Java und XML

Java und XML

- Einführung in XML
  - Merkmale von XML, DTDs, ...
- SAX
  - Grundlagen, Verwendung in Java, ...
- DOM
  - Grundlagen, Verwendung in Java, ...

tiscover
THE TRAVEL NETWORK

Motivation für XML (1/4)

Von HTML zu XML (1/2)

- HTML (HyperText Markup Language) ist die "Lingua Franca" zur Beschreibung von **Hypertextdokumenten** im WWW
- Basiskonzept: **Auszeichnungen** ("**Markup**") in Form von "**Tags**"
- Vom W3C (World Wide Web Consortium) standardisiert
- Weist eine Reihe von **Einschränkungen** auf:
  - beschränkte Anzahl vordefinierter Tags
    - daher ständig Erweiterungen um (proprietäre) Tags
  - Tags beschreiben vorwiegend Layout-Aspekte
    - daher wird die Suche im Web erschwert

Motivation für XML (2/4)

Von HTML zu XML (2/2)

HTML beschreibt
Layout des Dokumentinhalts

```
<h1>HandyKatalog</h1>
<h2>Nokia 8210</h2>
<table border="1">
 <tr>
    <td>Batterie</td>
    <td>900mAh</td>
 </tr>
 <tr>
  <td>Gewicht</td><td>141g</td>
 </tr> ...
</table>
```

XML beschreibt
Struktur u. **Semantik** d. Dokumentinhalts

```
<HandyKatalog>
 <Hersteller name="Nokia">
  <Modell name="8210">
 <Batterie>900mAh</Batterie>
   <Gewicht>141g</Gewicht>
   ...
  </Modell>
 </Hersteller>
</HandyKatalog>
```

tiscover
THE TRAVEL NETWORK

Motivation für XML (3/4)

Merkmale von XML

- **Layoutunabhängigkeit**
  - Trennung der Struktur und Semantik des Inhalts von dessen Layout
- **Erweiterbarkeit** (Metasprache)
  - Tags und Attribute können beliebig neu definiert und benannt werden
- **Strukturierbarkeit**
  - Tags können beliebig komplex geschachtelt werden
- **Semistrukturiertheit**
  - Inhalt kann auch nicht-strukturierte Teile aufweisen
- **Selbstbeschreibend**
  - Tags im XML Dokument beschreiben Struktur und Semantik des Inhalts
  - ... für den Menschen: einfach zu lesen u. zu erstellen
  - ... für die Maschine: einfach zu generieren u. zu parsen
- **Validierbarkeit**
  - XML Dokumente können optional ein Schema, d.h. eine formale Beschreibung ihres Vokabulars und ihrer Grammatik aufweisen (Document Type Definition - DTD oder XML Schema) und gegenüber dieser validiert werden

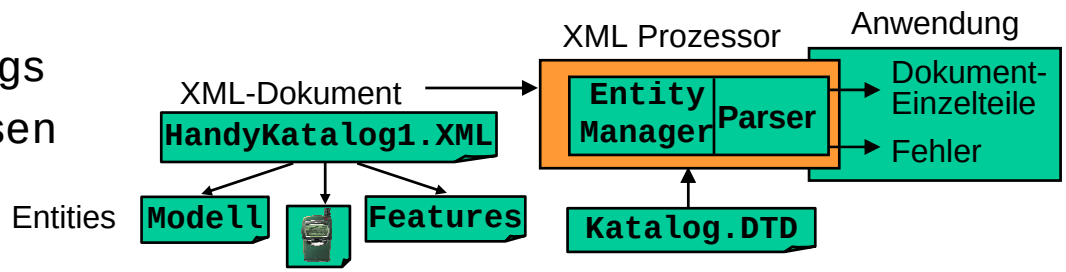
Motivation für XML (4/4)

Eigenschaften von XML

- **Wohlgeformtheit (well-formedness)**
  best. syntaktische Eigenschaften, z.B.:
  - Mindestens 1 Tag pro Dokument
  - Exakt 1 Tag als Wurzel
  - Keine Überlappungen bei Tags
  - Jedes Tag muss abgeschlossen werden
  - ....

Gültigkeit (validity)

XML Dokument ist wohlgeformt und entspricht einem Schema



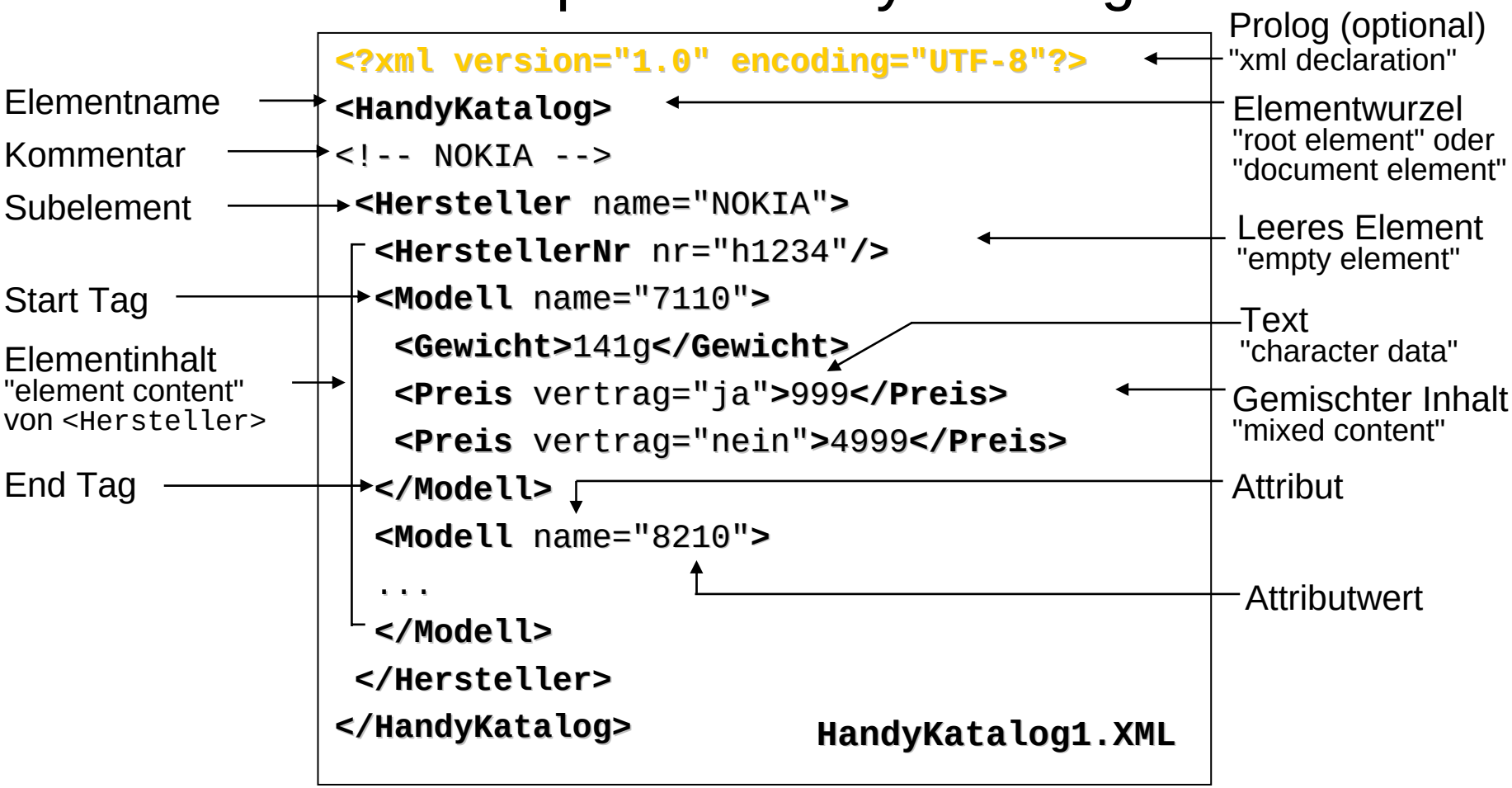
XML-Prozessoren lesen XML-Dokumente ein und überprüfen
entweder nur deren Wohlgeformtheit (**Nicht-validierende Prozessoren**)
oder auch deren Validität (**validierende Prozessoren**)

