Freie Universität Berlin

XML-Parser

Markus Luczak-Rösch
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
markus.luczak-roesch@fu-berlin.de

Was bisher geschah…

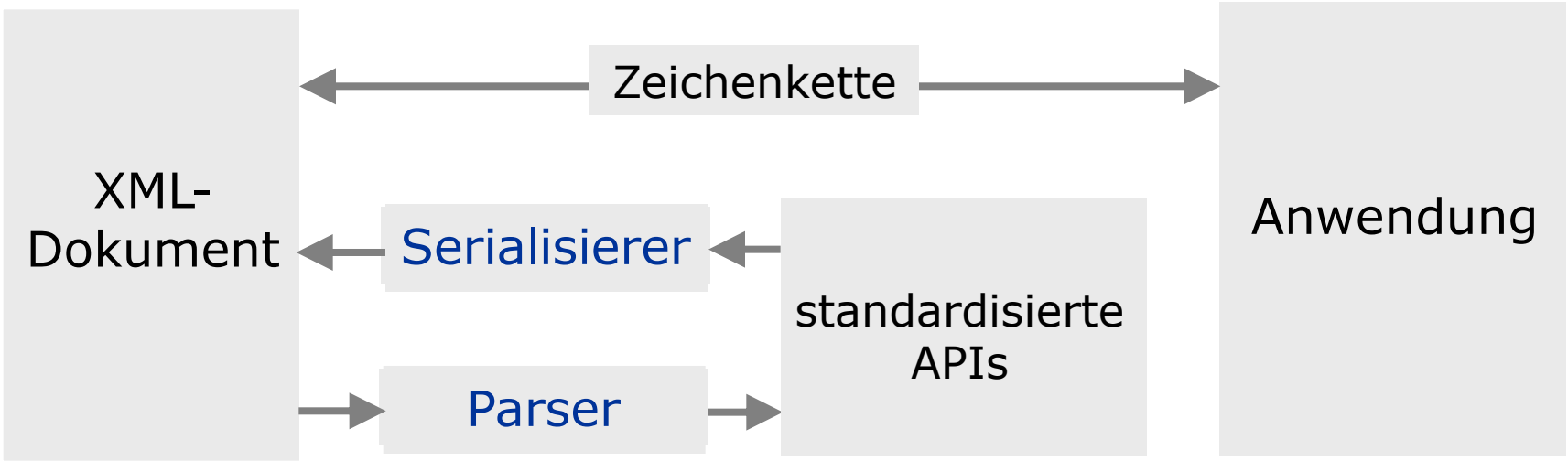
- Syntax wohlgeformter XML-Dokumente
  - XML-Infoset
- Namensräume
- Definition von XML-Sprachen mit DTDs und XML-Schema
  - Vergleich von DTD und XML-Schema
- XML-Schema im Detail
  - Datentypen definieren
  - Element- und Attribut-Deklarationen
  - Typsubstitution
  - Schemaübernahme

Heutige Vorlesung

- Welche XML-Parser gibt es?
- Was sind ihre Vor- und Nachteile?
- Schema-Übersetzer als Alternative

Parsertypen

Grundlegende Architektur



Parser

- analysiert XML-Dokument und erstellt evtl. Parse-Baum mit Tags, Text-Inhalten und Attribut-Wert-Paaren als Knoten

Serialisierer

- Datenstruktur ➔ XML-Dokument

Kategorien von Parser

Validierender vs. nicht-validierender Parser

- Wird die Validität des Dokumentes untersucht?

Pull- vs. Push-Parser

- Wer hat Kontrolle über das Parsen: die Anwendung oder der Parser?

Einschritt- vs. Mehrschritt-Parser

- Wird das XML-Dokument in einem Schritt geparst oder Schritt für Schritt?

- Beachte: Kategorien unabhängig voneinander, können kombiniert werden

Validierender Parser

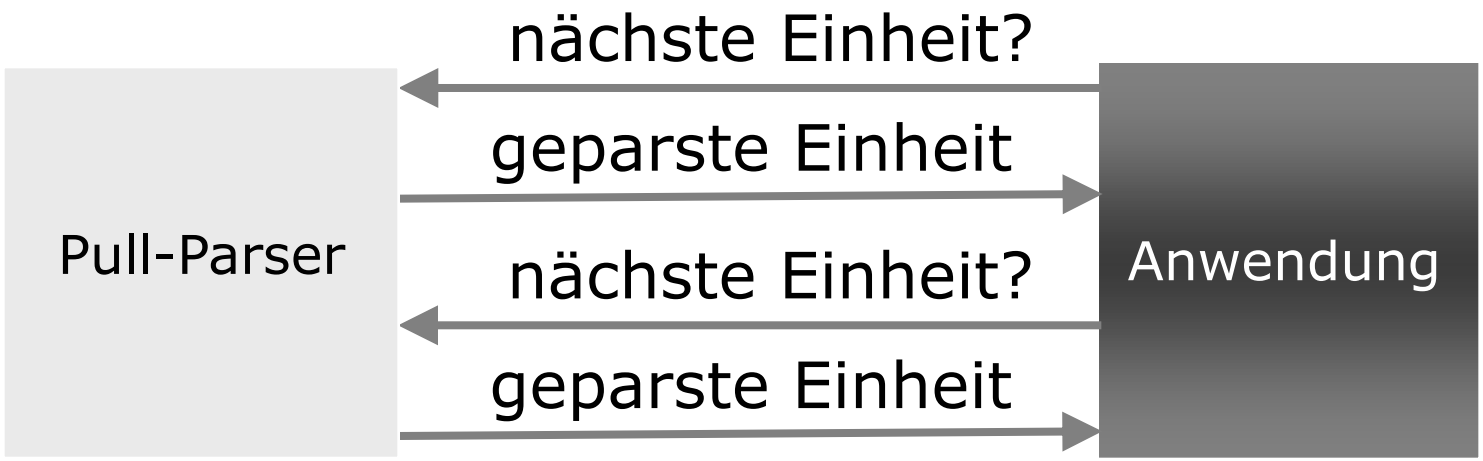
- Ist das XML-Dokument valide?
- DTD oder XML-Schema erforderlich

- Ist das XML-Dokument wohlgeformt?

Nicht-validierender Parser

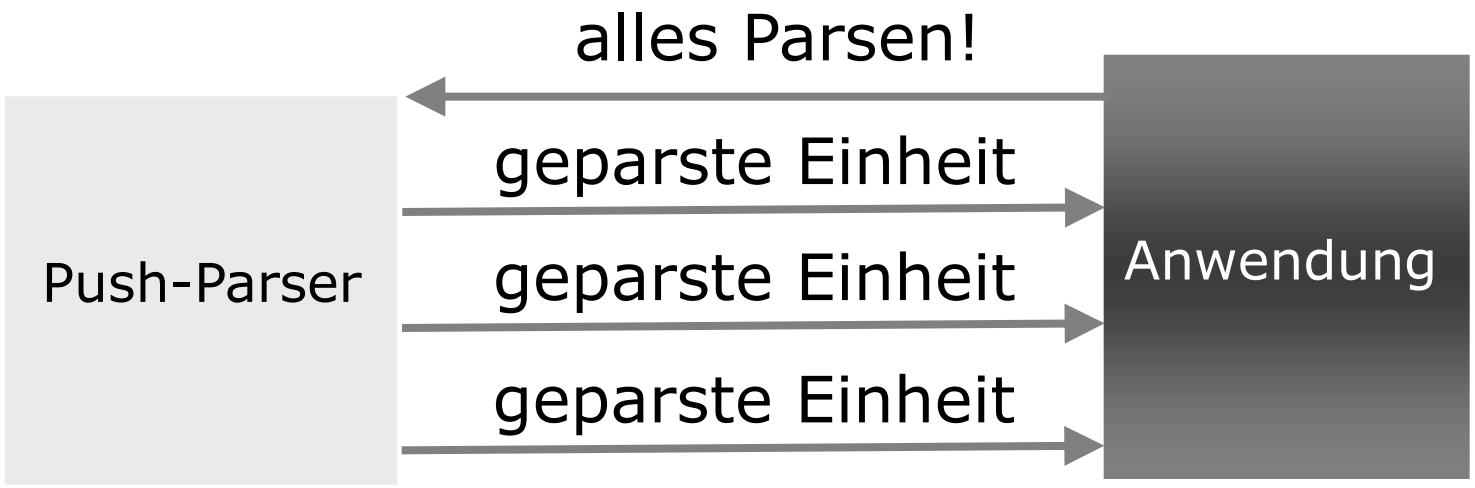
- Ist das XML-Dokument wohlgeformt?

Pull-Parser



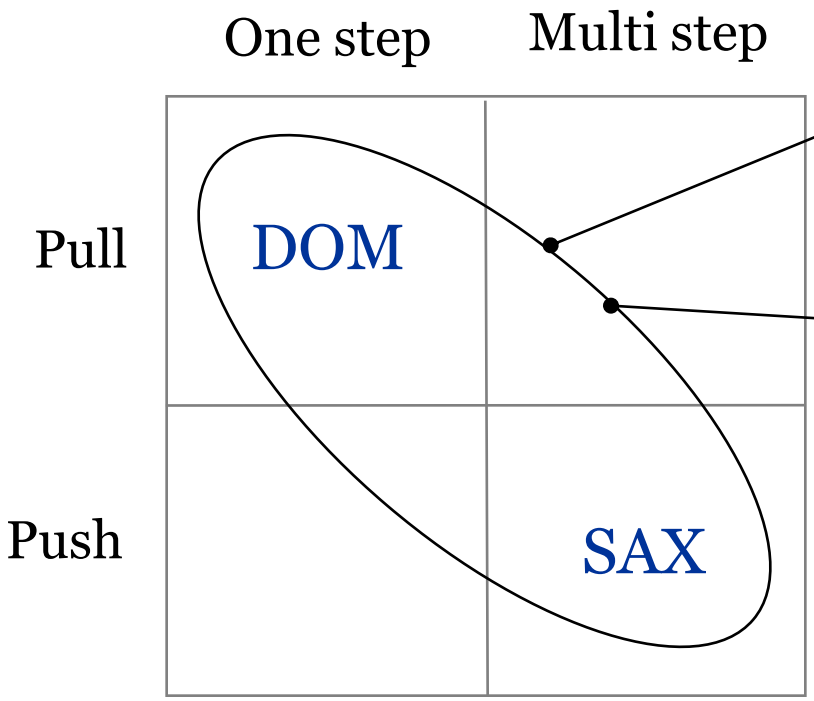
- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: „Pull” aus Perspektive der Anwendung.

Push-Parser



- Parser hat Kontrolle über das Parsen.
- Sobald der Parser eine syntaktische Einheit analysiert hat, übergibt er die entsprechende Analyse.
- Beachte: „Push" aus Perspektive der Anwendung.

