.createTextNode("Thomas"));  
    person.appendChild(firstName);  
    Element lastName = doc.createElement("lastname");  
    lastName.appendChild(doc.createTextNode("Kotzmann"));  
    person.appendChild(lastName);  
    Element email = doc.createElement("email");  
    email.appendChild(doc.createTextNode("kotzmann@ssw.jku.at"));  
    person.appendChild(email);  
  
    Element company = doc.createElement("company");  
    company.setAttribute("id", "c1");  
    addressbook.appendChild(company);  
    ...  
}
```



Erzeugen von XML-Dokumenten (2)

- Schreiben des DOM-Baums mittels XSL-Transformation
 - Quelle *DOM-Baum* in Ziel *Datei* transformieren (Achtung: es erfolgt keine wirkliche Datentransformation)
- Vorgehen beim Schreiben über XSL-Transformation

- Erzeugen eines Transformers

```
TransformerFactory fact = TransformerFactory.newInstance();  
Transformer t = fact.newTransformer();
```

- Ausgabeigenschaften setzen (für DOCTYPE und Leerzeilen)

```
t.setOutputProperty("doctype-system", "addressbook.dtd");  
t.setOutputProperty("indent", "yes");
```

- Transformation mit Dokument als Quelle und FileOutputStream als Ziel anwenden

```
t.transform(  
    new DOMSource(doc), // source  
    new StreamResult(new FileOutputStream(file)) // target  
);
```



Beispiel: Erzeugen des addressbook-Dokuments (2)

```
...  
File file = new File(" ... ");  
TransformerFactory fact = TransformerFactory.newInstance();  
Transformer t = fact.newTransformer();  
t.setOutputProperty("doctype-system", "addressbook.dtd");  
t.setOutputProperty("indent", "yes");  
t.transform(  
    new DOMSource(doc), // source  
    new StreamResult(new FileOutputStream(file)) // target  
);  
  
} catch (ParserConfigurationException e) {  
    ...  
} catch (TransformerConfigurationException e1) {  
    ...  
} catch (FileNotFoundException e) {  
    ...  
} catch (TransformerException e) {  
    ...  
}
```



- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- **Schreiben von XML-Dokumenten mit StAX**
- XSL-Transformation
- Java Architecture for XML Binding
- API – Auszug



StAX-Writer

- XML-Dokument schreiben mit StAX
 - Im Gegensatz zu SAX kann man mit StAX auch schreiben
 - Schreiben erfolgt wie das Lesen sequentiell
 - ausgehend vom Verwender
- Vorgehen
 - Erstellen einer XMLOutputFactory
 - XMLStreamWriter erstellen
 - Schreiben des XML-Dokuments

```
public interface XMLStreamWriter {  
    void writeStartDocument() throws XMLStreamException;  
    void writeStartElement(String localName) throws XMLStreamException;  
    void writeEndDocument() throws XMLStreamException;  
    void writeStartElement(String localName) throws XMLStreamException;  
    void writeEndElement() throws XMLStreamException;  
    void writeCharacters(String text) throws XMLStreamException;  
    void writeAttribute(String localName, String value) throws XMLStreamException;  
    void flush();  
    void close();  
    // ...  
}
```



StAX Beispiel MovieDB schreiben 1/2

```
void writeMovies(List<Movie> movies) {
    FileOutputStream movieFile = null;
    try {
        movieFile = new FileOutputStream("movies.xml");
        XMLOutputFactory xmlOutFact = XMLOutputFactory.newInstance();
        XMLStreamWriter movieWriter = xmlOutFact.createXMLStreamWriter(movieFile);
        writeMovies(movieWriter, movies);
    } catch (IOException exc) {
        // ... log ...
    } catch (XMLStreamException exc) {
        // ... log ...
    } finally {
        if (movieFile != null) {
            try {
                movieFile.close();
            } catch (IOException exc) {
                // ... log ...
            }
        }
    }
}
```



StAX Beispiel MovieDB schreiben 2/2