Können **in Anwendungen** (z.B. Browser) **eingebunden** werden

Zerlegen ein XML-Dokument in seine Einzelteile und erstellen einen **Baum**, durch den die Einzelteile für die Anwendung zugreifbar werden

Sprachkonzepte von XML (1/3)

Beispiel: Handykatalog



tiscover
THE TRAVEL NETWORK

Sprachkonzepte von XML (2/3)

Syntax von Elementen und Attributen

- Element- und Attributnamen müssen gültige "XML Namen" sein:
  - [**letter | _ | :**] [**letter** | '**0..9**' | '**.**' | '**-**' | '_' | '**:**']*
  - "letter" umfassen A-Z, a-z, sowie andere Schriftzeichen wie bspw. ä, ê, γ, ς
  - Verwendung von '**:**' ist Namensräumen vorbehalten
  - keine Längenbeschränkung
  - Case-sensitiv
- **Leere Elemente** können in **Kurzform** angeschrieben werden
  - `<HerstellerNr nr="h1234"></HerstellerNr>` oder
  - `<HerstellerNr nr="h1234"/>`
- **Attributwerte** müssen unter **Anführungszeichen** gesetzt werden
  - `<Modell name='8210'>` oder
  - `<Modell name="8210">`

Sprachkonzepte von XML (3/3)

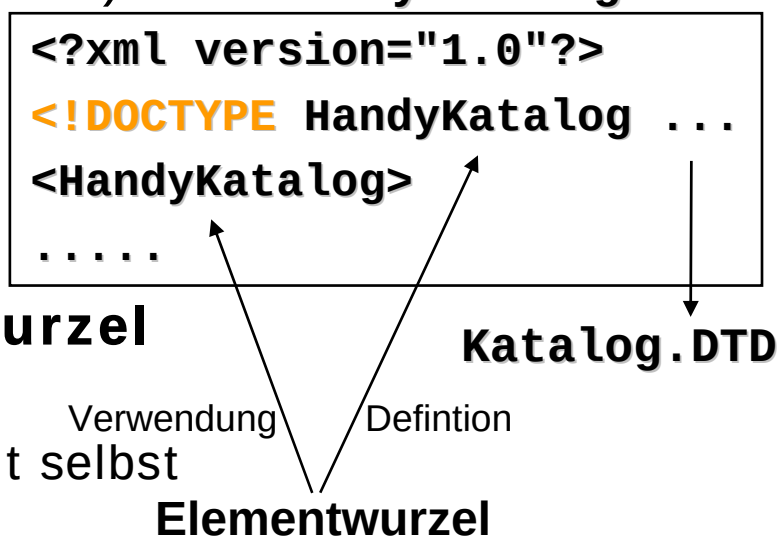
Syntax von Kommentaren

- Können sich über **mehrere Zeilen** erstrecken
  - **zwischen Start Tag** und **End Tag** eines Elements
  - **vor** oder **nach** der **Elementwurzel**
- **Restriktionen**:
  - nicht innerhalb eines Tags erlaubt
  - keine Schachtelung von Kommentaren erlaubt
  - **keine "--"** innerhalb eines Kommentars erlaubt

```
<!--
Ein Kommentar darf
auch Dinge wie <tagnamen>
oder &entitäten; enthalten
--> ...
```

DTD (1/8)
Zweck und Charakteristika

- Eine DTD beschreibt **Vokabular** und **Grammatik** für eine **Menge von XML-Dokumenten**
- Ein XML-Dokument darf **nur eine einzige DTD einbinden** ("document type declaration - **DOCTYPE**")
- Eine DTD muss im XML-Dokument **NACH dem Prolog**, **jedoch VOR der Elementwurzel** eingebunden werden
- Eine DTD **legt nicht die Elementwurzel** eines XML-Dokuments fest
  - dies erfolgt durch das XML-Dokument selbst innerhalb der **DOCTYPE**-Deklaration
  - kann ein beliebiges Element der DTD sein