XML-Parser



- **StAX**: Streaming API for XML
- **JAXP**: Java API for XML Processing
- JAXP in J2SE 5.0 & Java WSDP 2.0 enthalten

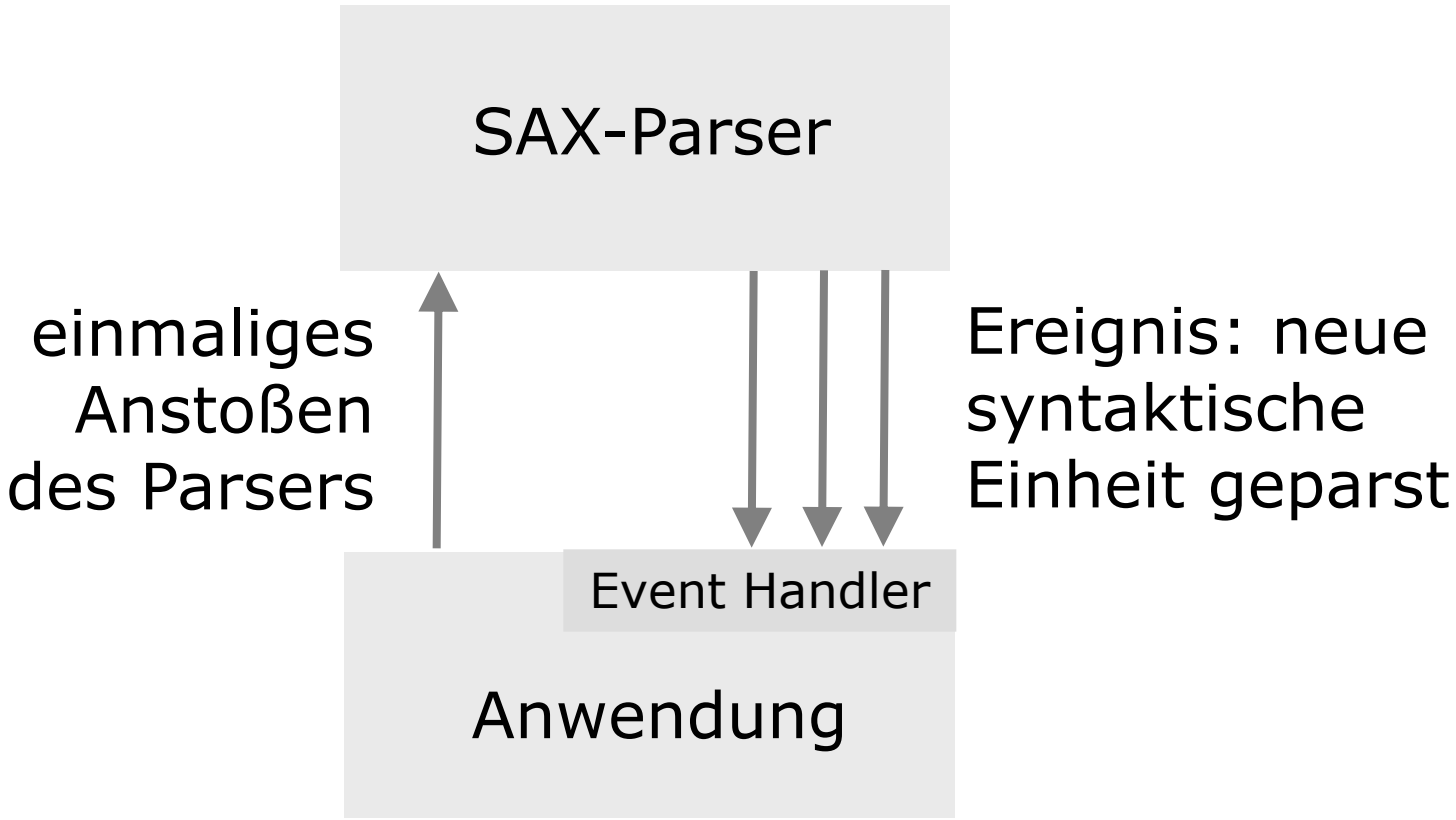
- **DOM**: Document Object Model
- **SAX**: Simple API for XML

SAX-Parser

SAX: Simple API for XML

- **Mehrschritt-Push-Parser für XML**
- kein W3C-Standard, sondern *de facto* Standard
- standardisierte API
- ursprünglich nur Java-API
- inzwischen werden aber auch viele andere Sprachen unterstützt: C, C++, VB, Pascal, Perl
- http://www.saxproject.org/
- auch in MSXML integriert

Ereignisbasiertes Parsen



Beispiel

```
<priceList>
  <coffee>
      <name>
            Mocha Java
      </name>
      <price>
            11.95
      </price>
  </coffee>
</priceList>
```

Parser ruft startElement(…,priceList,…) auf.
Parser ruft startElement(…,coffee,…) auf.
Parser ruft startElement(…,name,…) auf.
Parser ruft characters("Mocha Java",…) auf.
Parser ruft endElement(…,name,..) auf.
Parser ruft startElement(…,price,…) auf.
Parser ruft characters("11.95",…) auf.
Parser ruft endElement(…,price,…) auf.
Parser ruft endElement(…,coffee,…) auf.
Parser ruft endElement(…,priceList,…) auf.

- Ereignisfluss: Sobald Einheit geparst wurde, wird Anwendung benachrichtigt.
- Beachte: Es wird kein Parse-Baum aufgebaut!

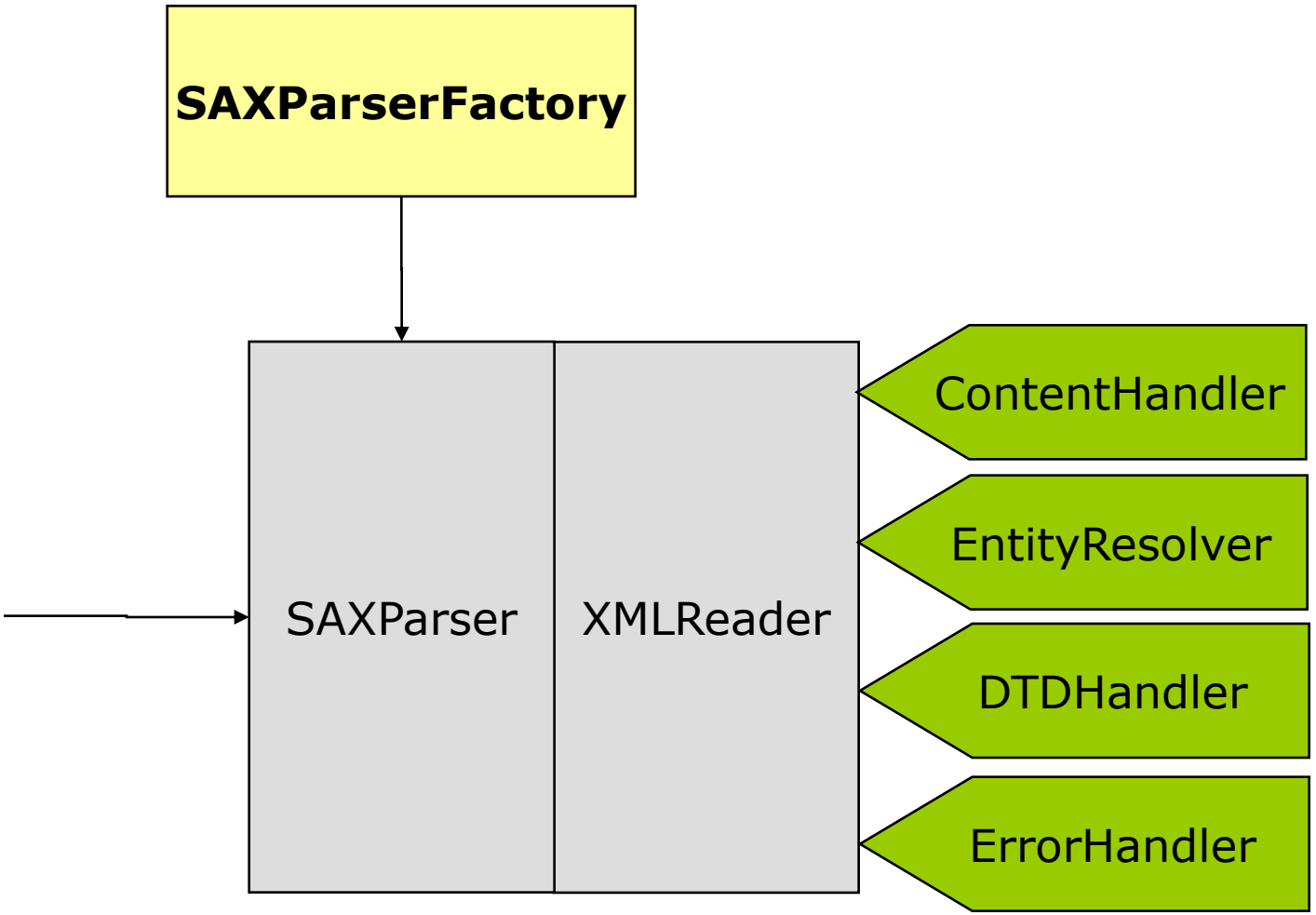
Callback-Methoden

- Methoden des **Event-Handlers** (also der Anwendung), die vom Parser aufgerufen werden
- für jede syntaktische Einheit eigene Callback-Methode, u.a.:
  - startDocument und endDocument
  - startElement und endElement
  - Characters
  - processingInstruction

DefaultHandler

- Standard-Implementierung der Callback-Methoden: tun jeweils nichts!
- können natürlich überschrieben werden

SAXParser & Handler



Quelle: H. Vonhoegen „Einstig in XML: Grundlagen, Praxis, Referenzen“,ISBN 978-3-8362-1074-4, 2007

Einfaches Beispiel: EntityResolver

```
<!ENTITY MyCustomEntity
        PUBLIC "-//Builder.com//TEXT MyCustomEntity//EN"
        "http://www.builder.com/xml/entities/MyCustomEntity">

<!ELEMENT CustomEntity (#PCDATA)>
<!ELEMENT Entity (CustomEntity)>
```

DTD

Standard-Implementierung der Callback-Methoden: tun jeweils nichts!

```
<?xml version="1.0" ?>
<!DOCTYPE entity SYSTEM "entity.dtd">
<Entity>
  <CustomEntity>&MyCustomEntity;</CustomEntity>
 </Entity>
```

XML-Dokument

Quelle: http://articles.techrepublic.com.com/5100-22_11-5075716.html

Custom Entity Resolver

```
import java.io.StringReader;
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;

public class CustomResolver implements EntityResolver {
  public InputSource resolveEntity (String publicId, String systemId) {
    StringReader strReader = new StringReader("This is a custom
 entity");
    if
(systemId.equals("http://www.builder.com/xml/entities/MyCus
tomEntity")) {
      System.out.println("Resolving entity: " + publicId);
      return new InputSource(strReader);
    } else {
      return null;
    }
  }
 }
```

Quelle: http://articles.techrepublic.com.com/5100-22_11-5075716.html

Interface ContentHandler

- **startDocument** – einmalig zu Beginn des Parsens
- **endDocument** – einmalig am Ende des Parsens aufgerufen
- **startPrefixMapping** – wenn eine Präfixbindung für einen Namensraum beginnt
- **endPrefixMapping** – wenn eine Präfixbindung für einen Namensraum endet
- **startElement** – wenn ein Starttag geparst wurde
- **endElement** – wenn ein Endtag geparst wurde
- **characters** – wenn beim Parsen des Elementinhalts Zeichendaten (#PCDATA) angetroffen werden