```
void writeMovies(XMLStreamWriter movieWriter, List<Movie> movies)
    throws XMLStreamException {
    movieWriter.writeStartDocument();
    movieWriter.writeStartElement("movielist");
    for (Movie m : movies) {
        movieWriter.writeStartElement(Movie.ID);
        movieWriter.writeAttribute(Movie.NAME_PROP, m.name);
        for (Actor a : m.actors) {
            movieWriter.writeEmptyElement(Actor.ID);
            movieWriter.writeAttribute(Actor.NAME_PROP, a.name);
        }
        movieWriter.writeEndElement();
    }
    movieWriter.writeEndElement();
    movieWriter.flush();
}
```

Lokaler Name

Wert



- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM aus DOM
- Schreiben von XML-Dokumenten mit StAX
- **XSL-Transformation**
- Java Architecture for XML Binding
- API – Auszug



XML-Transformation mit XSLT

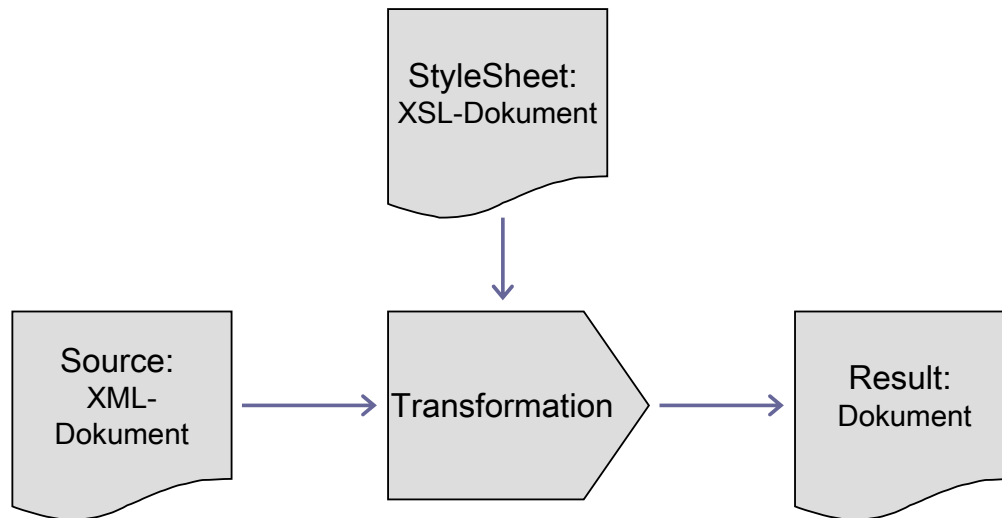
- XSLT ermöglicht die Transformation von XML-Dokumenten
- *XSL-Stylesheet* ist ein Dokument mit einer Menge von Transformationsregeln
- Regeln (*templates*) beschreiben die Transformation von XML-Elementen
 - XPath-Ausdrücke definieren die Prämissen der Regeln (*match*)
 - im Rumpf der Regel wird die Generierung des transformierten Elements beschrieben

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
<xsl:template  
  match=xpath-expression>  
  construction of transformed elements  
</xsl:template>  
...  
</xsl:stylesheet>
```



XSL-Transformation

- *Source*: das zu transformierende XML-Dokument
- *Result*: Zieldaten in beliebigem Format (z.B. HTML)
- *StyleSheet*: XSL-Dokument mit Transformationsregeln



XPath

- *XPath* ist ein Adressierungsmechanismus für Elemente in XML-Dokumenten
- XPath-Ausdruck (*Location path*) selektiert eine Menge von Knoten
- Ein *Location path* besteht aus *Location steps*, die mit "/" getrennt sind
- *Location steps* können definieren
 - Elementnamen: *element_name*
 - Attribute: *@attribute*
 - Bedingungen: [condition]
 - Positionen: [index]

Beispiele von Location paths:

" * "	Selektiert alle Knoten
" / "	Wurzelknoten
" /addressbook/ * "	Selektiert alle Elemente unter einen addressbook-Element
" /addressbook/person[1] "	Liefert die ersten person-Elemente von addressbook-Elementen
" /addressbook/ * /firstname "	Liefert alle firstname-Element unter den addressbook-Elementen
" /addressbook/person[@id="p1] "	Liefert den Personenknoten mit id = "p1"
" / * /email[@type="home] "	Liefert alle email-Knoten die von Typ "home" sind



Beispiel XSL-Stylesheet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <head> <title>XML Address Book</title> </head>
      <body>
        <table border="3" cellspacing="10" cellpadding="5">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="addressbook">
    <xsl:apply-templates select="person"/>
  </xsl:template>

  <xsl:template match="person">
    <tr>
      <td> <xsl:value-of select="firstname"/> </td>
      <td> <b><xsl:value-of select="lastname"/></b> </td>
      <td> <xsl:value-of select="email"/> </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```



Beispiel Transformation

ursprüngliches XML-Dokument

```
<?xml version='1.0' encoding="utf-8"?>
<addressbook owner="1">
  <person id="1">
    <firstname>Thomas</firstname>
    <lastname>Kotzmann</lastname>
    <email>kotzmann@ssw.jku.at</email>
  </person>
  <person id="2">
    <firstname>Markus</firstname>
    <lastname>Loeberbauer</lastname>
    <email>loeberbauer@ssw.jku.at</email>
  </person>
</addressbook>
```

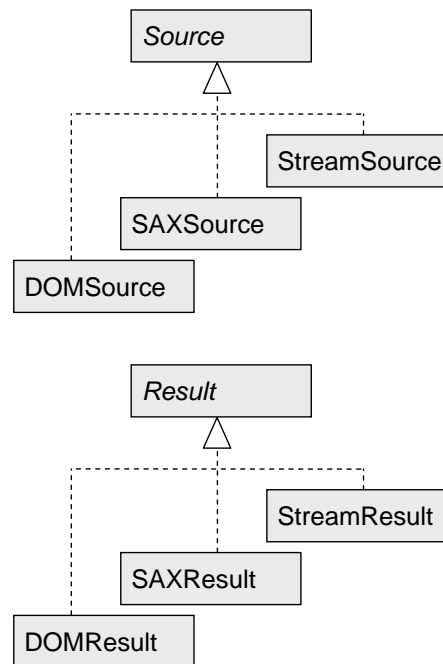
erzeugtes HTML-Dokument

```
<html>
<head>
  <META http-equiv="Content-Type"
  content="text/html; charset=utf-8">
  <title>XML-AddressBook</title>
</head>
<body>
  <table border="3" cellspacing="10" cellpadding="5">
    <tr>
      <td>Thomas</td>
      <td><b>Kotzmann</b></td>
      <td>kotzmann@ssw.jku.at</td>
    </tr>
    <tr>
      <td>Markus</td>
      <td><b>Loeberbauer</b></td>
      <td>loeberbauer@ssw.jku.at</td>
    </tr>
  </table>
</body>
</html>
```



XSL-Transformation in JAXP

- Package `javax.xml.transform`
- Transformer und TransformerFactory
- Abstraktion von Quellen (inkl StyleSheets) und Senken
 - Interfaces
 - Source
 - Result
 - Implementierungen
 - DOMSource, SAXSource, StreamSource
 - DOMResult, SAXResult, StreamResult



Vorgehen

- Source für StyleSheet und Transformer erzeugen

```
TransformerFactory facty = TransformerFactory.newInstance();
File styleSheetFile = new File("addressbook.xsl");
StreamSource styleSheet = new StreamSource(styleSheetFile);
Transformer t = facty.newTransformer(styleSheet);
```

- Source für Quelldokument erzeugen

```
File inFile = new File("addressbook.xml");
Document doc = builder.parse(inFile);
DOMSource source = new DOMSource(doc);
```