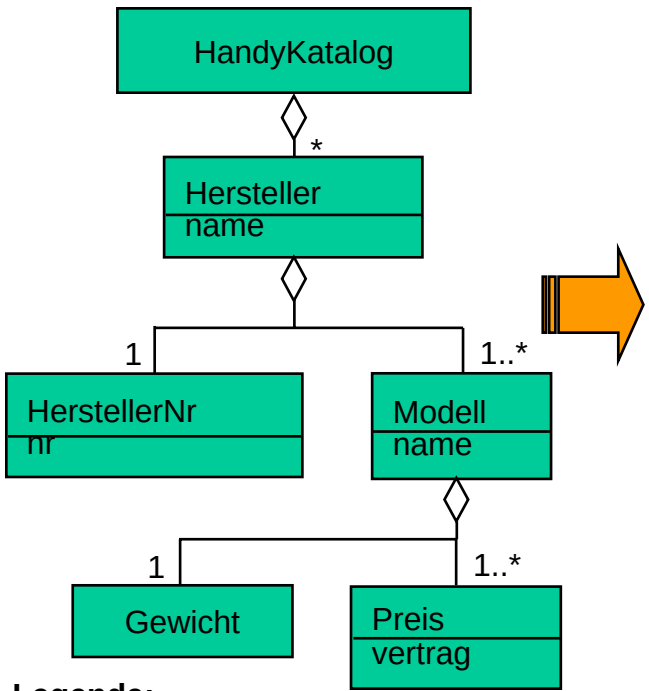
DTD (2/8)
Einbindung in XML Dokumente

- 3 Alternativen zur Einbindung in XML-Dokumente
  - **externe DTD**, d.h. in eigener Datei (`*.dtd`) identifiziert über **URI** ("external subset")
    ```
    <!DOCTYPE HandyKatalog SYSTEM "Katalog.dtd">
    ```
  - **interne DTD**, d.h. im Dokument selbst definiert ("internal subset")
    ```
    <!DOCTYPE HandyKatalog […]>
    ```
  - **externe & interne DTD**, d.h. externe ergänzt interne DTD
- **Exkurs - URL** vs. **URI**:
  - ein UR**L** (Uniform Resource Locator) identifiziert Internet-Ressourcen durch deren Lokation über den Domain Name Service (DNS)
  - ein UR**I** (Uniform Resource Identifier) identifiziert beliebige Ressourcen über deren Namen (z.B. ISBN#) oder andere Attribute der Ressource
  - jeder URL ist ein gültiger URI

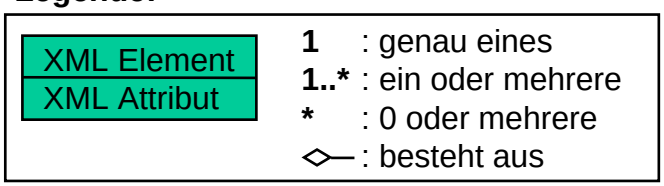
DTD (3/8)

Beispiel: Katalog-DTD

UML Klassendiagramm



Legende:



XML DTD

```
<!-- Katalog DTD Version 1.0 -->
<!ELEMENT HandyKatalog (Hersteller*)>
<!ELEMENT Hersteller (HerstellerNr, Modell+)>
<!ATTLIST Hersteller name CDATA #REQUIRED>
<!ELEMENT HerstellerNr EMPTY>
<!ATTLIST HerstellerNr nr ID #REQUIRED>
<!ELEMENT Modell (Gewicht, Preis+)>
<!ATTLIST Modell name CDATA #REQUIRED>
<!ELEMENT Gewicht (#PCDATA)>
<!ELEMENT Preis (#PCDATA)>
<!ATTLIST Preis vertrag (ja|nein) "nein">
```

DTD (4/8)

Elementdeklaration (1/2)

```
<!ELEMENT Elementname
          (Inhaltsmodell)>
```

- **Sequenz**

```
<!ELEMENT Hersteller (HerstellerNr, Modell+)>
```

- **Alternative**

```
<!ELEMENT Batterie (LiIo | NiMh | NiCd)>
```

- **Kardinalitäten**
  - Optional (Null- oder einmal)

```
<!ELEMENT Modell (Kommentar?)>
```

  - Optional und mehrmals (Null oder mehr)

```
<!ELEMENT HandyKatalog (Hersteller*)>
```

  - Notwendig und wiederholbar (Eins oder mehr)

```
<!ELEMENT Hersteller (Modell+)>
```

  - Inhaltsmodell durch Klammern gruppierbar

```
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
```

DTD (5/8)
Elementdeklaration (2/2)

- **Leeres Element** ("empty element")
  - Element beinhaltet optional Attribute, weder Text noch Subelemente
    ```
    <!ELEMENT HerstellerNr EMPTY>
    ```
- **Elementinhalt** ("element content")
  - Element beinhaltet Subelemente und optional Attribute, keinen Text
    ```
    <!ELEMENT HandyKatalog (Hersteller*)>
    ```
- **Gemischter Inhalt** ("mixed content")
  - Element beinhaltet Text und optional Subelemente oder Attribute
    ```
    <!ELEMENT Preis (#PCDATA)>
    <!ELEMENT Preis (#PCDATA | Kategorie | Rabatt)*>
    ```
- **Element mit beliebigem Inhalt**
  - Nicht näher spezifiziert in DTD
  - Verwendete Elemente müssen allerdings sehr wohl deklariert werden!
    ```
    <!ELEMENT Kommentar ANY>
    ```

DTD (6/8)
Attributdeklaration

```
<!ATTLIST Elementname
              Attributname1 Typ Default
              Attributname2 Typ Default
>                    ...
```

- **Attributnamen** müssen innerhalb eines Elements **eindeutig** sein

- **Defaultspezifikationen**
  - Notwendiger Wert **#REQUIRED**
  - Optionaler Wert **#IMPLIED**
  - Vorgabewert [**#FIXED**] **"wert"**

DTD (7/8)
Attributdeklaration – 10 Typen (1/2)

- **CDATA**
  - Zeichenkette
  - `<!ATTLIST Hersteller name CDATA #REQUIRED>`

- **ID, IDREF(S)**
  - **ID** gewährleistet Eindeutigkeit von Attributwerten innerhalb eines Dokuments
  - pro Element ist nur 1 Attribut vom Typ **ID** erlaubt
  - **IDREF** ist eine Referenz auf ein Attribut vom Typ **ID**
  - referentielle Integrität (ungetypt!) wird durch XML-Prozessor geprüft
  - Werte von **ID**- u. **IDREF(S)**-Attributen müssen gültige XML Namen sein, d.h. dürfen z.B. nicht mit Zahlen beginnen

```
<!ATTLIST beispiel
 identität ID    #IMPLIED
 referenz  IDREF #IMPLIED>
```

