

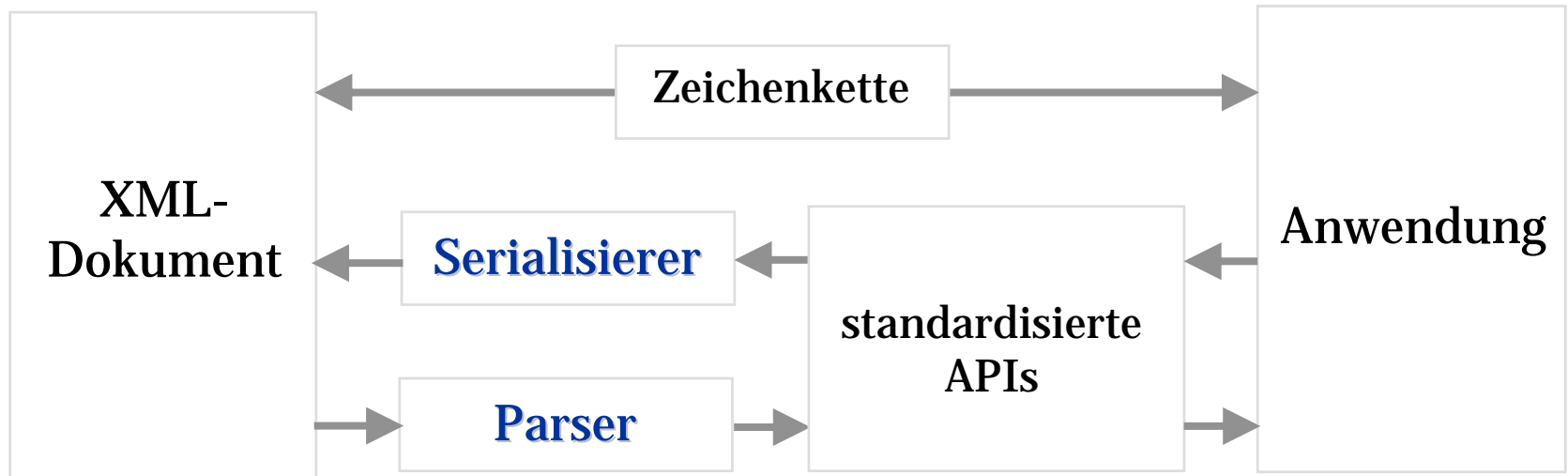
# XML-Parser

letzte Woche

- ☑ XML-Schema
  - Datentypen
  - Element- und Attribut-Deklarationen
  - Typsubstitution

## heutige Vorlesung

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



## Parser

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

## Serialisierer

- Datenstruktur → XML-Dokument

## 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

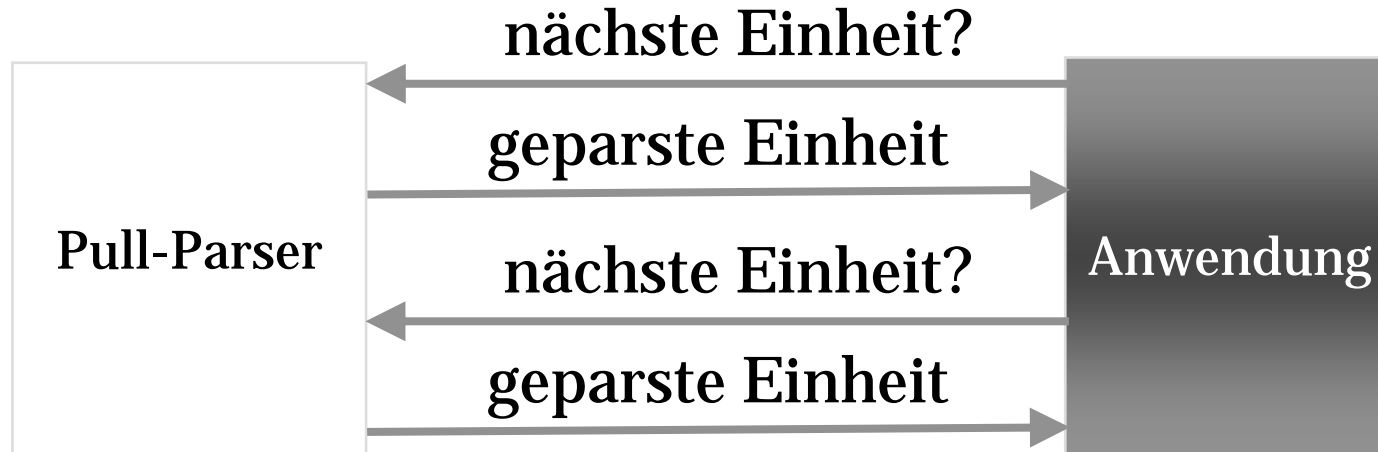
- Wird das XML-Dokument in einem Schritt vollständig geparkt oder Schritt für Schritt?
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden

## Validierender Parser

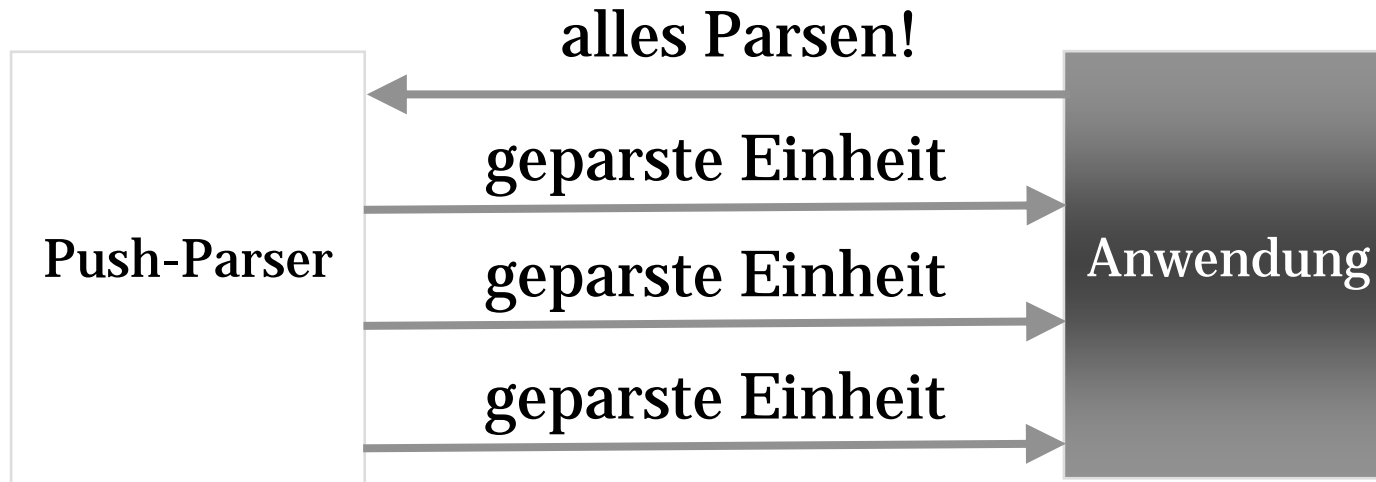
- Ist das XML-Dokument valide?
- DTD oder XML-Schema erforderlich
  
- Ist das XML-Dokument wohlgeformt?

## Nicht-validierender Parser

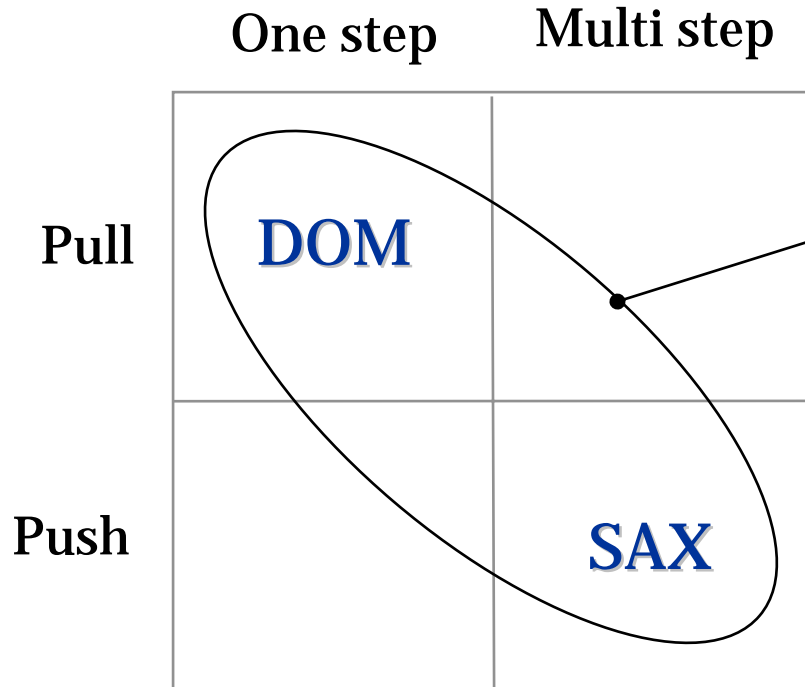
- Ist das XML-Dokument wohlgeformt?



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



- 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.