Transformer mit Stylesheet

- Result für Ergebnis erzeugen

```
File outFile = new File("addressbook.html");
StreamResult result = new StreamResult(
    new FileOutputStream(
        outFile));
```

- Transformation durchführen

```
t.transform(source, result);
```



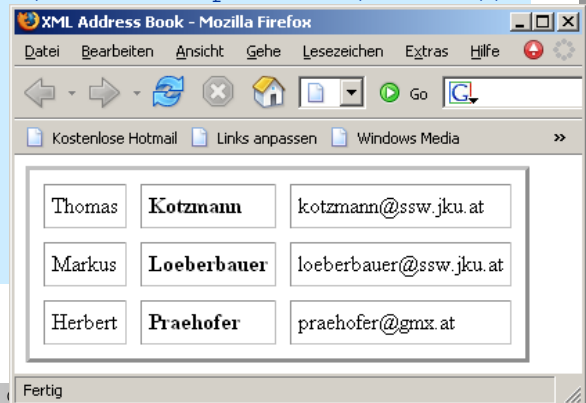
Beispiel: Transformation XML - HTML

```
try {
    ...
    // create transformer with style sheet
    TransformerFactory fact = TransformerFactory.newInstance();
    File styleSheetFile = new File("addressbook.xsl");
    StreamSource styleSheet = new StreamSource(styleSheetFile);
    Transformer t = fact.newTransformer(styleSheet);

    // create source
    File inFile = new File("addressbook.xml");
    Document doc = builder.parse(inFile);
    DOMSource source = new DOMSource(doc);

    // create target
    File outFile = new File("addressbook.html");
    StreamResult result = new StreamResult(new FileOutputStream(outFile));

    // transform
    t.transform(source, result);
} catch (TransformerException e) {
    ...
}
```



XML

- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- **Java Architecture for XML Binding**
- API – Auszug



Java Architecture for XML Binding (JAXB 2.0)

- Bindet Objektzustände an XML-Dokumente
 - Abbildung durch Annotationen festlegbar
 - Automatisches Serialisieren ganzer Objektbäume
- Vorgehen
 - Annotieren der Klassen, Felder, Properties
 - JAXBContext anlegen, bekanntgeben der betroffenen Klassen
 - Lesen mit einem Unmarshaller
 - Schreiben mit einem Marshaller



Beispiel *MovieDB* annotieren (JAXB 2.0)

```
@XmlElement(name="movielist")
class MovieList implements Iterable<Movie> {
    public List<Movie> movies = new ArrayList<Movie>();

    public void add(Movie m) {
        movies.add(m);
    }

    public Iterator<Movie> iterator() {
        return movies.iterator();
    }
}

@XmlRootElement(name="movie")
class Movie {
    @XmlAttribute
    public String name;

    @XmlElement(name="actor-list")
    public List<Actor> actors = new ArrayList<Actor>();
}

@XmlRootElement
class Actor {
    @XmlElement(name="actor-name")
    public String name;
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <movielist>
- <movies name="Return of the Tutors ..." >
- <actor-list>
  <actor-name>Kurt P.</actor-name>
</actor-list>
</movies>
<movies name="A Story without Actors." />
- <movies name="The Lecture of Horror!" >
- <actor-list>
  <actor-name>Markus Loeberbauer</actor-name>
</actor-list>
- <actor-list>
  <actor-name>Herbert Praehofer</actor-name>
</actor-list>
</movies>
</movielist>
```



Beispiel *MovieDB* lesen (JAXB 2.0)

```
MovieList readMovies() {
    MovieList movies = new MovieList();
    FileInputStream movieFile = null;
    try {
        movieFile = new FileInputStream("movies.xml");
        JAXBContext ctx = JAXBContext.newInstance(MovieList.class);
        Unmarshaller um = ctx.createUnmarshaller();
        movies = (MovieList) um.unmarshal(movieFile);
    } catch (IOException exc) {
        // ... log ...
    } catch (JAXBException exc) {
        // ... log ...
    } finally {
        if (movieFile != null) {
            try {
                movieFile.close();
            } catch (IOException exc) {
                // ... log ...
            }
        }
    }
    return movies;
}
```