DTD (8/8)
Attributdeklaration – 10 Typen (2/2)

- **Aufzählungstyp**
  - Eine vorgegebene Wertemenge bestehend aus XML name tokens
  - `<!ATTLIST Preis vertrag (ja|nein) "nein">`
- **ENTITY, ENTITIES**
  - Attributwert ist Name eines deklarierten nicht-analysierten Entity
  - `<!ATTLIST Image filename ENTITY #REQUIRED>`
- **NMTOKEN(S)**
  - "XML name tokens" sind eine erweiterte Form von XML Namen
  - können zusätzlich mit "0..9 ", ". " und "-" beginnen
  - `<!ATTLIST journal year NMTOKEN #REQUIRED>`
- **NOTATION**
  - Attributwert ist Name einer deklarierten Notation - selten verwendet
  - `<!ATTLIST image type NOTATION (gif | tiff) #REQUIRED>`

tiscover
THE TRAVEL NETWORK

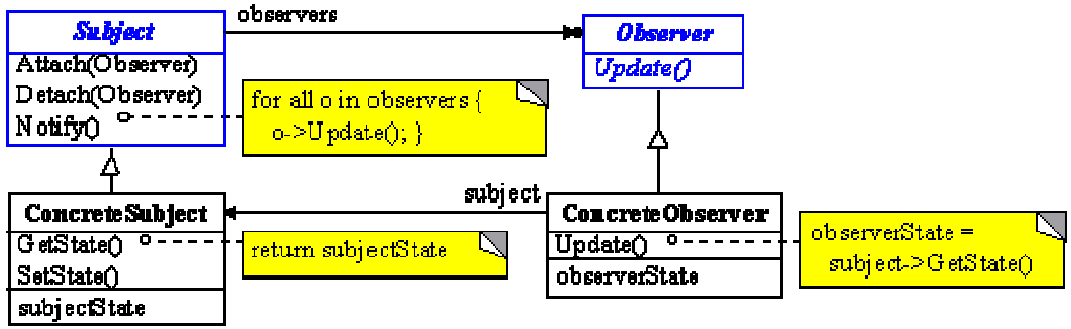
Java und XML

- Einführung in XML
  - Merkmale von XML, DTDs, ...
- SAX
  - Grundlagen, Verwendung in Java, ...
- DOM
  - Grundlagen, Verwendung in Java, ...

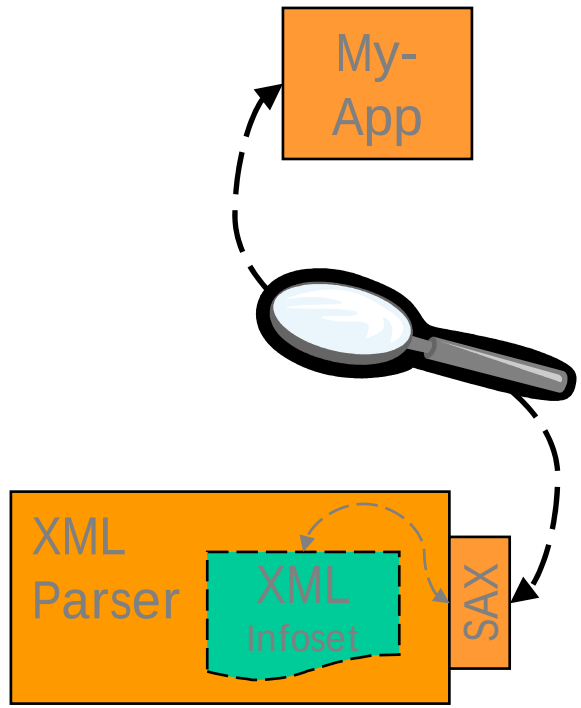
SAX (1/13)
Einführung (1/2)

- SAX ist ein de-facto Standard
  - Keine „formale“ Standardisierung (z.B. von W3C, IETF)
  - Entwickelt von der XML Community (xml-dev@w3.org Mailing Liste)
  - Aktuell: SAX 2.0.1 (Mai 2004)
- SAX 2.0
  - Erweiterung von SAX 1.0 um Namespace Unterstützung
  - Hinzunahme neuer Interfaces, Änderung und Verwerfung alter SAX 1.0 Interfaces (Achtung!)
  - Stellt Adapter Interfaces für die Portierung von SAX 1.0 Code zur Verfügung
  - In neu zu entwickelndem Code SAX 2.0 benutzen!

SAX (2/13)
Einführung (2/2)



- SAX ist ereignis-basiert
  - Callback Interface (vgl. Observer Pattern [Gamm95])
  - Parser liest XML Dokument und DTD, initiiert von Anwendung
  - Parser liefert ein Ereignis je Start Tag, Ende Tag, etc. in Form von Methodenaufrufen
  - Ereignis kann von Anwendung entgegengenommen und verarbeitet werden
  - Ausschließlich serieller Zugriff auf Dokument



SAX (3/13)
Verwendung von SAX in Java (1/8)

- (Schritt 0: XML Dokument erzeugen)
- Schritt 1: Benötigte Java-Packages einbinden:

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

Für IO-Operationen

SAX-Interfaces

Klasse, welche die SAX-Events behandelt

Erzeugt Instanz des SAX-Parsers

tiscover
THE TRAVEL NETWORK

SAX (4/13)
Verwendung von SAX in Java (2/8)

- Schritt 2: Implementierung des ContentHandler-Interface

```
public void characters(char[] ch, int start, int length);
public void startDocument();
public void endDocument();
public void startElement(String namespaceURI, String localName,
  String qName, Attributes atts);
public void endElement(String namespaceURI, String localname,
  String qName);
```

```
…

public class MySAXEcho extends DefaultHandler {

  …
}
```

tiscover
THE TRAVEL NETWORK

SAX (5/13)
Verwendung von SAX in Java (3/8)