**JAXP**

- **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

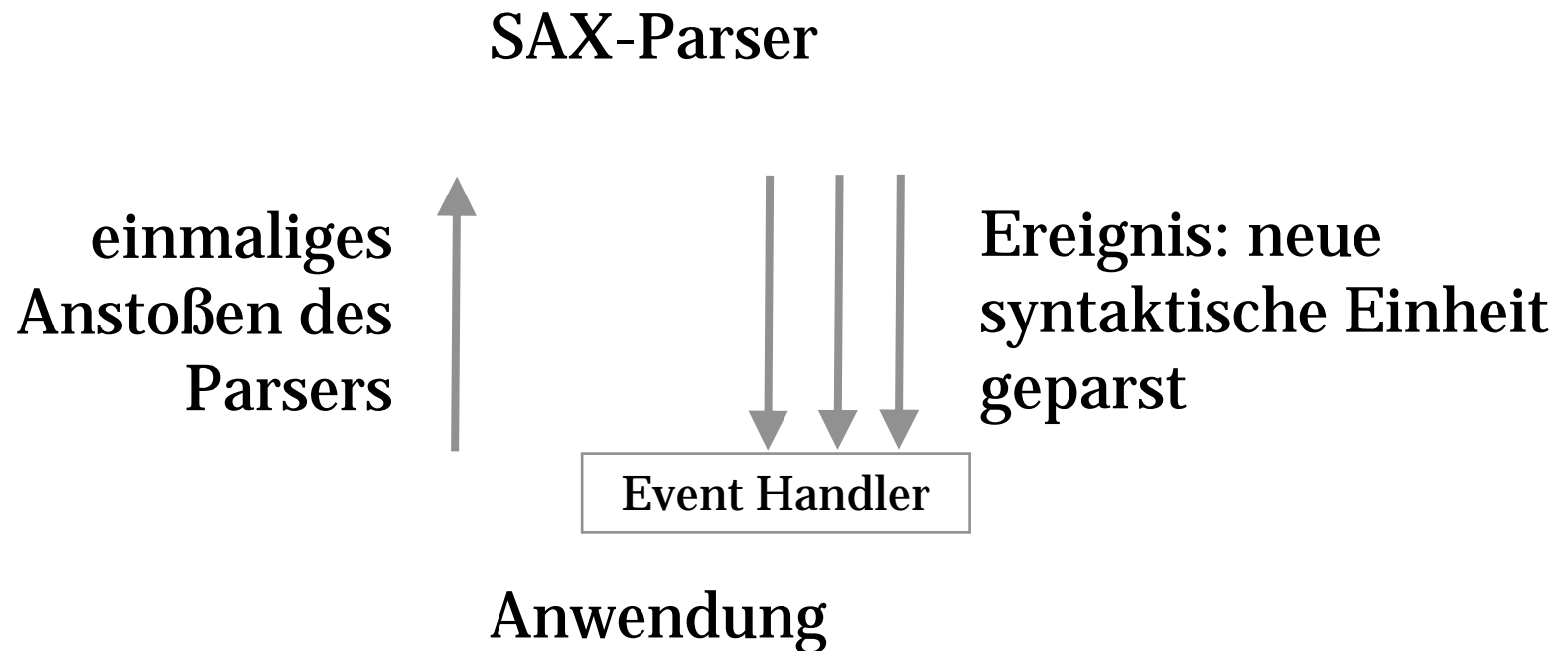


# SAX-Parser

# SAX: Simple API for XML

- **Mehrschritt-Push-Parser für XML**
- kein W3C-Standard, sondern *de facto* Standard
- standardisiertes 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





# Beispiel

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

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

- 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

# 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 geparkt wurde
- `endElement` – wenn ein Endtag geparkt wurde
- `characters` – wenn beim Parsen des Elementinhalts Zeichendaten (`#PCDATA`) angetroffen werden