Betroffene Klassen
(referenzierte zur Compilezeit
bekannte Klassen werden
selbstständig erkannt)



Beispiel *MovieDB* schreiben (JAXB 2.0)

```
void writeMovies(MovieList movies) {
    FileOutputStream movieFile = null;
    try {
        movieFile = new FileOutputStream("movies.xml");
        JAXBContext ctx = JAXBContext.newInstance(MovieList.class);
        Marshaller ma = ctx.createMarshaller();
        ma.marshal(movies, movieFile);
    } catch (IOException exc) {
        // ... log ...
    } catch (JAXBException exc) {
        // ... log ...
    } finally {
        if (movieFile != null) {
            try {
                movieFile.close();
            } catch (IOException exc) {
                // ... log ...
            }
        }
    }
}
```



JAXB 2.0 – API, wichtige Annotationen

- Namespaces
 - javax.xml.bind
 - javax.xml.bind.annotation
- Annotationen:
 - @XmlRootElement
 - Klassen, Enums
 - Abbildung auf XML Elemente
 - @XmlElement
 - Properties, Felder
 - Abbildung auf XML Elemente
 - @XmlAttribute
 - Properties, Felder
 - Abbildung auf XML Attribute



Zusammenfassung

- SAX, StAX, DOM oder JAXB?

	Lesen	Schreiben	Ändern	Speicherbedarf	Einfachheit
SAX	Ja	Nein	Nein	Niedrig	Komplex
StAX	Ja	Ja	Nein	Niedrig	Einfach
DOM	Ja	Ja	Ja	Hoch	Einfach
JAXB	Ja	Ja	-	-	Einfach bis Komplex



- XML-Dokumente
- Schemadefinition
- DOM-Parser
- SAX-Parser
- StAX-Reader
- Erzeugen von XML-Dokumenten aus DOM
- Schreiben von XML-Dokumenten mit StAX
- XSL-Transformation
- Java Architecture for XML Binding
- **API – Auszug**



API – Auszug (1)

`org.w3c.dom`

`interface Node`

<code>String getNodeName()</code>	Name des Knotens
<code>String getNodeValue()</code>	Wert des Knotens
<code>Document getOwnerDocument()</code>	Dokument
<code>boolean hasAttributes()</code>	testet, ob Attribute definiert
<code>NamedNodeMap getAttributes()</code>	Knotentabelle mit <code>Attr</code> -Knoten
<code>boolean hasChildNodes()</code>	testet, ob Sohnknoten vorhanden
<code>NodeList getChildNodes()</code>	Knotenliste mit untergeordneten Knoten
<code>Node getParentNode()</code>	Vaterknoten
<code>Node getFirstChild()</code>	erster untergeordneten Knoten
<code>Node getLastChild()</code>	letzter untergeordneten Knoten
<code>Node getNextSibling()</code>	nächster gleichgeordneten Knoten
<code>Node getNextSibling()</code>	nächster gleichgeordneten Knoten
<code>Node appendChild(Node newChild)</code> <code>throws DOMException</code>	Anfügen eines Sohnknotens
<code>Node removeChild(Node oldChild)</code> <code>throws DOMException</code>	Löschen eines Sohnknotens



API – Auszug (2)