- Schritt 3: Parse-Vorgang starten

```
public static void main(String argv[]) {
  if (argv.length != 1) {
    System.err.println("Usage: cmd filename");
    System.exit(1);
  } // Use an instance of ourselves as the SAX event handler
  DefaultHandler handler = new MySAXEcho(); // Use the default parser
  SAXParserFactory factory = SAXParserFactory.newInstance();
  try {
    // Set up output stream
    out = new OutputStreamWriter(System.out, "UTF8");
    // Parse the input SAXParser
    saxParser = factory.newSAXParser();
    saxParser.parse( new File(argv[0]), handler );
  } catch (Throwable t) { t.printStackTrace(); }
  System.exit(0);
}
```

tiscover
THE TRAVEL NETWORK

SAX (6/13)
Verwendung von SAX in Java (4/8)

- Für "echte" XML-Verarbeitung, müssen die einzelnen Methoden des Content-Handler Interface überschrieben werden

```
static private Writer out;

private void emit(String s) throws SAXException {
  try {
    out.write(s);
    out.flush();
  } catch (IOException e) {
    throw new SAXException("I/O error", e);
  }
}
…
```

tiscover
THE TRAVEL NETWORK

SAX (7/13)
Verwendung von SAX in Java (5/8)

- Beispiel (Forts.):

```
…
private void nl() throws SAXException {
  String lineEnd = System.getProperty("line.separator");
  try {
    out.write(lineEnd);
  } catch (IOException e) {
    throw new SAXException("I/O error", e);
  }
}
…
public void startDocument() throws SAXException {
  emit("<?xml version='1.0' encoding='UTF-8'?>");
  nl();
}
…
```

SAX (8/13)
Verwendung von SAX in Java (6/8)

- Beispiel (Forts.):

```
…
public void endDocument() throws SAXException {
  try {
    nl();
    out.flush();
  } catch (IOException e) {
    throw new SAXException("I/O error", e);
  }
}

public void endElement(String namespaceURI,
                       String sName, // simple name
                       String qName // qualified name )
  throws SAXException {
  emit("</"+sName+">");
}
…
```

tiscover
THE TRAVEL NETWORK

SAX (9/13)
Verwendung von SAX in Java (7/8)

- Beispiel (Forts.):

```
public void startElement(String namespaceURI,
                         String sName, // simple name (localName)
                         String qName, // qualified name
                         Attributes attrs) throws SAXException {
  String eName = sName; // element name
  if ("".equals(eName))
    eName = qName;
  // namespaceAware = false
  emit("<"+eName);
  if (attrs != null) {
    for (int i = 0; i < attrs.getLength(); i++) {
      String aName = attrs.getLocalName(i); // Attr name
      if ("".equals(aName)) aName = attrs.getQName(i);
      emit(" ");
      emit(aName+"=\""+attrs.getValue(i)+"\"");
    }
  }
  emit(">");
}
```

tiscover
THE TRAVEL NETWORK

SAX (10/13)
Verwendung von SAX in Java (8/8)

- Beispiel (Forts.):

```
public void characters(char buf[], int offset, int len)
  throws SAXException {

  String s = new String(buf, offset, len);
  emit(s);

}
```

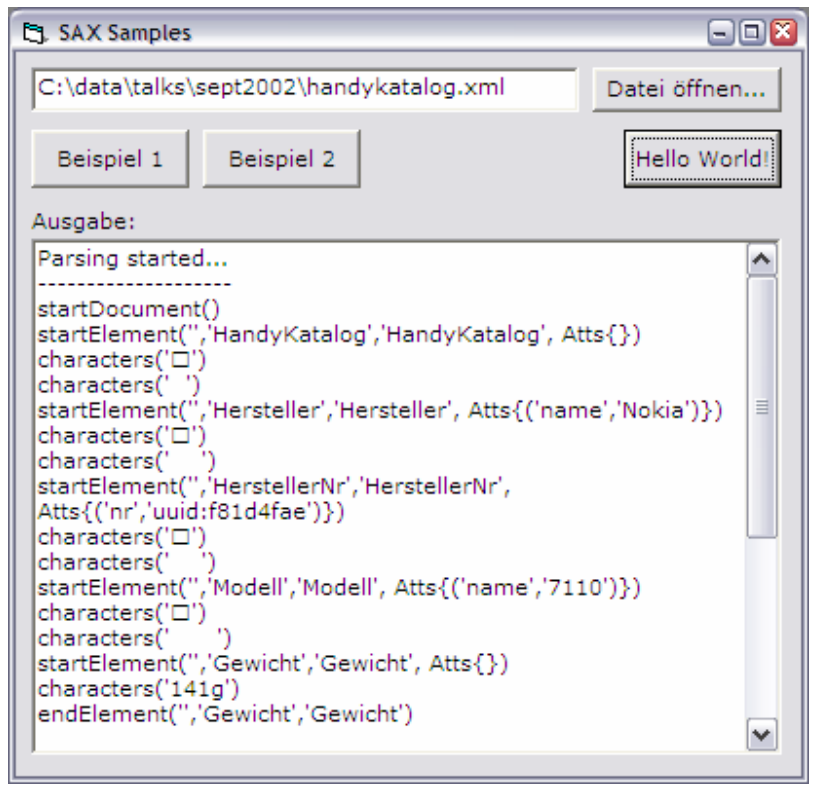
- Die vorgestellten Funktionen sind nur ein Bruchteil der vollständigen Funktionalität von SAX

SAX (11/13)
Beispiel

XML Dokument

```
<HandyKatalog>
  <Hersteller name="Nokia">
    <HerstellerNr nr="uuid:f81d4fae">
    <Modell name="7110">
      <Gewicht>141g</Gewicht>
      <Preis vertrag="ja">999</Preis>
      ...
    </Modell>
  </Hersteller>
</HandyKatalog>
```

SAX Event Stream



SAX (12/13)
Vor- und Nachteile

- Vorteile
  - Sehr gut für Streaming Anwendungen geeignet (d.h. Verarbeiten des Dokuments, bevor es vollständig übertragen ist)
  - Performant
  - Geringe Speicheranforderungen
  - Daher auch für sehr große Dokumente geeignet
- Nachteile
  - Zugriff ausschließlich seriell
  - Applikation muss Puffer-Datenstruktur aufbauen, um Zugriff auf Kontext eines Events zu haben

tiscover
THE TRAVEL NETWORK

SAX (13/13)
Zusammenfassung

- Low-level API
  - Geringes Abstraktionsniveau
  - DOM API benutzt SAX API
- SAX ist geeignet, wenn...
  - die Puffer-Datenstruktur "einfacher" als das Dokument ist
  - es reicht, das Dokument einmal zu durchlaufen
  - man nur einige Daten braucht
  - der Kontext der benötigten Daten nicht von besonderem Interesse ist
  - Speicher-Restriktionen bestehen
  - Schnelle Verarbeitung für grosse Dokumente

Java und XML

- Einführung in XML
  - Merkmale von XML, DTDs, ...
- SAX
  - Grundlagen, Verwendung in Java, ...
- DOM
  - Grundlagen, Verwendung in Java, ...