Callback-Methode startElement

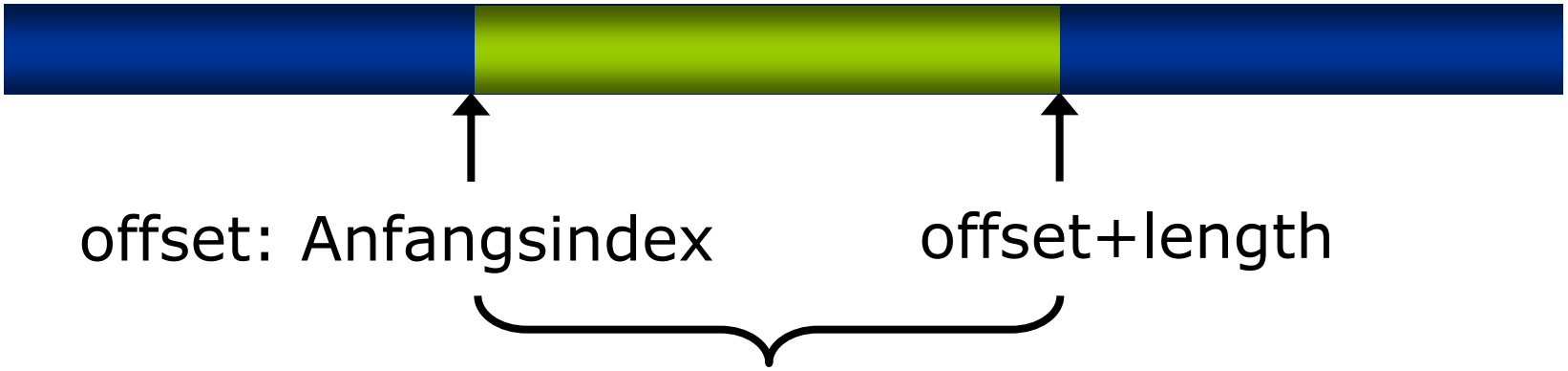
```
public void startElement(java.lang.String uri,
                         java.lang.String localName,
                         java.lang.String qName,
                         Attributes attributes)
        throws SAXException
```

- **uri**: Namensraum-Bezeichner oder leerer String
- **localName**: lokaler Name ohne Präfix oder leerer String
- **qName**: Name mit Präfix oder leerer String
- **attributes**: zu dem Element gehörige Attribute
- Attribute können über ihre Position (Index) oder ihren Namen zugegriffen werden
- **endElement** ähnlich, jedoch ohne attributes

Callback-Methode characters

```
public void characters(char[] buffer,
                       int offset,
                       int length)
        throws SAXException
```

buffer: Liste von Zeichen




```
String s = new String(buffer, offset, length);
```

Beispiel

```
<priceList>
  <coffee>
      <name>
          Mocha Java
      </name>
      <price>
          11.95
      </price>
  </coffee>
</priceList>
```

- Aufgabe: Gib den Preis von Mocha Java aus!
- Hierfür benötigen wir zwei Dinge:
  1. einen SAX-Parser
  2. passende Callback-Methoden

Wie bekomme ich einen SAX-Parser?

Freie Universität Berlin

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

- liefert eine SAXParserFactory

```
SAXParser saxParser = factory.newSAXParser();
```

- liefert einen SAXParser

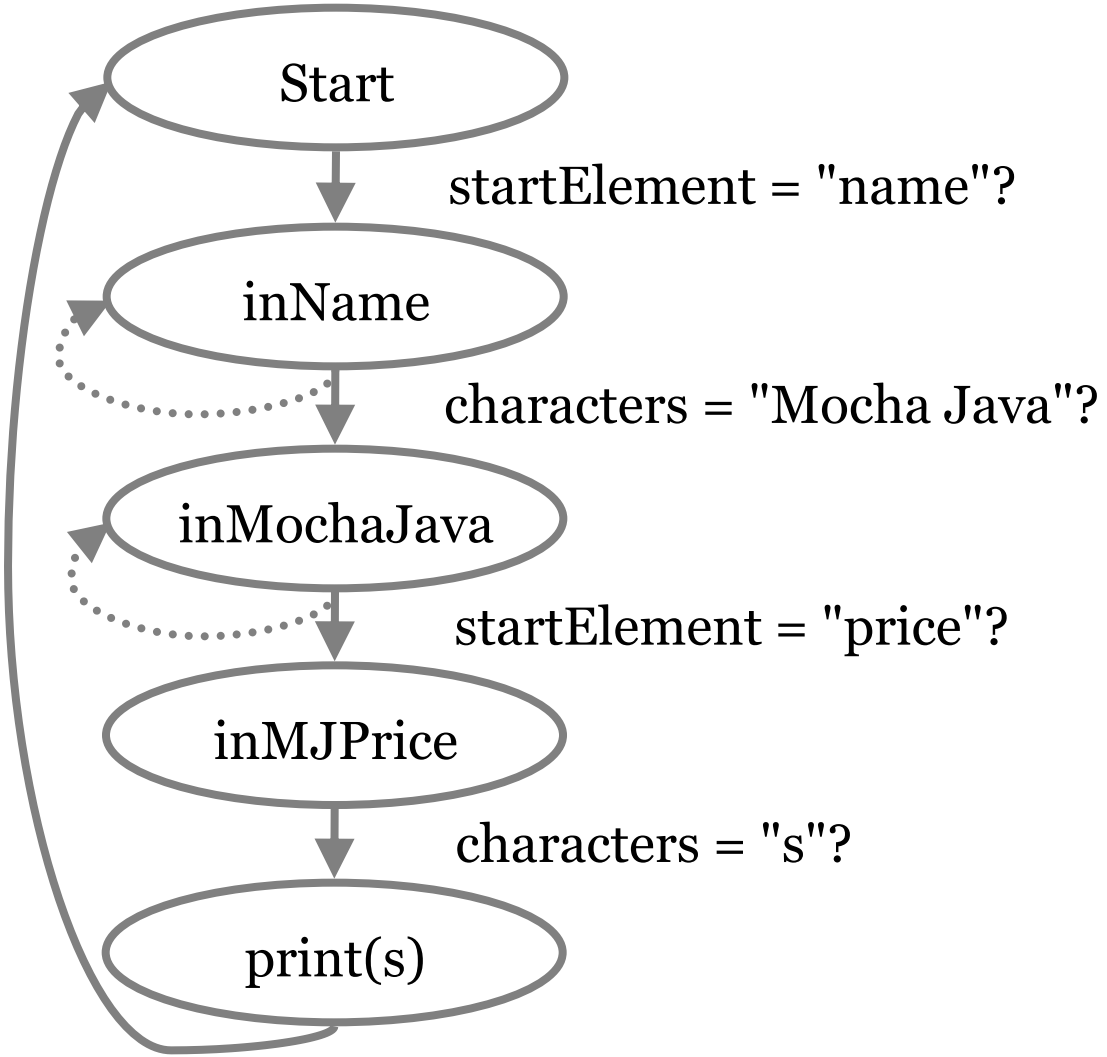
```
saxParser.parse("priceList.xml", handler);
```

- stößt SAX-Parser an
- priceList.xml: zu parsende Datei, kann auch URL oder Stream sein
- handler: Instanz von DefaultHandler, implementiert Callback-Funktionen

Logik der Callback-Methoden


```
<priceList>
  <coffee>
      <name>
          Mocha Java
      </name>
      <price>
          11.95
      </price>
  </coffee>
</priceList>
```

- Zustände als boolesche Variablen



Die Callback-Methoden in Java


```
public void startElement(..., String elementName, ...) {
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") && inMochaJava ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inMJPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMJPrice = false;     }   }
```

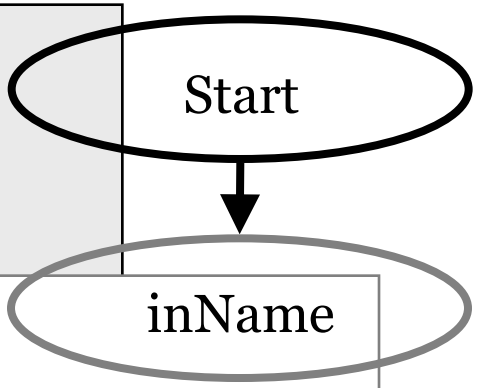
- alle anderen Callback-Methoden aus DefaultHandler = tun nichts

Start: Auf <name> warten

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){
   inName = true; }
else if (elementName.equals("price") && inMochaJava ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inMJPrice) {
   System.out.println("The p
   inMJPrice = false;     }   }
```



Start

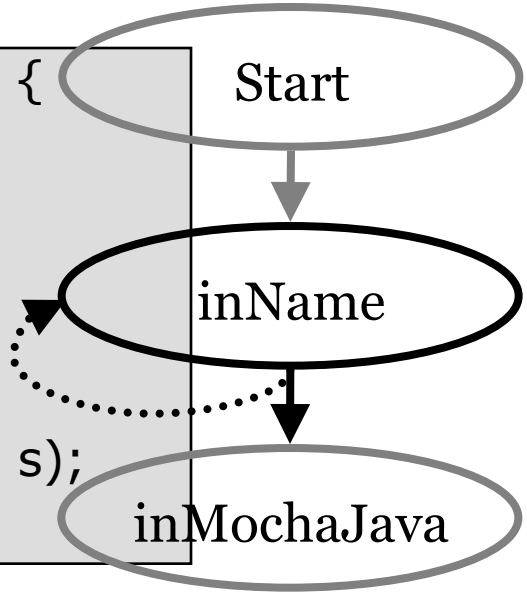
- Anfangszustand
- keine eigene Zustandsvariable
- alle Zustandsvariablen = false

inName: Auf "Mocha Java" warten

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){
   inName = true;   }
else if (elementName.equals("price") && inMochaJava ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

```
<name>Mocha Java</name>
<price>11.95</price>
```

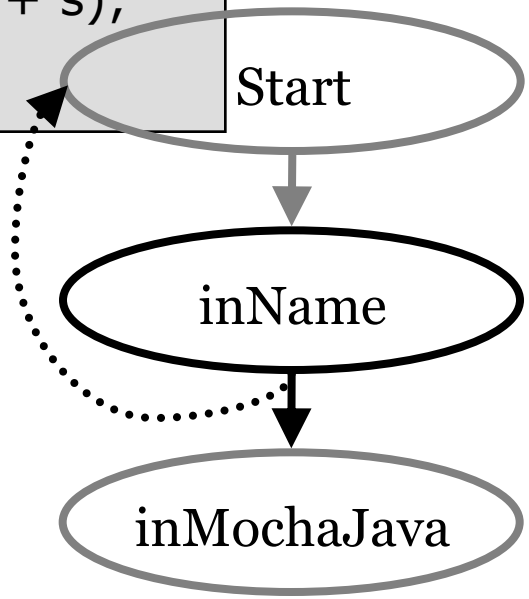
```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inMJPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMJPrice = false;     }   }
```



Eine bessere Alternative

```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName) { if (s.equals("Mocha Java")) {
                    inMochaJava = true;
                   inName = false;  }
                 else inName = false; }
else if (inMJPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMJPrice = false;     }   }
```

```
<name>Mocha Java</name>
<price>11.95</price>
```

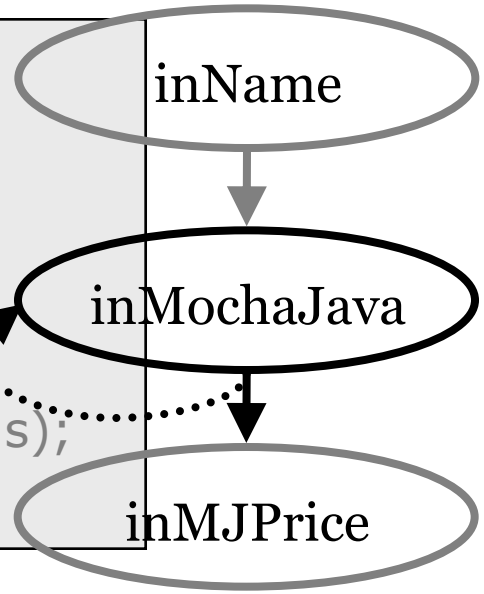


inMochaJava: Auf <price> warten

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") && inMochaJava
   ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inMJPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMJPrice = false;     }   }
```