`org.w3c.dom`

interface Document extends Node

<code>Element getElementElement()</code>	Wurzelement des Dokuments
<code>Element getElementById(String id)</code>	Zugriff auf Element über eindeutige ID
<code>DocumentType getDoctype()</code>	Zugriff auf DOCTYPE-Deklaration
<code>NodeList getElementsByTagName(String tagname)</code>	Zugriff auf Elemente mit bestimmten Namen
<code>Element createElement(String tagName) throws DOMException</code>	Erzeugen eines Elements für dieses Dokument
<code>Text createTextNode(String data)</code>	Erzeugen eines Textknotens für Dok.
<code>Attr createAttribute(String name) throws DOMException</code>	Erzeugen eines Attributs für dieses Dokument
<code>CDATASection createCDATASection(String data) throws DOMException</code>	Erzeugen eines CDATA-Knotens



API – Auszug (3)

`org.w3c.dom`

interface Element extends Node

<code>String getTagName()</code>	Name des Elements
<code>boolean hasAttribute(String name)</code>	Test, ob Attribute vorhanden
<code>String getAttribute(String name)</code>	Attributwert von Attribut
<code>Attr getAttributeNode(String name)</code>	Attributknoten
<code>void setAttribute(String name, String value) throws DOMException</code>	Setzen eines Attributs
<code>Attr setAttributeNode(Attr newAttr) throws DOMException</code>	Setzen eines Attributknotens
<code>void removeAttribute(String name) throws DOMException</code>	Löschen eines Attributs
<code>NodeList getElementsByTagName(String name)</code>	Zugriff auf Elemente mit Namen name



API – Auszug (4)

org.w3c.dom

interface Text extends Node

```
Text splitText(int offset)
    throws DOMException
```

Aufbrechen des Textes in zwei Knoten

org.w3c.dom

interface CharacterData extends Text

```
String getData()
    throws DOMException
```

Text für diesen Knoten

```
void setData(String data)
    throws DOMException
```

Setzen des String-Wertes

```
int getLength()
```

Länge des Textes

```
void appendData(String arg)
    throws DOMException
```

Anfügen eines Strings



API – Auszug (5)

org.w3c.dom

interface NodeList

```
int getLength()
```

Länge dieser Knotenliste

```
Node item(int index)
```

Knoten für angegebenen Index

org.w3c.dom

interface NamedNodeMap

```
int getLength()
```

Länge dieser Knotentabelle

```
Node item(int index)
```

Knoten für angegebenen Index

```
Node getNamedItem(String name)
```

Knoten mit Name name



API – Auszug (6)

`org.xml.sax`

`interface XMLReader`

- `void setFeature(String name, boolean value)`
`throws SAXNotRecognizedException, SAXNotSupportedException`
 - Setzen von Features für diesen Reader
- `void setProperty(String name, Object value)`
`throws SAXNotRecognizedException, SAXNotSupportedException`
 - Setzen von Eigenschaften für diesen Reader
- `void setDTDHandler(DTDHandler handler)`
 - Setzen des DTD-Handlers
- `void setContentHandler(ContentHandler handler)`
 - Setzen des ContentHandlers
- `void setErrorHandler(ErrorHandler handler)`
 - Setzen des ErrorHandlers
- `void parse(InputSource input) throws IOException, SAXException`
 - Parst das XML-Dokument vom Input-Source
- `void parse(String systemId) throws IOException, SAXException`
 - Parst das XML-Dokument mit angegebener URI



API – Auszug (7)

`org.xml.sax`

`interface ContentHandler`

- `void startDocument () throws SAXException`
 - Aufgerufen bei Start des Dokumentes
- `void endDocument() throws SAXException`
 - Aufgerufen bei Ende des Dokumentes
- `void startElement (String namespaceURI, String localName, String qName, Attributes atts) throws SAXException`
 - Aufgerufen bei Start eines Elements
- `void endElement (String namespaceURI, String localName, String qName) throws SAXException`
 - Aufgerufen bei Ende eines Elements
- `void characters (char ch[], int start, int length)`
`throws SAXException`
 - Aufgerufen bei Auftreten eines Textknotens
- `void processingInstruction (String target, String data)`
`throws SAXException`
 - Aufgerufen bei Auftreten einer Processing-Instruction



API – Auszug (8)