DOM (1/18)
Charakteristika (1/2)

- DOM = **D**ocument **O**bject **M**odel
- Ist plattform- und programmiersprachen-unabhängig
- Definiert die **logische Struktur eines Dokumentes** in Form eines **Baumes**, bestehend aus Knoten und Kanten
- Ermöglicht das Lesen, sowie die dynamische Änderung von Inhalt und Struktur eines XML-Dokuments
- Entstand aus dem Grundgedanken, die Manipulation von HTML mittels JavaScript browserunabhängig zu machen
- Ist ein Standard des W3C *)

DOM (2/18)
Unterschiede zwischen DOM und SAX

- DOM ist ***dokumenten***-orientiert
  - Das **gesamte** Dokument wird in den Hauptspeicher geladen
  - Dort kann traversiert bzw. bearbeitet werden
- SAX ist ***ereignis***-orientiert
  - Das Dokument muss **nicht** als Ganzes in den Hauptspeicher geladen werden ⇨ wesentlich schneller und besser geeignet für große Dokumente
  - **Direkte Manipulation** von XML Dokumenten **nicht möglich**

DOM (3/18)

Baumstruktur (1/3)

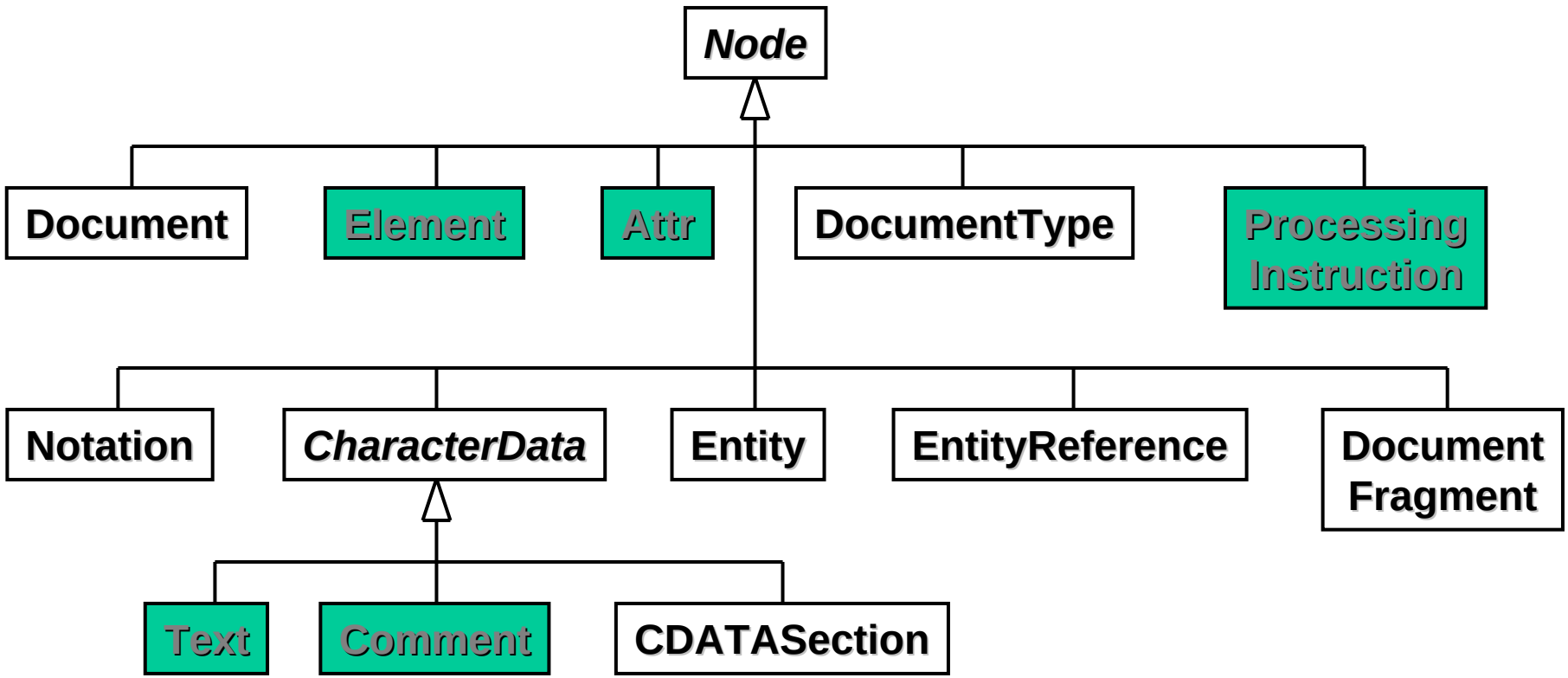
- XML Dokument wird als **Baumstruktur** dargestellt
- Dabei wird **jede Komponente** als **Knoten** repräsentiert

XML / HTML Komponente	In DOM repräsentiert als
Tag (z.B.: ***<name>***)	Elementknoten
Attribut (z.B.: **<name *persNr="1234"*>**	Attributknoten
Text (z.B.: ***Max Mustermann***)	Textknoten

Eine Liste sämtlicher Transformationsregeln (über 40) ist zu finden unter
http://www.w3c.org/TR/DOM-Level-2-Core/core.html

DOM (4/18)

Baumstruktur (2/3)