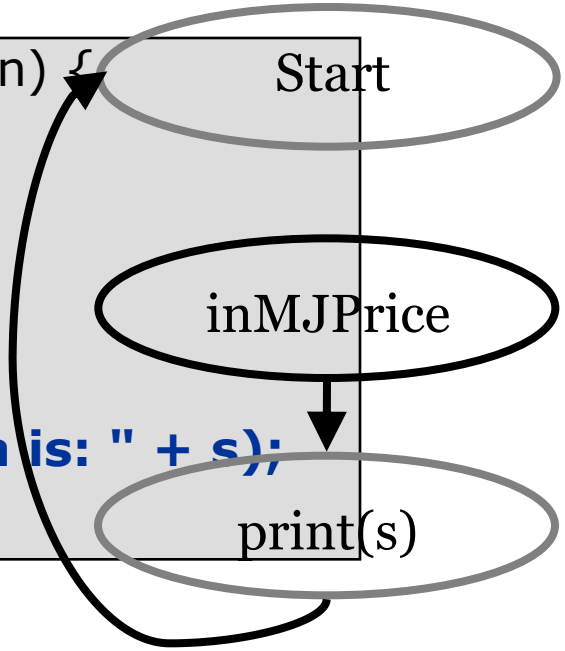


inMJPrice: Preis ausgeben

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") && inMochaJava
   ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

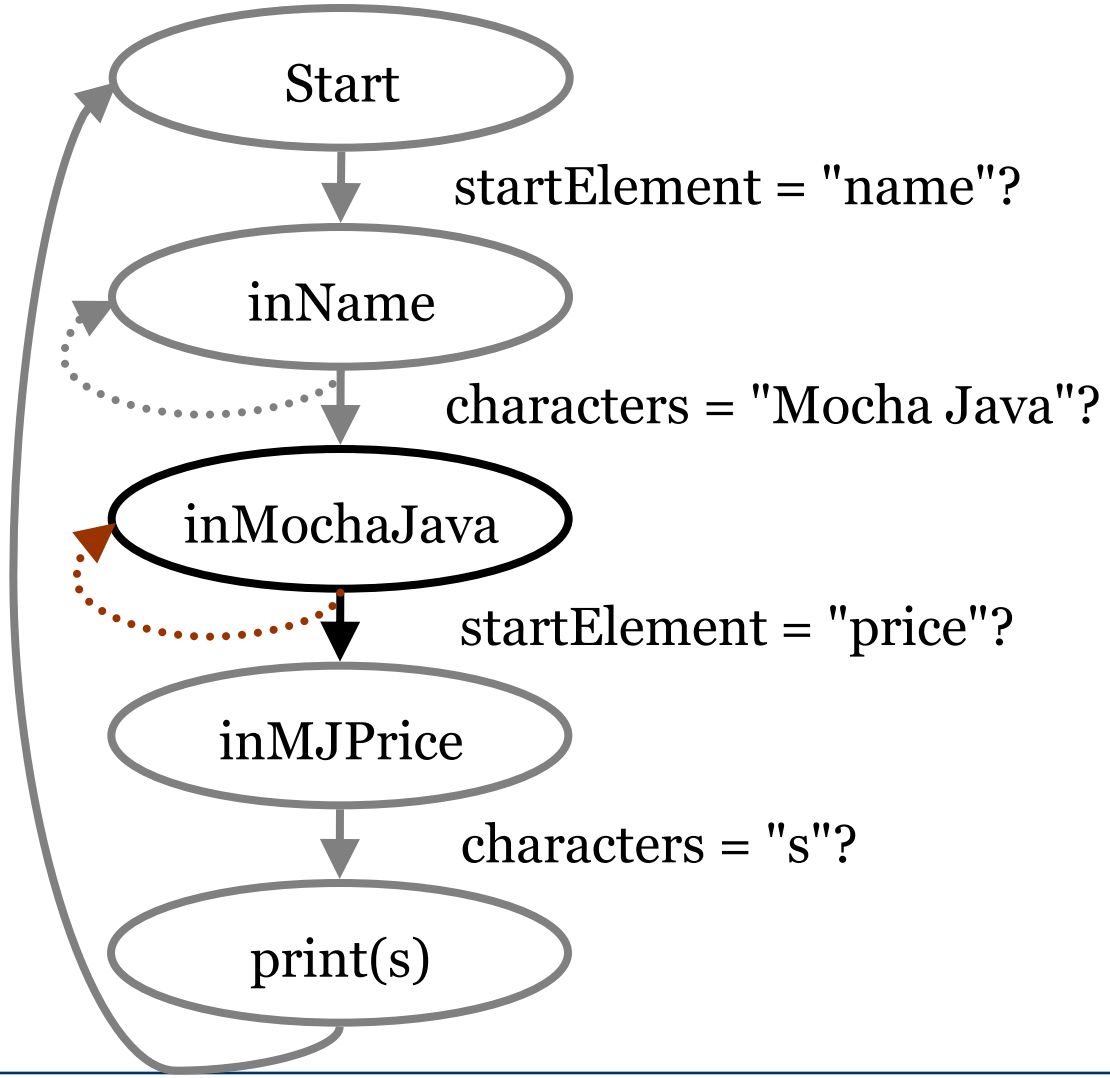
```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inMJPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMJPrice = false;    }   }
```



Fehlerbehandlung

```
<priceList>
  <coffee>
      <name>
          Mocha Java
      </name>
      <name>
          MS Java
      </name>
      <price>
          11.95
      </price>
  </coffee>
</priceList>
```



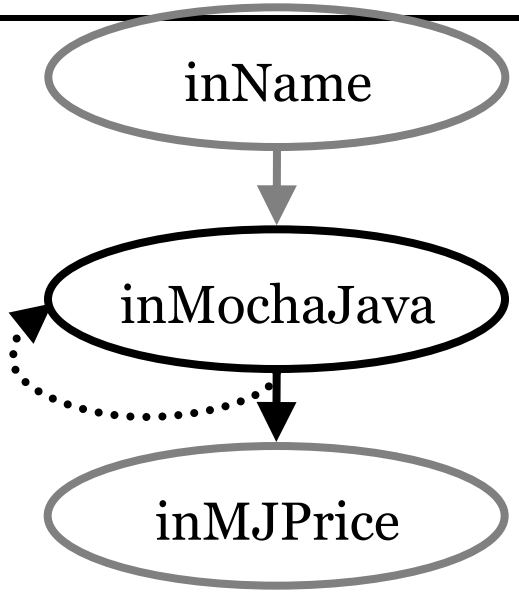
Fehlerbehandlung

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") &&
   inMochaJava ){
   inMJPrice = true;
   inMochaJava = false;   } }
```

```
<name>Mocha Java</name>
<name>MS Java</name>
<price>11.95</price>
```

- inMochaJava erwartet <price>
- kommt stattdessen <name>, wird aktueller Zustand inMochaJava nicht verändert
- kommt danach <price>, wird aktueller Zustand inMJPrice

⇨ Preis von MS Java wird ausgegeben!



Fehlerbehandlung

- SAX-Parser überprüft immer Wohlgeformtheit eines XML-Dokumentes.
- kann aber auch die Zulässigkeit bzgl. einer DTD oder eines Schema überprüfen
  - Schema kann z.B. (name, price)+ verlangen

⇨ Syntax- und Strukturfehler kann bereits der SAX-Parser abfangen

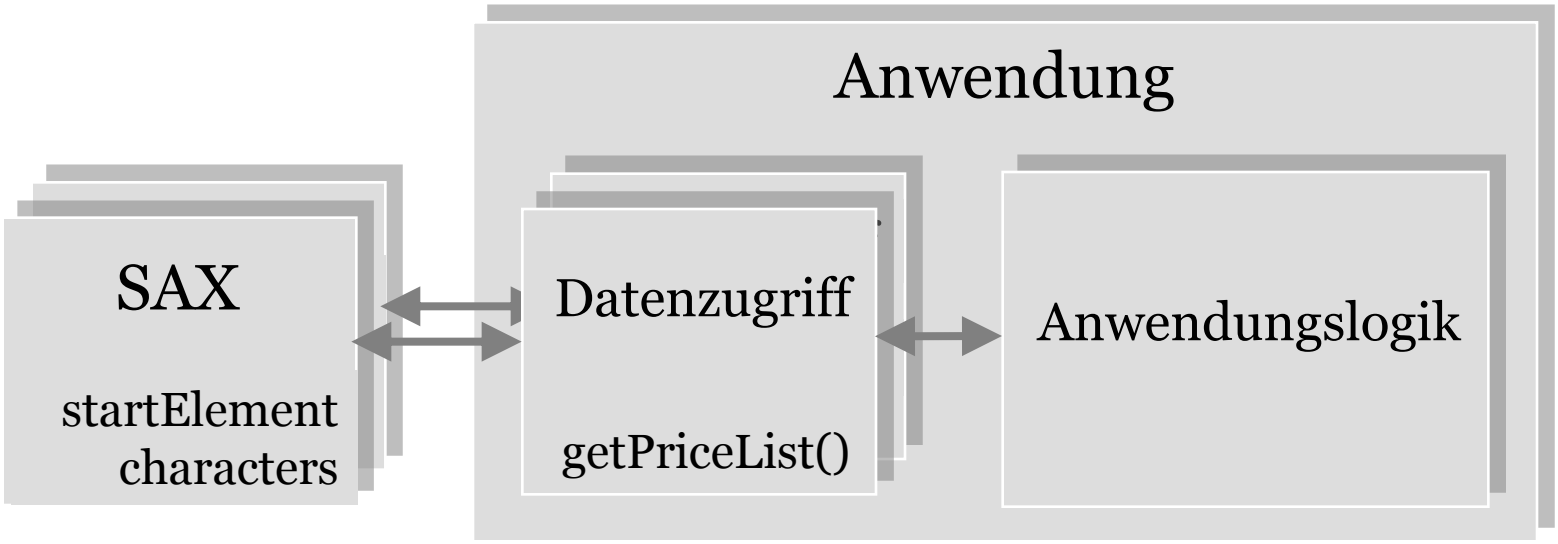
⇨ Callback-Methoden können dann von wohlgeformten und zulässigen Dokument ausgehen.

Vor- und Nachteile von SAX

+ sehr effizient und schnell, auch bei großen XML-Dokumenten
+ relative einfach

- Kontext (Parse-Baum) muss von Anwendung selbst verwaltet werden.
- abstrahiert nicht von XML-Syntax
- nur Parsen möglich, keine Modifikation oder Erstellung von XML-Dokumenten

Zusätzliche Schicht zum Datenzugriff



- Anwendungslogik durch zusätzliche Schicht vom Datenzugriff trennen (nicht nur bei SAX).
- Z.B. könnte getPriceList() eine Liste von Ware-Preis-Paaren liefern.
- sollte nicht nur von SAX-APIs, sondern auch von XML-Syntax abstrahieren

Wohlgeformtheit und Validation in SAX