`javax.xml.parsers`

class `DocumentBuilderFactory`

static `DocumentBuilderFactory newInstance()`

- Liefert eine Instanz der Klasse

`DocumentBuilder newDocumentBuilder()`

- Liefert einen `DocumentBuilder` (= Parser)

void `setIgnoringComments(boolean ignoreComments)`

- erzeugter Parser ignoriert Kommentare

void `setIgnoringElementContentWhitespace(boolean whitespace)`

- erzeugter Parser ignoriert Leerzeichen

void `setValidating(boolean validating)`

- erzeugter Parser validiert XML-Dokument

`javax.xml.parsers`

class `DocumentBuilder`

`Document parse(File file)`

Parst XML-Dokument aus Datei

`Document parse(String url)`

Parst XML-Dokument aus URL

`Document parse(InputStream in)`

Parst XML-Dokument aus `InputStream`

`Document newDocument()`

Erzeugt neues Dokument



API – Auszug (9)

`javax.xml.parsers`

class `SAXBuilderFactory`

- Factory für `SAXParser`

static `SAXParserFactory newInstance()`

throws `FactoryConfigurationError`

- Liefert eine Instanz der Klasse

abstract `SAXParser newSAXParser()`

throws `ParserConfigurationException`, `SAXException`

- Liefert `SAXParser`

void `setValidating(boolean validating)`

- Bestimmt, ob erzeugte Parser validieren sollen



API – Auszug (10)

`javax.xml.parsers`
`class SAXParser`

- kapselt XMLReader

```
abstract void setProperty(String name, Object value)
    throws SAXNotRecognizedException, SAXNotSupportedException
```

- Setzt eine Property für den Parser

```
abstract XMLReader getXMLReader() throws SAXException
```

- Zugriff auf gekapselten Reader

```
void parse(DataSource is, DefaultHandler dh)
    throws SAXException, IOException
```

- Parst XML-Dokument aus DataSource unter Verwendung von DefaultHandler

```
void parse(InputStream is, DefaultHandler dh)
    throws SAXException, IOException
```

- Parst XML-Dokument von InputStream unter Verwendung von DefaultHandler

...

`org.xml.sax.helpers`

`class DefaultHandler`

```
extends ContentHandler, DTDHandler, EntityResolver,
ErrorHandler
```



API – Auszug (11)

`javax.xml.transform`
`interface Source`

- Abstraktion für Datenquelle in XSL-Transformation

```
void setSystemId(String systemId)
```

```
String getSystemId()
```

- Setzen und Zugriff auf ID der Datenquelle

`javax.xml.transform`
`interface Result`

- Abstraktion für Datensenke in XSL-Transformation

```
void setSystemId(String systemId)
```

```
String getSystemId()
```

- Setzen und Zugriff auf ID der Datensenke

`javax.xml.transform`
`interface Result`

- Abstraktion für Datensenke in XSL-Transformation

```
void setSystemId(String systemId)
```

```
String getSystemId()
```

- Setzen und Zugriff auf ID der Datensenke



`javax.xml.transform`

class TransformerFactory

```
static TransformerFactory newInstance()  
    throws TransformerFactoryConfigurationException
```

- Erzeugen einer TransformerFactory

```
abstract Transformer newTransformer(Source source)  
    throws TransformerConfigurationException
```

- Erzeugen eines Transformers mit StyleSheet von source

`javax.xml.transform`

class Transformer

```
abstract void transform(Source xmlSource, Result outputTarget)  
    throws TransformerException
```

- Transformation von xmlSource zu outputTarget

```
abstract void setOutputProperty(String name, String value) throws  
    IllegalArgumentException
```

- Setzen eines OutputProperties für diesen Transformer

```
abstract void setParameter(String name, Object value)
```

- Setzen eines Parameters für diesen Transformer

```
abstract void setErrorListener(ErrorListener listener)  
    throws IllegalArgumentException
```

- Setzen eines ErrorListeners für diesen Transformer