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

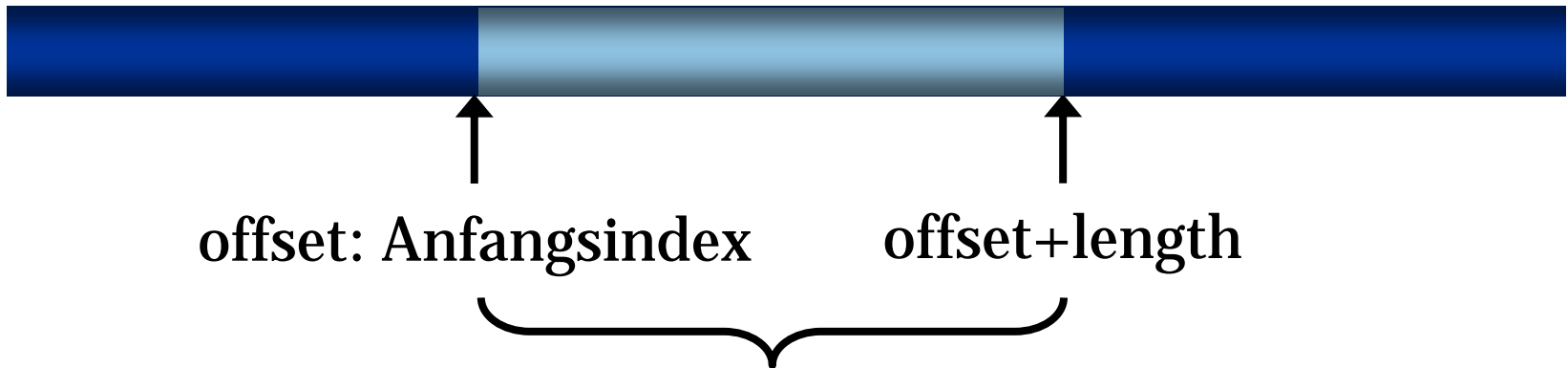
throws SAXException

- **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);
```



```
<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?

```
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

- Entwurfsmuster aus „Design Patterns“ von Gamma, Helm, Johnson, Vlissides (1995)
- liefert ein Objekt
- Objekt ist Instanz einer abstrakten Klasse oder einem Interface.
- abstrakte Klasse / Interface von mehreren Klassen implementiert
- Beispiel: `Iterator i = list.iterator();`
- Beispiel: `SAXParser saxParser = factory.newSAXParser();`

# Logik der Callback-Methoden

<priceList>

<coffee>

<**name**>

**Mocha Java**

</name>

<**price**>

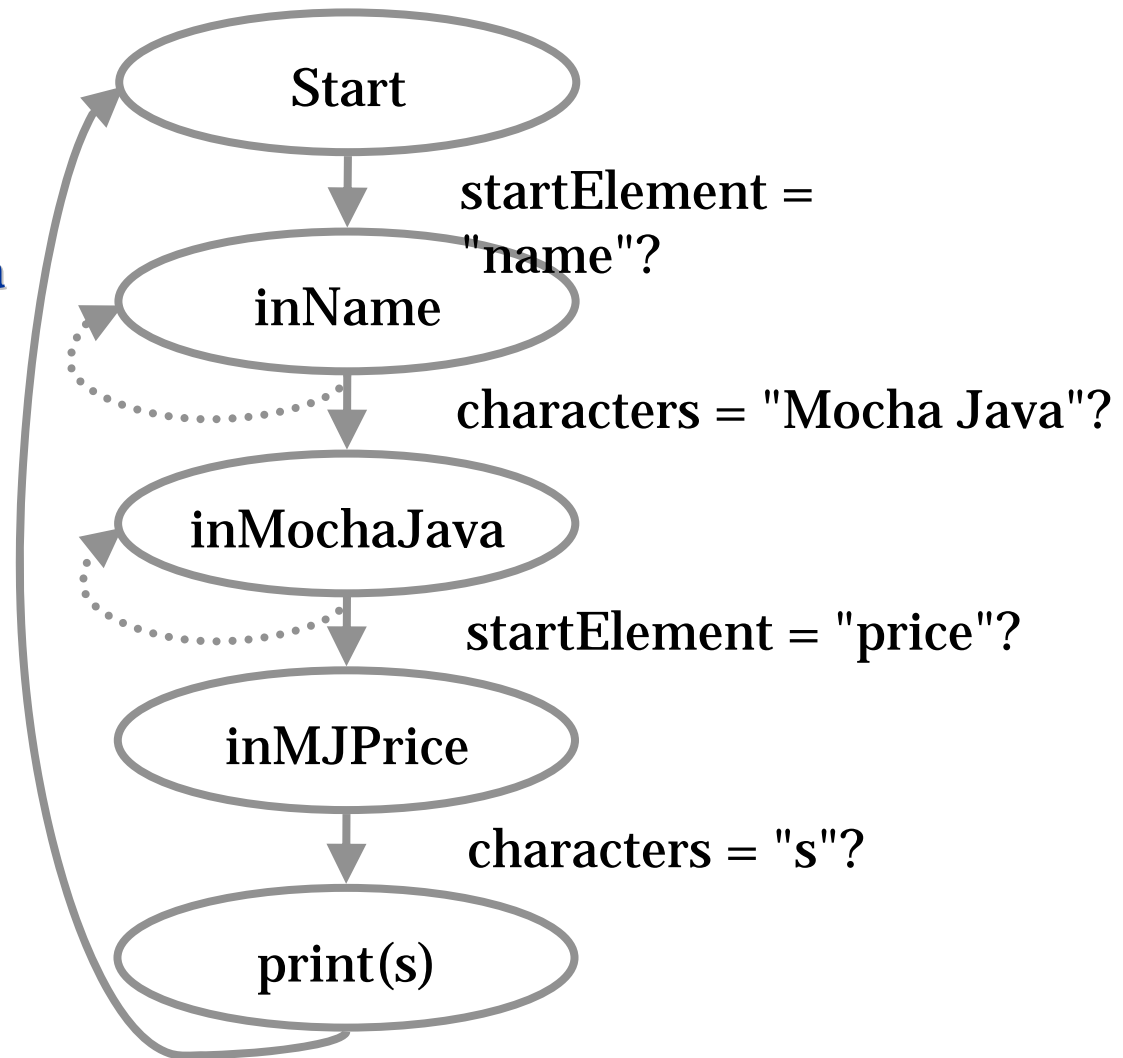
**11.95**

</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

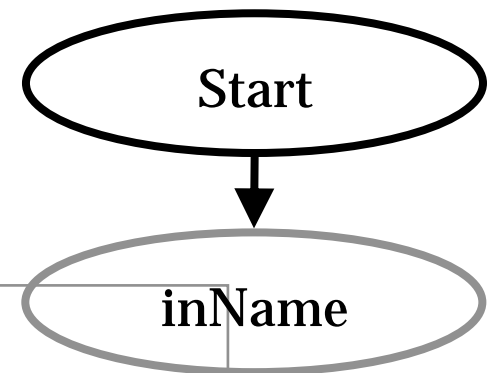
# Start: Auf <name> warten