```
String parserClass = "org.apache.xerces.parsers.SAXParser„;
String validationFeature = "http://xml.org/sax/features/validation";
String schemaFeature = http://apache.org/xml/features/validation/schema;
try {
        String x = args[0]; XMLReader r =
        XMLReaderFactory.createXMLReader(parserClass);
        r.setFeature(validationFeature,true);
        r.setFeature(schemaFeature,true);
        r.setErrorHandler(new MyErrorHandler());
        r.parse(x);
} …
```

Validation Feature

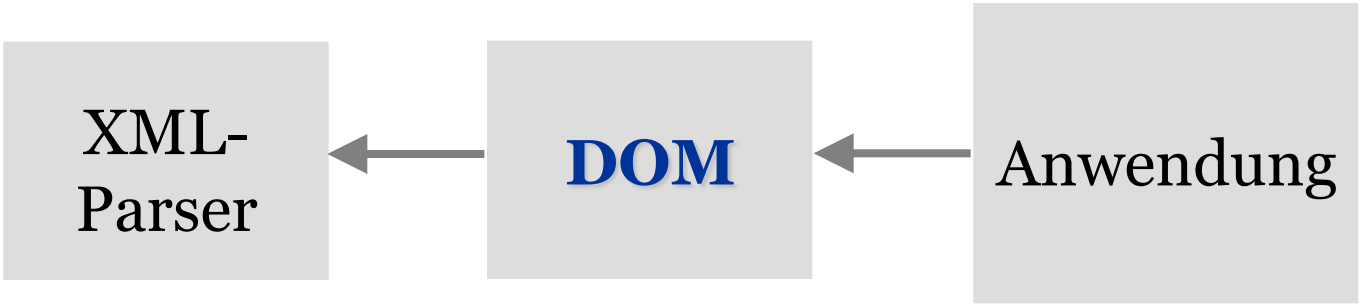
Error Handling für Wohlgeformtheit und Validation

- Fehlererkennung zur Laufzeit des Parsens
  - Tritt ein Fehler erst spät in der Verarbeitung auf, müssen zuvor bereits durch Callback Methoden ausgeführte Aktionen rückgängig gemacht werden!

Beispiel: http://www.herongyang.com/JDK/xsd_validation.html

DOM-Parser

Document Object Model (DOM)



- streng genommen kein Parser, sondern abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- W3C-Standard
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis aber **Einschritt-Pull-Parser**

DOM-Module – Level 1

- DOM-Kern (DOM Core) + Spezialisierungen für HTML und XML;
- Bearbeitung von Dokumentinhalten & Navigation innerhalb der Dokumente
- W3C Recommendation seit 1998
- http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

DOM-Module – Level 2

- Namensräumen + CSS
- Änderungen der Baumstruktur eines Dokuments
- Dynamischer Zugriff und Update von Kontent und Struktur
- unterschiedliche Views von Dokumenten
- W3C Recommendation seit 2000

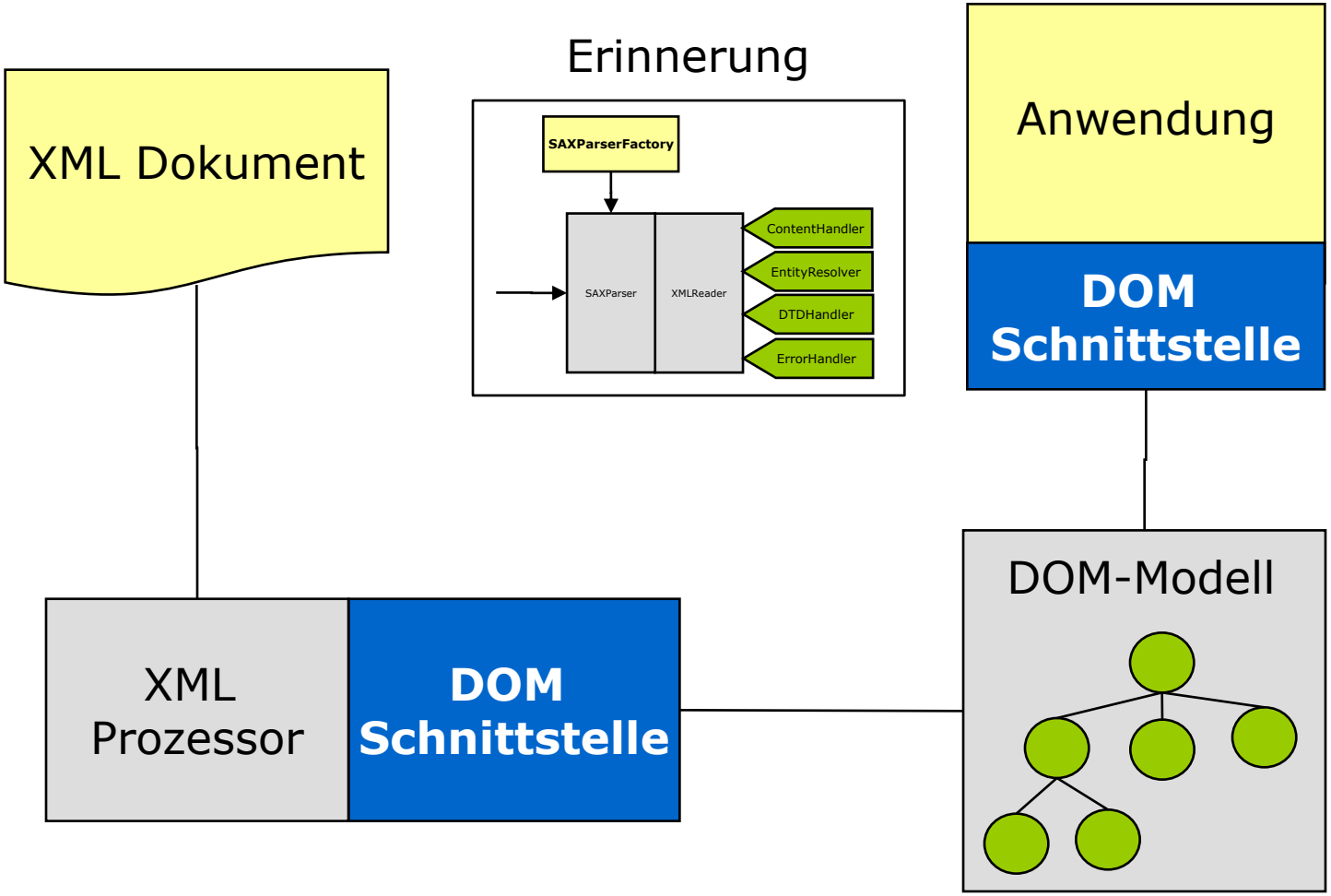
- Core Specification
  - baut auf Level 1 Core
- Views Specification
  - baut auf Level 2 Core
- Events Specification
  - baut auf Level 2 Core & Level 2 Views
- Style Specification
  - baut auf Level 2 Core & Level 2 Views

DOM-Module – Level 3

- Anpassungen an XML Infoset, XML Base und XPath,
- W3C Recommendation seit 2004

- Core Specification
  - baut auf Level 2 Core
- Load and Save Specification
- Validation Specification

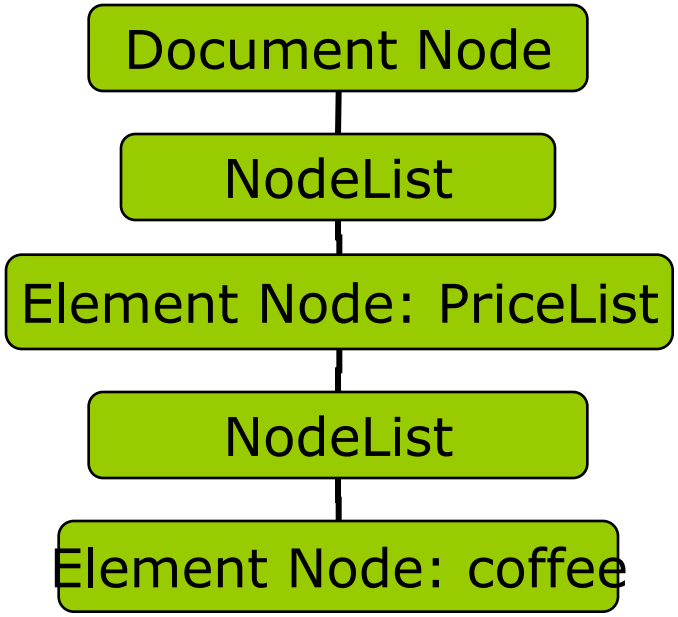
Architektur von DOM-Anwendungen



Quelle: H. Vonhoegen „Einstig in XML: Grundlagen, Praxis, Referenzen“,ISBN 978-3-8362-1074-4, 2007

DOM-Parse-Bäume

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

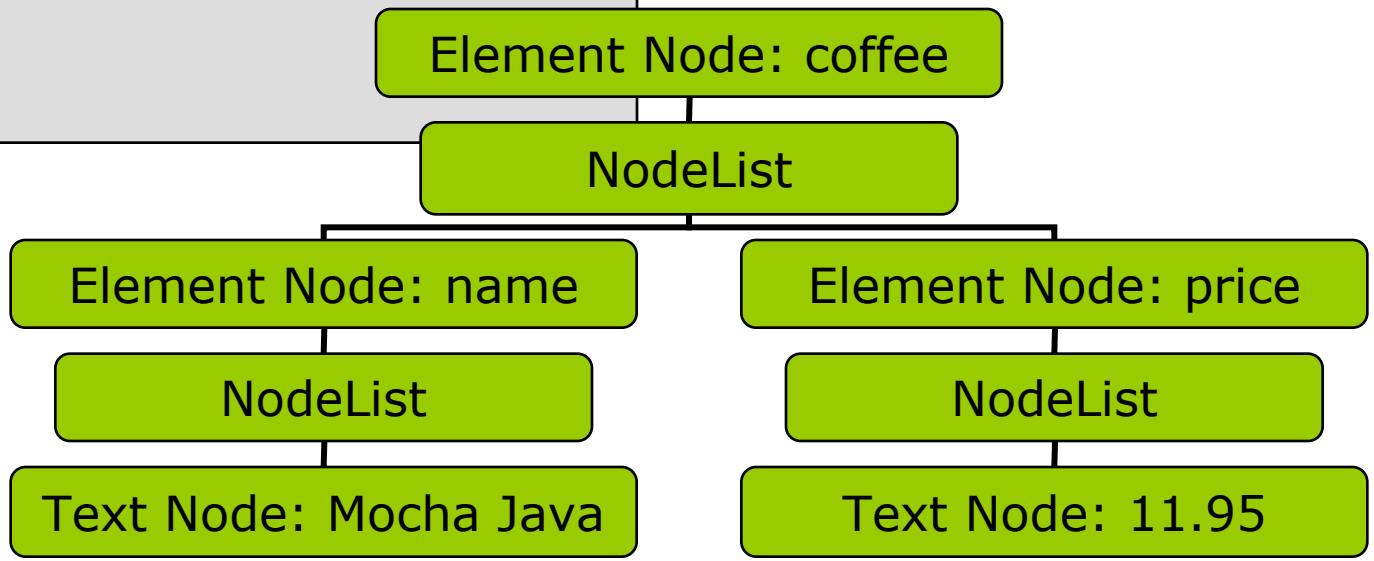


- Beachte: Dokument-Wurzel (Document Node) ≠ priceList
- Document Node: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer NodeList als Kind

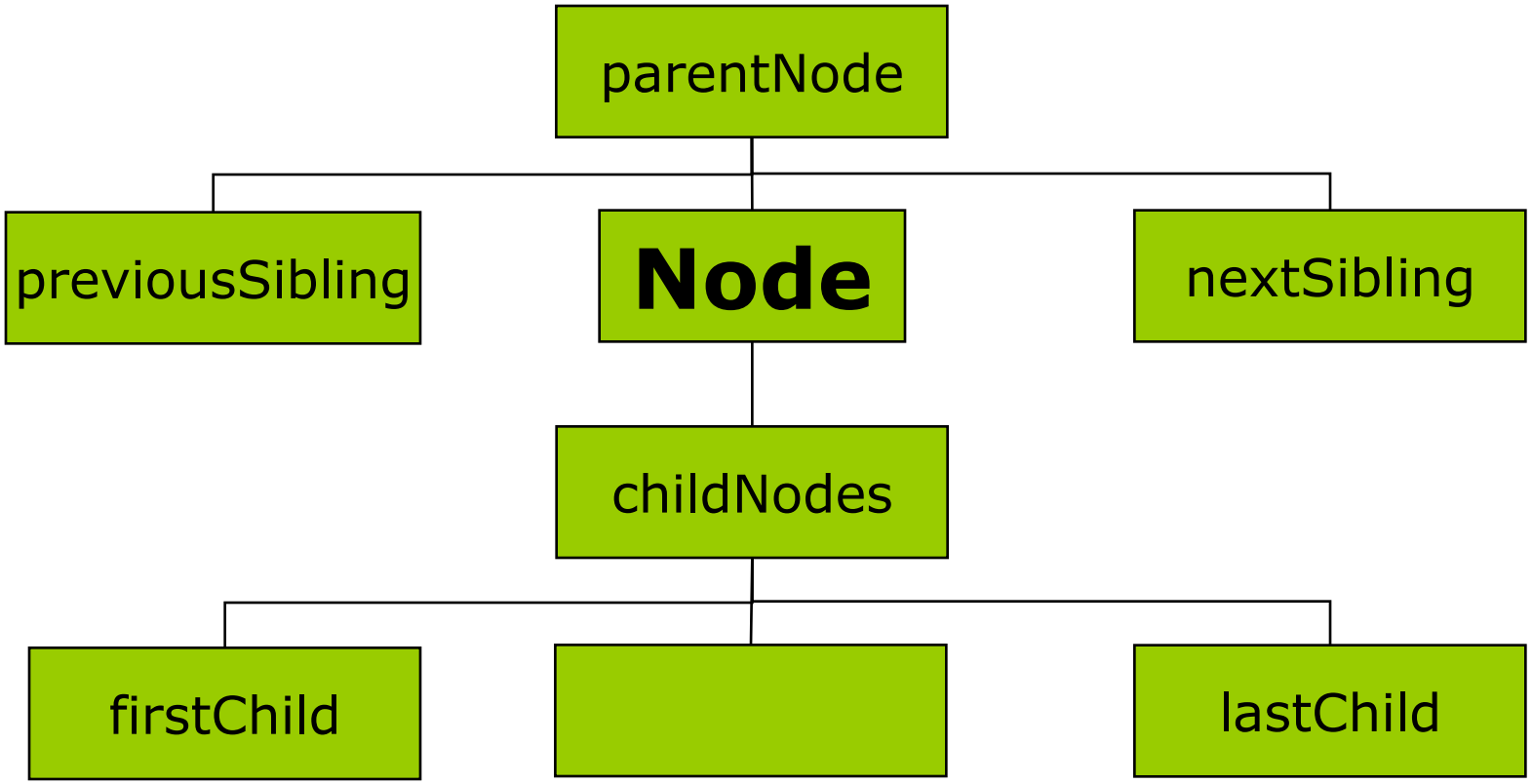
Rest des Parse-Baumes

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

- Beachte: PCDATA wird als eigener Knoten dargestellt.



Navigationsmodell & Verwandtschaften



- direkter Zugriff über Namen möglich: getElementsByTagName

Beispiel