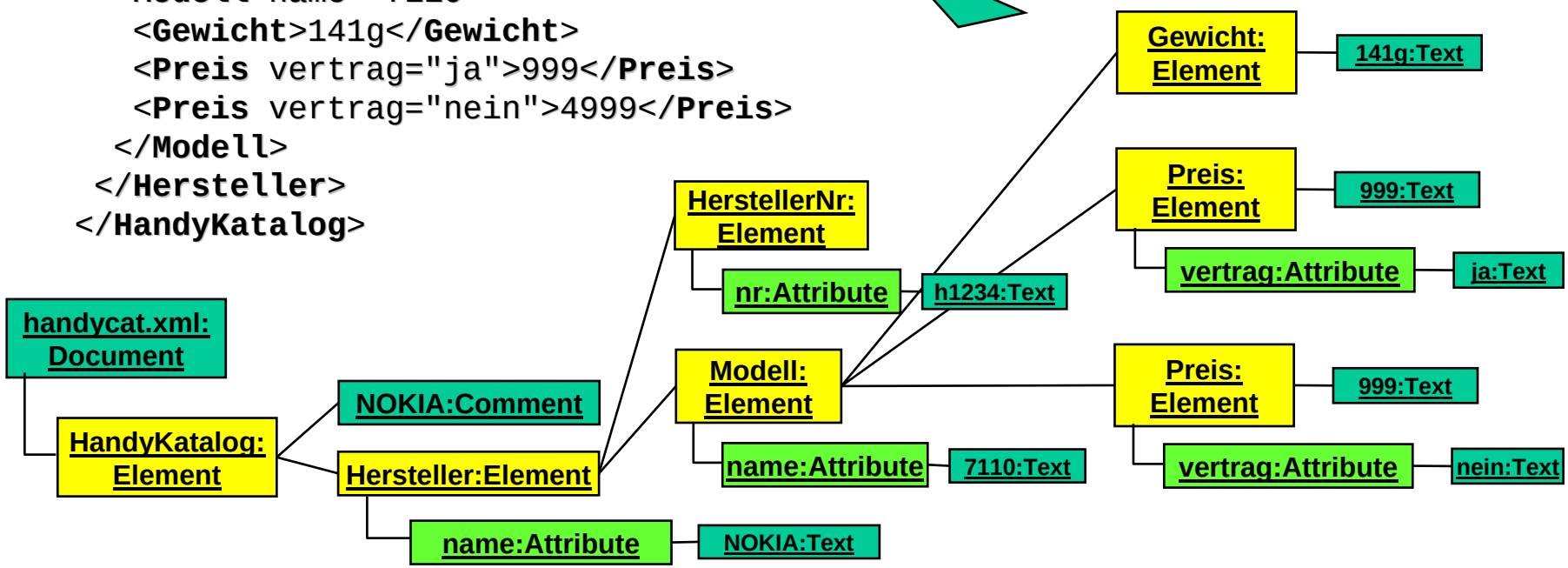


Kommen auch im XPath Datenmodell vor

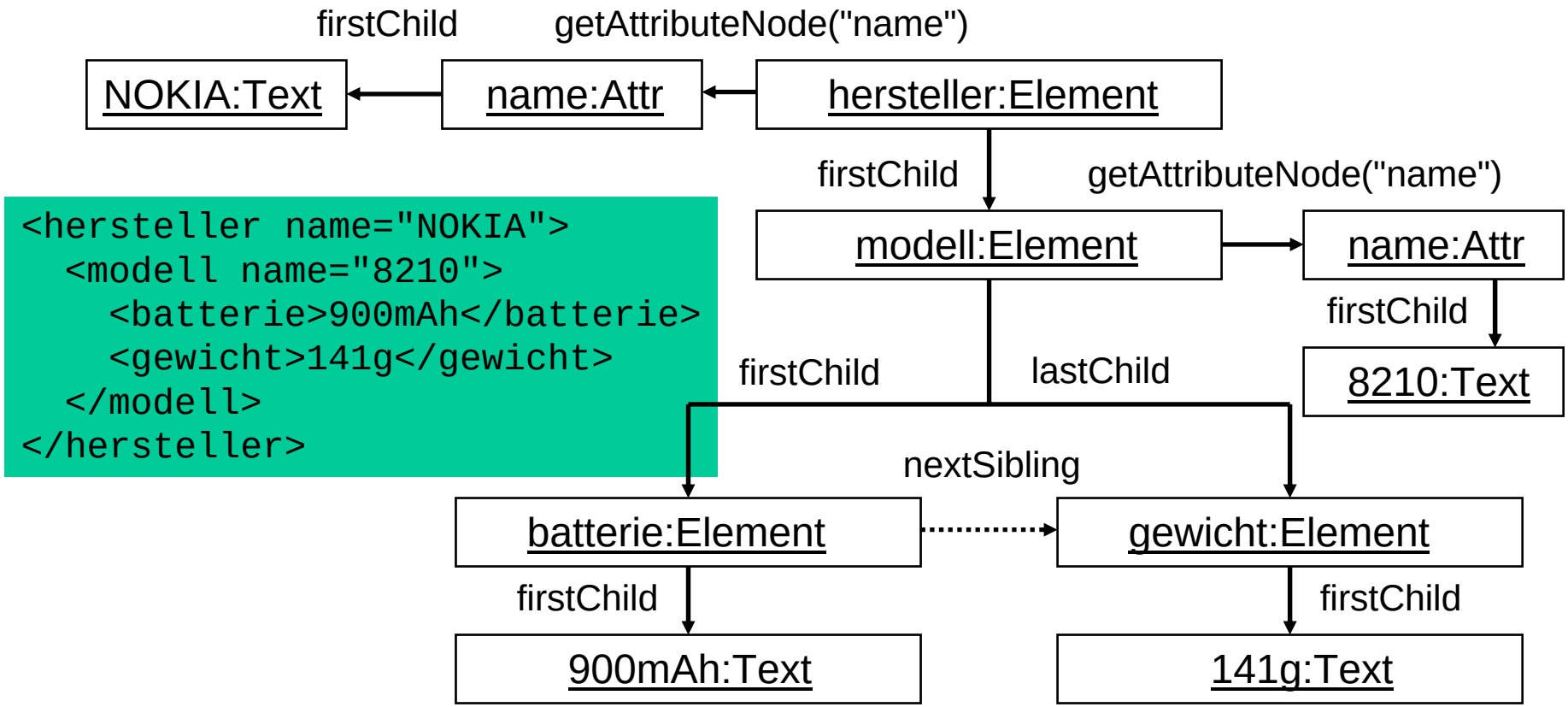
DOM (5/18)

Baumstruktur (3/3)

```
<HandyKatalog>
<!-- NOKIA -->
 <Hersteller name="NOKIA">
  <HerstellerNr nr="h1234"/>
  <Modell name="7110">
   <Gewicht>141g</Gewicht>
   <Preis vertrag="ja">999</Preis>
   <Preis vertrag="nein">4999</Preis>
  </Modell>
 </Hersteller>
</HandyKatalog>
```