```
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;  
        Start  
    }  
    else if (inMJPrice)  
        System.out.println(s);  
    inMJPrice = false;  
}
```

- Anfangszustand
- keine eigene Zustandsvariable
- alle Zustandsvariablen = false

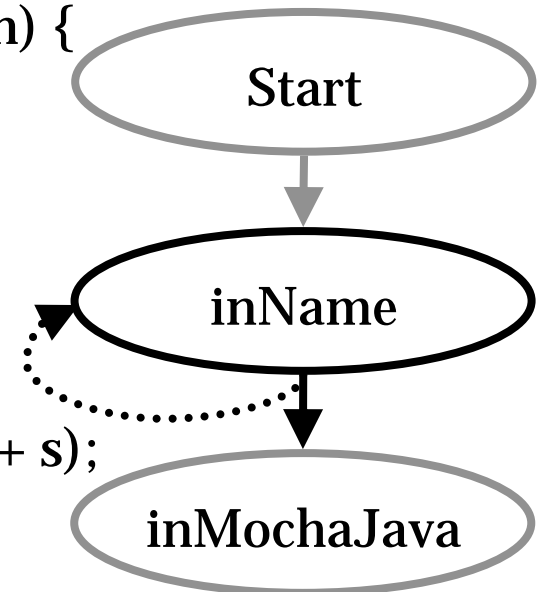


# inName: Auf "Mocha Java" warten

```
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>
```

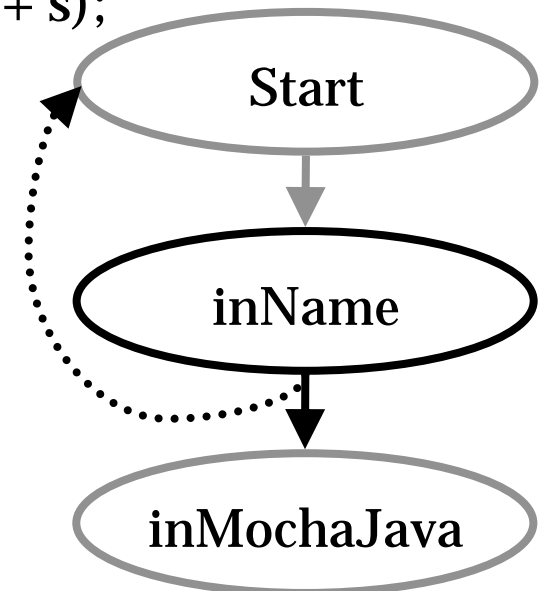
```
public void characters(char [] buf, int offset, int len) {  
  String s = new String(buf, offset, len);  
  if (inName && s.equals("Mocha Java")) {  
    inMochaJava = true;  
    inName = false;  }  
  else if (inMJPrice) {  
    System.out.println("The price of Mocha Java is: " + s);  
    inMJPrice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