```
<priceList>
  <coffee>
      <name>
          Mocha Java
      </name>
      <price>
          11.95
      </price>
  </coffee>
</priceList>
```

- Aufgabe: Gib den Preis von Mocha Java aus!
- Hierfür benötigen wir zwei Dinge:
  1. einen DOM-Parser
  2. eine passende Zugriffsmethode

Wie bekomme ich einen DOM-Parser?

Freie Universität Berlin

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

- liefert DocumentBuilderFactory

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

- liefert DOM-Parser

```
Document document = builder.parse("priceList.xml");
```

- DOM-Parser hat Methode parse().
- liefert in einem Schritt kompletten DOM-Parse-Baum

Wie sehen die Zugriffsmethoden aus?

Freie Universität Berlin

```
NodeList coffeeNodes =
   document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.getFirstChild();
 String data = thisNameNode.getFirstChild().getNodeValue();
 if (data.equals("Mocha Java")) {
   Node thisPriceNode = thisNameNode.getNextSibling();
   String price =  thisPriceNode.getFirstChild().getNodeValue();
   break; } }
```

= Java-Programm, das DOM-Methoden benutzt

org.w3c.dom.NodeList

- Kinder eines bestimmten Knotens (Kollektion)
- ***int getLength()***
  →Anzahl der Knoten in der Liste
- ***Node item(int index)***
  → item mit dem vorgegebenen Index

org.w3c.dom.Node

- ***Node appendChild(Node newChild)*** → Hängt einen neuen Kindknoten an die bereits existierenden an
- ***NodeList getChildNodes()*** → Liste mit allen Kindknoten
- ***Node getFirstChild()***
Node getLastChild() } Referenz auf den ersten/letzen Kinderknoten zurück
- ***Node getNextSibling()***
Node getPreviousSibling() } Referenz auf den nachfolgenden/ vorhergehenden Bruderknoten zurück
- ***String getNodeValue()*** → je nach Knotentyp der Wert/Inhalt (oder *null)*

org.w3c.dom.Document

- ***Element getDocumentElement()***
  → Referenz auf das Wurzelelement des Dokuments

- ***Element createElement(String tagName)***
  → neuer Element-Knoten mit angegebenem Namen

- ***Text createTextNode(String data)***
  →neuer Text-Knoten

- ***DocumentTyp egetDoctype()***
  → Document Type Declaration des Dokuments

org.w3c.dom.Element

- ***NodeList getElementsByTagName(String name)***
  → Liste von Kindelementen, die einen bestimmten Namen haben

- ***String getAttribute(String name)***
  → Wert des Attributs mit dem angegeben Namen

- ***Attr getAttributeNode(String name)***
  → Referenz auf das Attribut mit dem angegeben Namen zurück

- ***void removeAttribute(String name)***
  → Löscht das Attribut mit einem bestimmten Namen

Gib mir alle coffee-Elemente!


```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
```

- getElementsByTagName: direkter Zugriff auf Elemente über ihren Namen
- egal, wo Elemente stehen
- Resultat immer eine NodeList

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Betrachte Elemente der coffee-Liste!

Freie Universität Berlin

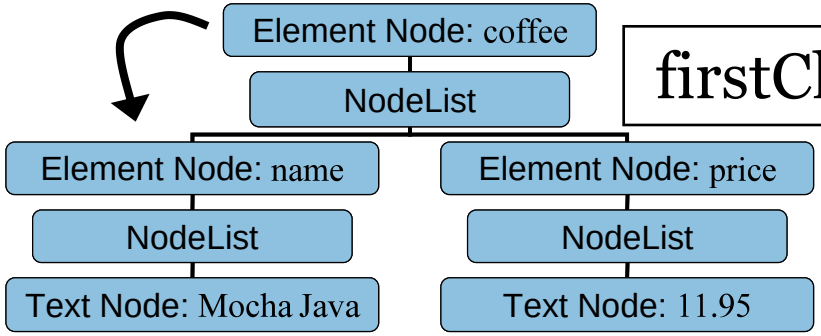
```
NodeList coffeeNodes =
   document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 ...
 }
```

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

coffeeNodes.item(0)

Gib mir erstes Kind von coffee!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.getFirstChild();
 String data = thisNameNode.getFirstChild().getNodeValue();
 if (data.equals("Mocha Java")) {
   Node thisPriceNode = thisCoffeeNode.getNextSibling();
   String price =  thisPriceNode.getFirstChild().getNodeValue();
   break; } }
```

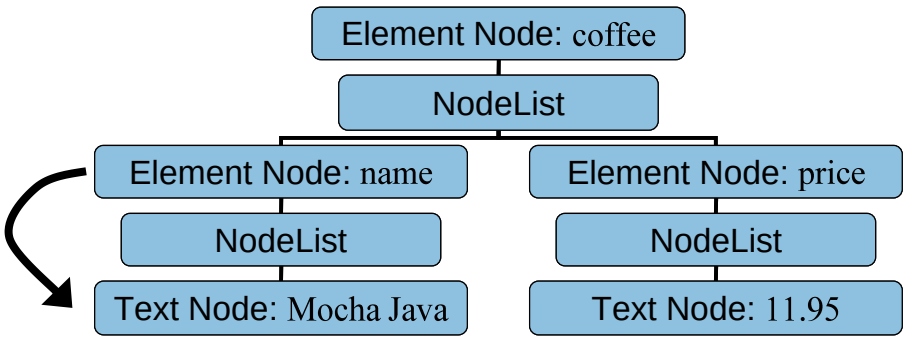


firstChild

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Gib mir den Inhalt von name!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.getFirstChild();
 String data = thisNameNode.getFirstChild().getNodeValue();
 if (data.equals("Mocha Java")) {
   Node thisPriceNode = thisCoffeeNode.getNextSibling();
   String price =  thisPriceNode.getFirstChild().getNodeValue();
   break; } }
```

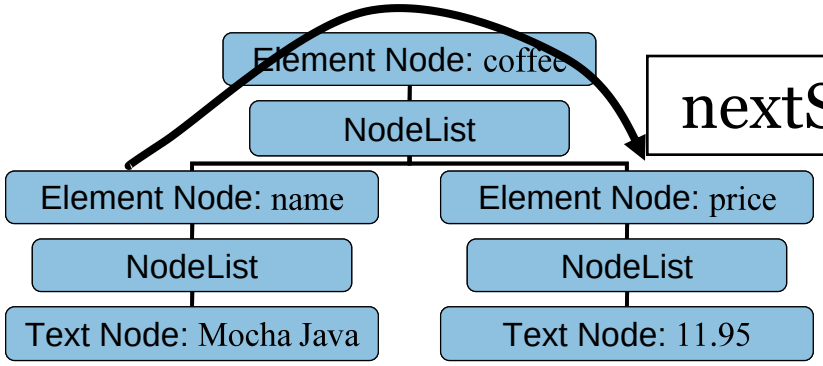



```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

firstChild

Gib mir das Geschwister-Element!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.getFirstChild();
 String data = thisNameNode.getFirstChild().getNodeValue();
 if (data.equals("Mocha Java")) {
   Node thisPriceNode = thisNameNode.getNextSibling();
   String price =  thisPriceNode.getFirstChild().getNodeValue();
   break; } }
```



nextSibling

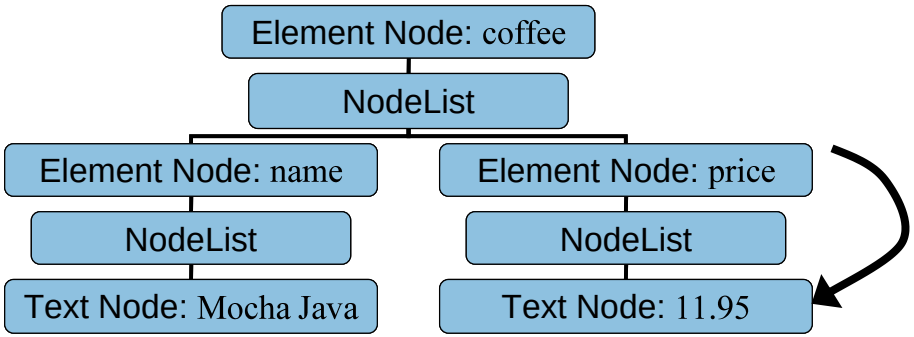
```
<?xml version="1.0" ?>
<priceList>
 <coffee>
  <name>Mocha Java</name>
  <price>11.95</price>
 </coffee>
</priceList>
```

Gib mir den Inhalt von price!