DOM (6/18)
Baum traversieren



tiscover
THE TRAVEL NETWORK

DOM (7/18)
Verwendung von DOM in Java (1/9)

- Schritt 1: Basisgerüst des Programs erstellen

```
public class DomEcho {

  public static void main(String argv[]) {

    if (argv.length != 1) {
      System.err.println("Usage: java DomEcho filename");
      System.exit(1);
    }

  } // main

} // DomEcho
```

DOM (8/18)
Verwendung von DOM in Java (2/9)

- Schritt 2: Importieren der nötigen Java-Packages

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import java.io.File;
import java.io.IOException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

tiscover
THE TRAVEL NETWORK

DOM (9/18)
Verwendung von DOM in Java (3/9)

- Schritt 3: Deklarieren des DOM-Objekts

```
public class DomEcho {

  static Document document;

  public static void main(String argv[]) { …
```

- Schritt 4: Deklaration der Fehlerbehandlung

```
public static void main(String argv[]) {

  if (argv.length != 1) { ... }

  try {
    …
```

tiscover
THE TRAVEL NETWORK

DOM (10/18)
Verwendung von DOM in Java (4/9)

- Schritt 4: Deklaration der Fehlerbehandlung (Forts.)

```
  …
  } catch (SAXException sxe) { // Error generated during parsing
    Exception x = sxe;
    if (sxe.getException() != null)
      x = sxe.getException();
    x.printStackTrace();
  } catch (ParserConfigurationException pce) {
    // Parser with specified options can't be built
    pce.printStackTrace();
  } catch (IOException ioe) { // I/O error
    ioe.printStackTrace();
  }

} // main
```

tiscover
THE TRAVEL NETWORK

DOM (11/18)
Verwendung von DOM in Java (5/9)

- Schritt 5: Parse-Vorgang starten

```
public static void main(String argv[]) {

  if (argv.length != 1) { ... }

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

  try {

    // Get a Parser and Parse the File
    DocumentBuilder builder = factory.newDocumentBuilder();

    document = builder.parse( new File(argv[0]) );

  } catch (SAXParseException spe) {
    …
```

tiscover
THE TRAVEL NETWORK

DOM (12/18)
Verwendung von DOM in Java (6/9)

- Wichtigste Klasse: `org.w3c.dom.Node` (Auszug)

```
Node getFirstChild();
Node getLastChild();
NodeList getChildNodes();
NamedNodeMap getAttributes();
String getNodeName();
short getNodeType();
Document getOwnerDocument();
Node getParentNode();
Node getPreviousSibling();
Node getNextSivling();
boolean hasAttributes();
boolean hasChildNodes();
Node insertBefore(Node newChild, Node refChild);
Node removeChild(Node oldChild);
void setNodeValue(String value);
```

DOM (13/18)
Verwendung von DOM in Java (7/9)

- Beispiel:

```
// Zugriff auf Wurzelelement
Element rootElem = document.getDocumentElement();

if (rootElem.hasChildNodes()) {
  NodeList childNodes = rootElem.getChildNodes();
  for (int i=0; i<childNodes.getLength; i++) {
    System.out.println(childNodes.item(i).getNodeName());
    if (childNodes.item(i).getNodeType() == TEXT_NODE)
      System.out.println(childNodes.item(i).getNodeValue();
  }
}

…
```

tiscover
THE TRAVEL NETWORK

DOM (14/18)
Verwendung von DOM in Java (8/9)

- Neue Elemente erzeugen: Methoden von Document

```
Attr createAttribute(String name);
CDATASection createCDATASection(String data);
Comment createComment(String data);
Element createElement(String tagName);
Text createTextNode(String data);

Element getDocumentElement();
Element getElementById(String elementID);
NodeList getElementsByTagName(String tagName);
```

DOM (15/18)
Verwendung von DOM in Java (9/9)

- Beispiel: Element erzeugen

```
Element newElem = document.createElement("Handy");

Attr newAttribute = document.createAttribute("type");
newAttribute.setValue("Siemens S55");

newElem.setAttributeNode(newAttribute);

rootElem.appendChild(newElem);
```

tiscover
THE TRAVEL NETWORK

DOM (16/18)

Sichern eines DOK-Dokumentes in eine Datei (9/9)

```
FileOutputStream os = null;
try {
  os = new FileOutputStream(output);
  // Use a Transformer for output
  TransformerFactory tFactory = TransformerFactory.newInstance();
  Transformer transformer = tFactory.newTransformer();
  transformer.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM, <DTD>);
  DOMSource source = new DOMSource(document);
  StreamResult result = new StreamResult(os);
  transformer.transform(source, result);
} catch (<Exceptions>) {
  e.printStackTrace();
} finally {
  if (os != null) {
    try {
      os.close();
    } catch (IOException e) {}
  }
}
```

DOM (17/18)
Vor- und Nachteile

- Vorteile
  - W3C Standard von vielen Parsern implementiert
  - Einfache Verarbeitung von XML Dokumenten
  - Bietet auch die Möglichkeit XML Dokumente wahlfrei zu manipulieren (vs. SAX → nur seriell)

- Nachteile
  - Hoher Speicherbedarf bei großen XML Dokumenten
  - Hohe Geschwindigkeitsverluste bei großen XML Dokumenten
  - Nicht für Streaming geeignet; Traversieren erst möglich, wenn XML Dokument vollständig geladen
    (MS XML Parser verwendet als Abhilfe standardmäßig asynchrones Parsen)

tiscover
THE TRAVEL NETWORK

DOM (18/18)
Links und Tutorials

- JAXP
  - http://java.sun.com/xml/jaxp/index.jsp
  - http://www-106.ibm.com/developerworks/edu/x-dw-xml-i.html?S_TACT=104AHW02&S_CMP=EDU
- XML Grundlagen
  - http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/index.html
  - http://www-106.ibm.com/developerworks/xml/newto/
- SAX
  - http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/sax/index.html
- DOM
  - http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/dom/index.html
- SAX Benchmarks
  - http://piccolo.sourceforge.net/bench.html
- Parser
  - http://xml.apache.org

tiscover
THE TRAVEL NETWORK

Ende der 7. Übung