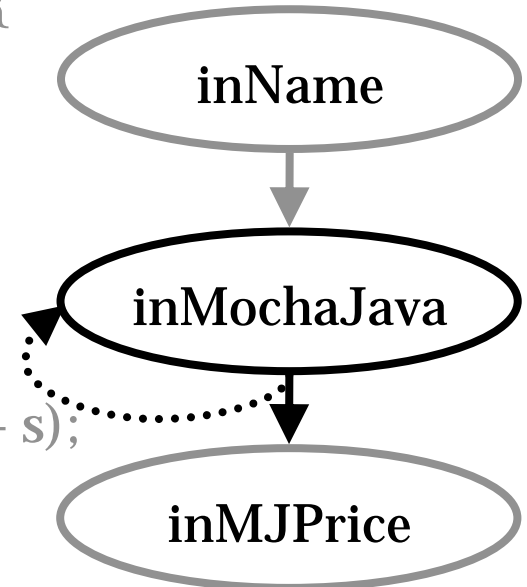


# inMochaJava: Auf <price> warten

```
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; } }
```

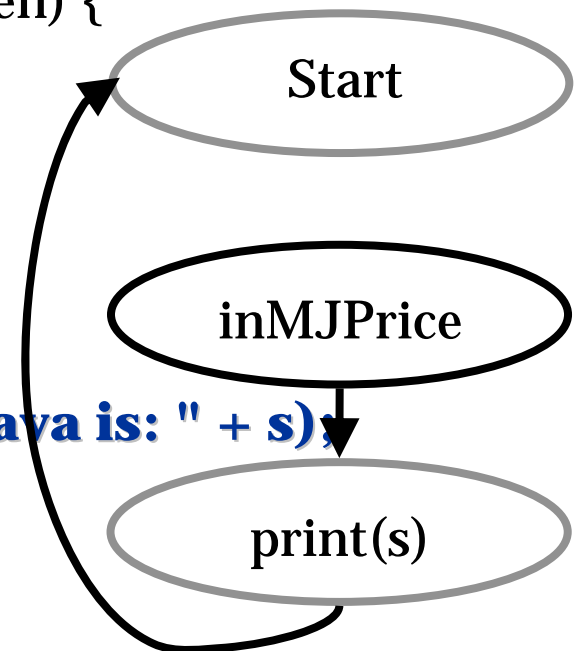


# inMJPrice: Preis ausgeben

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

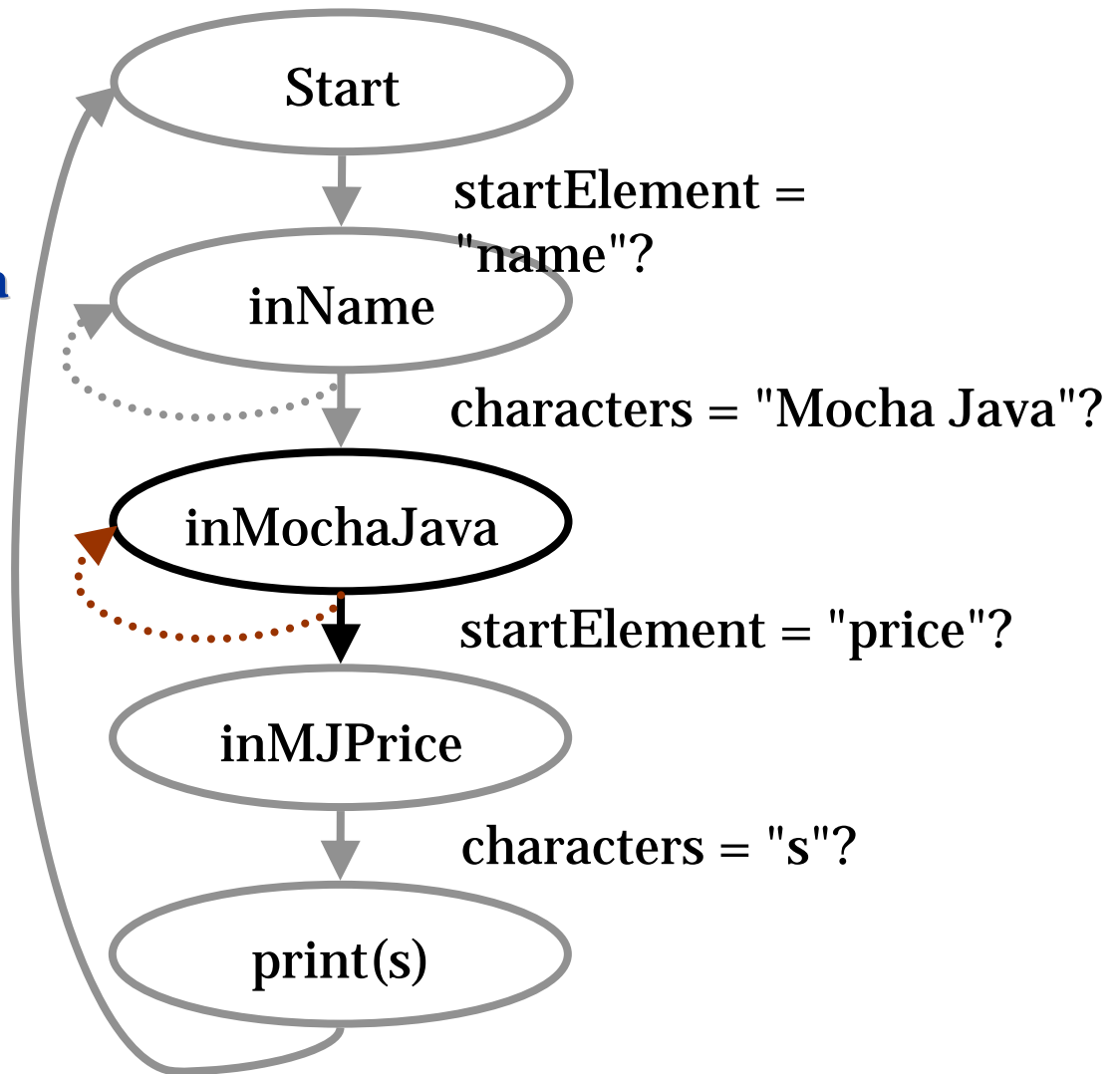
```
<name>Mocha Java</name>  
<price>11.95</price>
```

```
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

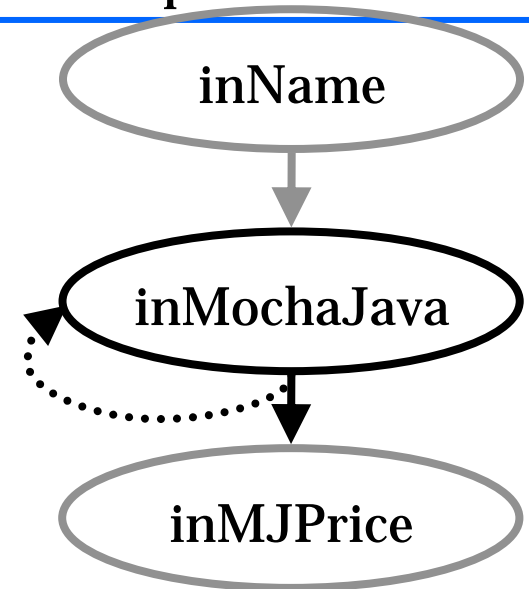
```
<priceList>
  <coffee>
    <name>
      Mocha Java
    </name>
    <name>
      MS Java
    </name>
    <price>
      11.95
    </price>
  </coffee>
</priceList>
```