```
NodeList coffeeNodes = document.ge
for (int i=0; i < coffeeNodes.getLeng
 thisCoffeeNode = coffeeNodes.item(
 Node thisNameNode = thisCoffeeNo
 String data = thisNameNode.getFirst
 if (data.equals("Mocha Java")) {
   Node thisPriceNode = thisNameNo
  String price =  thisPriceNode.getFirstChild().getNodeValue();
   break; } }
```

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

firstChild



Operationen an den Knoten

- Knoten einfügen
  - Node.appendChild – einen Knoten an eine Folge von Kinderknoten anhängen
  - Node.insertBefore – en welcher Stelle ein Knoten in einer Liste von Kinderknoten eingefügt werden soll
  - Node.clone – klonen eines Knoten (und evt. seiner Unterknoten)

- Knoten entfernen
  - Node.removeChild – einen bestehenden Knoten löschen
  - Node.replaceChild – eine bestehenden Knoten mit einem neuen Knoten ersetzen

Besonderheit des Attributknotens

- nicht als Knoten im eigentlichen Sinne betrachte
- keine Kinder des Elementsknotens, zu dem sie gehören
- nicht als Teil des Knotenbaums zu betrachten
- nicht über Methoden für Knoten erreichbar
- zwischen mehreren Attributen eines Knoten besteht keine Geschwisterbeziehung

- Zugriff über:
  - Node.Attributes
  - getAttributes / setAttributes
  - Attr.ownerElement – liefert das Element, zu dem Attribut gehört

Vor- und Nachteile von DOM

+ Kontext (Parse-Baum) muss nicht von Anwendung verwaltet werden.
+ einfache Navigation im Parse-Baum
+ direkter Zugriff auf Elemente über ihre Namen
+ nicht nur Parsen, sondern auch Modifikation und Erstellung von XML-Dokumenten

- speicherintensiv
- abstrahiert nicht von XML-Syntax

SAX oder DOM?

Freie Universität Berlin

SAX	DOM

SAX oder DOM?

Freie Universität Berlin

SAX

- geeignet, um gezielt bestimmte Teile von XML-Dokumenten herauszufiltern, ohne zu einem späteren Zeitpunkt andere Teile des Dokumentes zu benötigen
- nur Parsen, kein Erstellen oder Modifizieren von XML-Dokumenten

DOM

- geeignet, um auf unterschiedliche Teile eines XML-Dokumentes zu verschiedenen Zeitpunkten zuzugreifen
- auch Erstellen und Modifizieren von XML-Dokumenten

StAX – Streaming API for XML

StAX

- weiterer Ansatz zum Parsen von XML-Daten (2004)
- ereignisorientierter (meistens) Pull Parser
  - Verarbeitung, wenn Applikation bereit
  - Struktur des Codes entspricht Struktur des XML-Dokuments
  - Zustandsverwaltung

- verbindet die Vorteile von SAX und DOM
  - effizientes Arbeiten mit großen Dokumenten
  - Schreiben/Erzeugen von XML-Dokumeten

Cursor API

- streamingbasiert
- Cursor zeigt auf jeweils ein XML-Infoset-Element
- Anwendung muss nächsten Infoset-Element „pullen“
- „Quasi-SAX“

```
public interface XMLStreamReader {
        public int next() throws XMLStreamException;
        public boolean hasNext() throws XMLStreamException;
        …
}
```

```
public interface XMLStreamWriter {
        public void writeStartElement(String localName) throws XMLStreamException;
        public void writeEndElement() throws XMLStreamException;
        public void writeCharacters(String text) throws   XMLStreamException;
        …
}
```

http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/StAX3.html

StAX

```
URL u = new URL("http://www.cafeconleche.org/");
InputStream in = u.openStream();
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader parser =
   factory.createXMLStreamReader(in);
[…]
while (true) {
   int event = parser.next();
   if (event == XMLStreamConstants.END_DOCUMENT) {
      parser.close();
      break;
   }
   if (event == XMLStreamConstants.START_ELEMENT) {
      System.out.println(parser.getLocalName());
   }
}
```

Elliotte Rusty Harold. An Introduction to StAX
http://www.xml.com/pub/a/2003/09/17/stax.html

Iterator API

- eventbasiert
- Menge diskreter Event-Objekte
- Anwendung muss nächstes Event „pullen“

```
public interface XMLEventReader extends Iterator {
        public XMLEvent nextEvent() throws XMLStreamException;
        public boolean hasNext();
        …
}
```

vs. Cursor:
- einfacheres Editieren
- Verwendbar in Arrays, Listen und Maps

```
public interface XMLEventWriter {
        public void flush() throws XMLStreamException;
        public void close() throws XMLStreamException;
        public void add(XMLEvent e) throws XMLStreamException;
        public void add(Attribute attribute) throws XMLStreamException;
         ...
}
```

http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/StAX3.html

StAX – Eingenschaften

+ Durch die verschiedenen API- Realisierungen flexiblerer Umgang

+ Je nach Wahl ähnlich performant wie SAX

+ Fähigkeit, Dokumente schreiben zu können

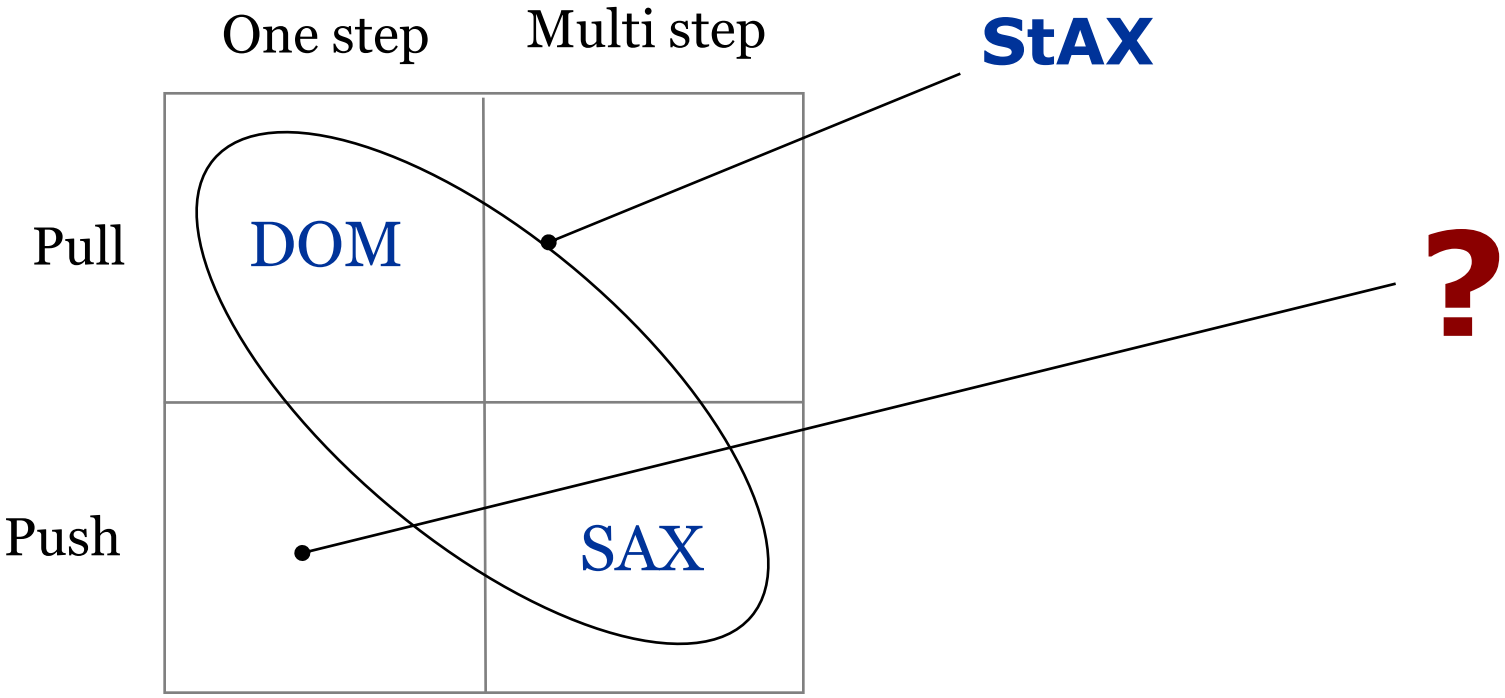
SAX, DOM & StAX

Parservergleich

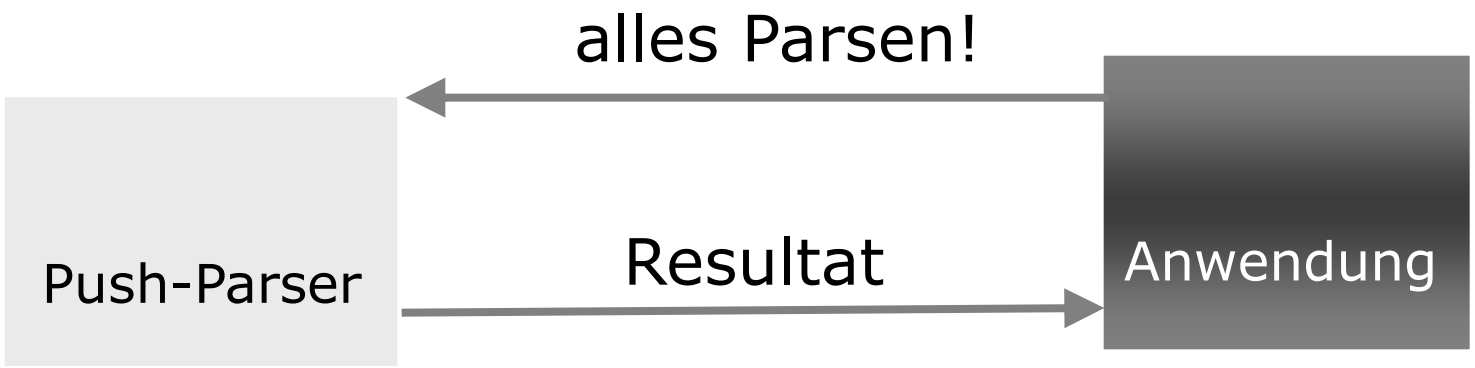
Kriterium \ Parser	SAX	DOM	StAX
Klassifikation	Push-Mehrschritt	Pull-Einschritt	Pull-Mehrschritt
wiederholter Zugriff	nein	ja	nein
geeignet für Verarbeitung großer Dokumente	ja	nein	ja
Dokumenten-manipulation möglich	nein	ja	ja

Und sonst?
Untertitel: Fragen aus der Vorlesung

XML-Parser



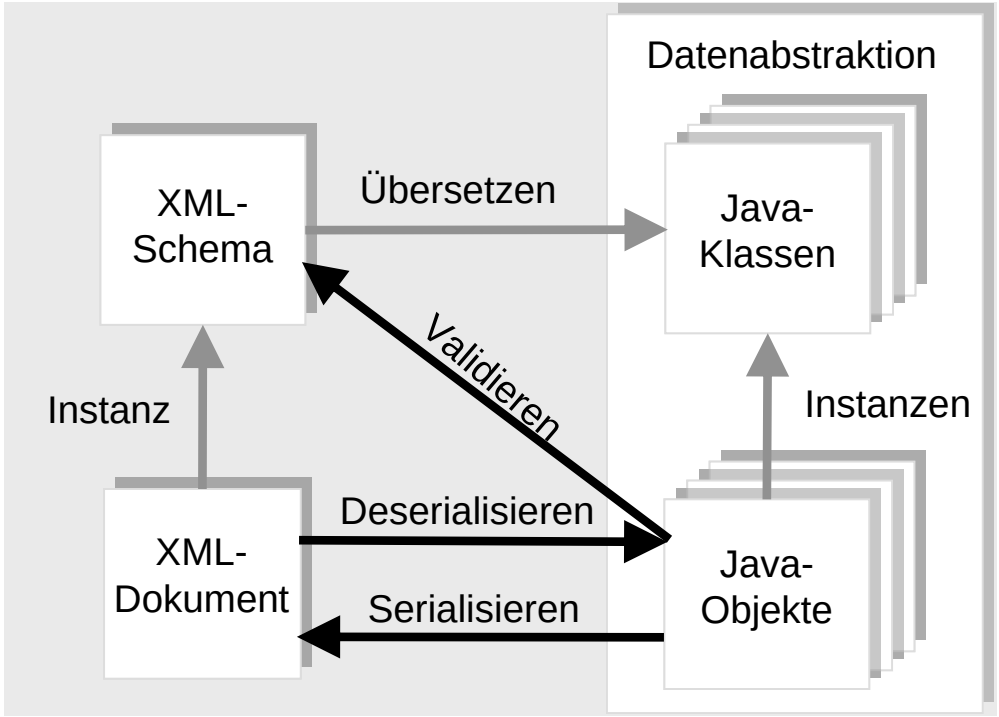
One-Step-Push-Parser



- Parser hat Kontrolle über das Parsen und parst das gesamte Dokument in einem Schritt
- liefert irgendein festes Resultat (Wert?, XML-Element?)
- Macht das Sinn?
  - Fiktives Beispiel: 100000 identische medizinische Dokumente in denen auf Basis einer Information true oder false ausgegeben werden soll
  - Ist das wirklich etwas, was man auf Ebene des Parsers selbst implementieren sollte?

Schema-Übersetzer

Schema-Übersetzer



- Klassen/Methoden werden generiert
- Zweck: Schnittstelle, die von XML abstrahiert
- Lesen, Modifizieren und Erstellen von XML-Dokumenten

- JAXB: Java Architecture for XML Binding
- Teil von Java WSDP 2.0

Beispiel