```
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>
```

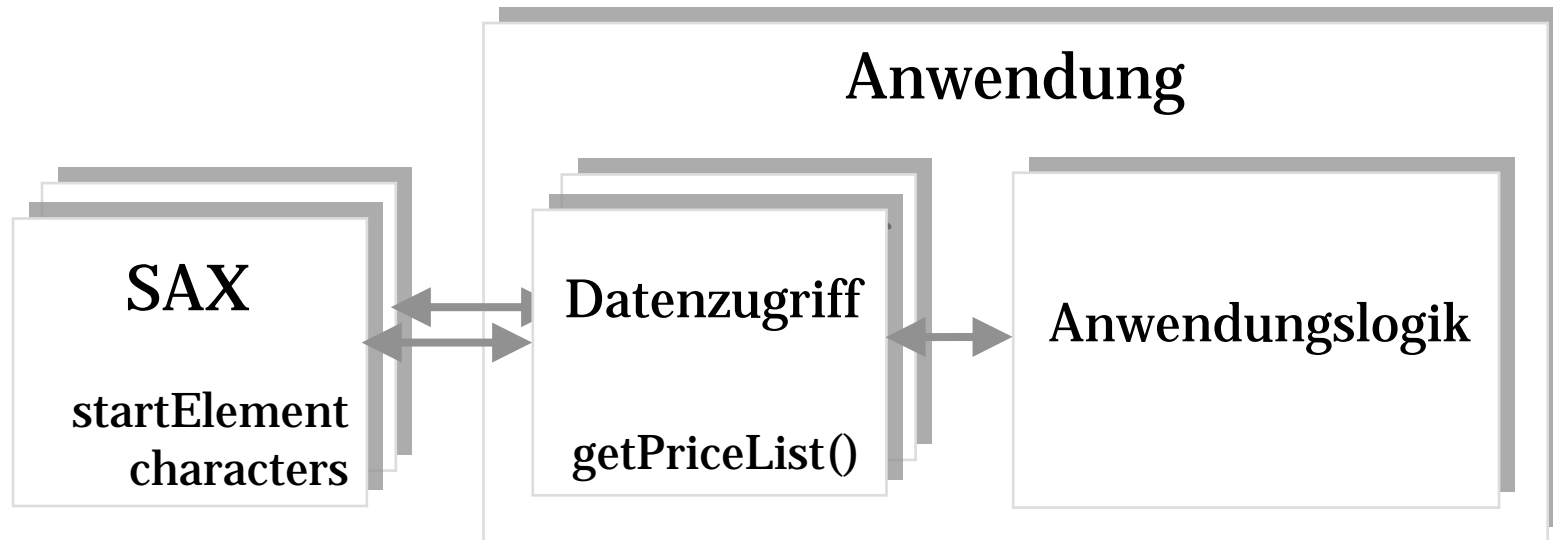
- inMochaJava erwartet <price>
  - kommt stattdessen <name>, wird aktueller Zustand inMochaJava nicht verändert
  - kommt danach <price>, wird aktueller Zustand inMJPrice
- ⇒ Preis von MS Java wird ausgegeben!



- 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)<sup>+</sup> verlangen
- ⇒ Syntax- und Strukturfehler kann bereits der SAX-Parser abfangen
- ⇒ Callback-Methoden können dann von wohlgeformten und zulässigen Dokument ausgehen.

- + sehr effizient, auch bei großen XML-Dokumenten
- 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

# DOM-Parser



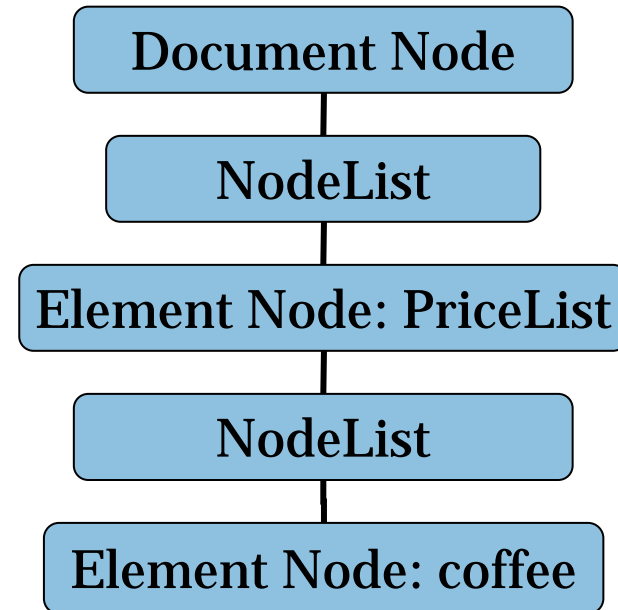
# Document Object Model (DOM)



- 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**

- **Level 1:**
  - DOM-Kern (DOM Core) + Spezialisierungen für HTML und XML;
  - Bearbeitung von Dokumentinhalten & Navigation innerhalb der Dokumente
  - <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>
  
- **Level 2:**
  - Namensräumen + CSS
  - Änderungen der Baumstruktur eines Dokuments
  - unterschiedliche Views von Dokumenten
  
- **Level 3 (W3C Working Draft):**
  - Anpassungen an XML Infoset, XML Base und XPath,

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



- Beachte: Dokument-Wurzel (Document Node)  $\neq$  priceList
- **Document Node**: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer Node**List** als Kind

# Rest des Parse-Baumes