```
<priceList>
  <coffee>
      <name>Mocha Java</name>
      <price>11.95</price>
  </coffee>
</priceList>
```

zugehöriges XML-Schema

JAXB

```
public interface PriceList {
  java.util.List getCoffee();
  public interface CoffeeType {
      String getName();
      void setName(String value)
      java.math.BigDecimal getPrice();
      void setPrice(java.math.BigDecimal value) } }
```

Übersetzung von xsd:choice


```
<xs:element name="BoolCommentOrValue">
  <xs:complexType>
    <xs:choice>
      <xs:element name="bool" type="xs:boolean"/>
      <xs:element name="comment" type="xs:string"/>
      <xs:element name="value" type="xs:int"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- Wie kann dieser Datentyp nach Java übersetzt werden?

Übersetzung von xsd:choice

```
<xs:element name="BoolCommentOrValue">
  <xs:complexType>
    <xs:choice>
      <xs:element name="bool"
      <xs:element name="comm
      <xs:element name="value
    </xs:choice>
  </xs:complexType>
</xs:element>
```

```
public interface BoolCommentOrValue {
  int getValue();
  void setValue(int value);
  boolean isSetValue();

  java.lang.String getComment();
  void setComment(java.lang.String value);
  boolean isSetComment();

  boolean getBool();
  void setBool(boolean value);
  boolean isSetBool();

  Object getContent();
  boolean isSetContent();
  void unSetContent(); }
```

Vor- und Nachteile von JAXB

+ bessere Performance bei komplexen Aufgaben
+ übersichtlicher und robuster → weniger fehleranfällig
+ geringerer Wartungsaufwand
+ bessere Skalierbarkeit für steigenden Funktionsumfang.
+ Schnittstelle, die von XML abstrahiert

- Keine Unterstützung der Typsubstitution

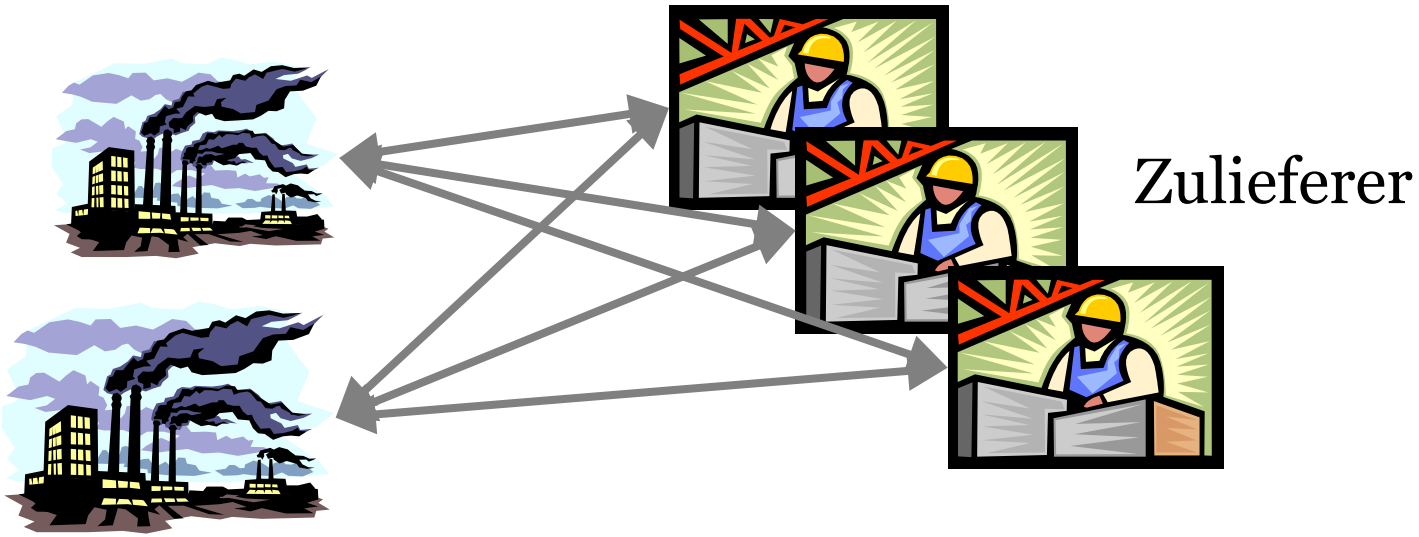
Warum?

Warum noch XML lernen?

- Schema-Übersetzer: XML-Schema kann in Java- oder C#-Klassen übersetzt werden.
- Hiermit können XML-Dokumente gelesen, modifiziert und erstellt werden, ohne dass XML sichtbar ist.

Warum sich also noch mit XML und XML-Schemata beschäftigen?

Typisches E-Business-Projekt

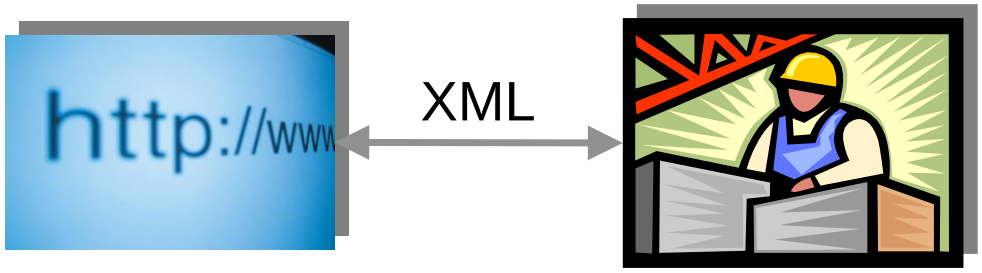


- Branchenvertreter wollen über das Internet miteinander Geschäfte abwickeln.
- Sie müssen sich auf ein Austauschformat einigen.

E-Business-Projekt: Phase I

- Welche Geschäftsdaten sollen ausgetauscht werden?
- Gibt es bereits einen passenden Branchenstandard?
- Wie sollen die Geschäftsdaten in XML repräsentiert werden?
- Gibt es bereits einen geeigneten XML-Standard?
- Ziel: Branchenstandard in Form eines XML-Schemas

Software-Architekten entwickeln gemeinsam einen XML-basierten Branchenstandard.



gegeben

- Branchenstandard in Form eines XML-Schemas
- gemeinsames Verständnis der XML-Syntax

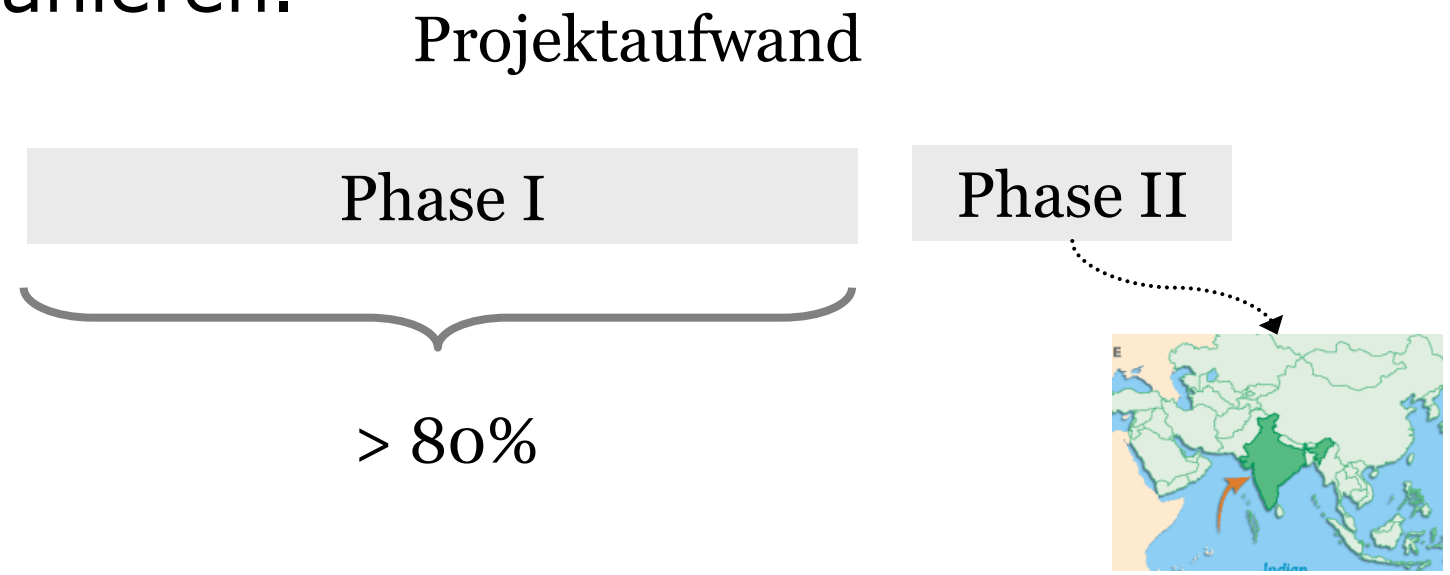
Aufgabe

- Realisierung der Schnittstelle zwischen betriebsinterner Software und XML-Standard.

Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.

Warum sich mit XML beschäftigen?

- Phase I: Software-Architekten beschäftigen sich intensiv mit entsprechender Branche, XML und XML-Schemata.
- Phase II: Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.



Wie geht es weiter?

heutige Vorlesung

- ☑ SAX-, DOM & StAX-Parser
- ☑ Vor- und Nachteile von SAX, DOM & StAX
- ☑ Schema-Übersetzer

Vorlesung morgen

- XPath & Co.