```
<?xml version="1.0" ?>
```

```
<priceList>
```

```
<coffee>
```

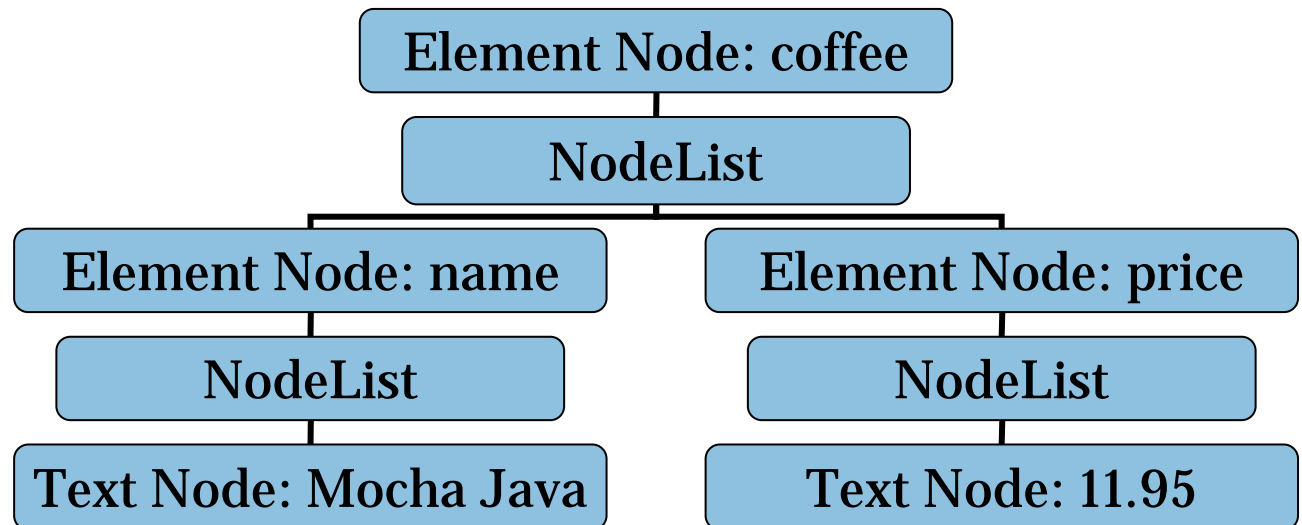
```
<name>Mocha Java</name>
```

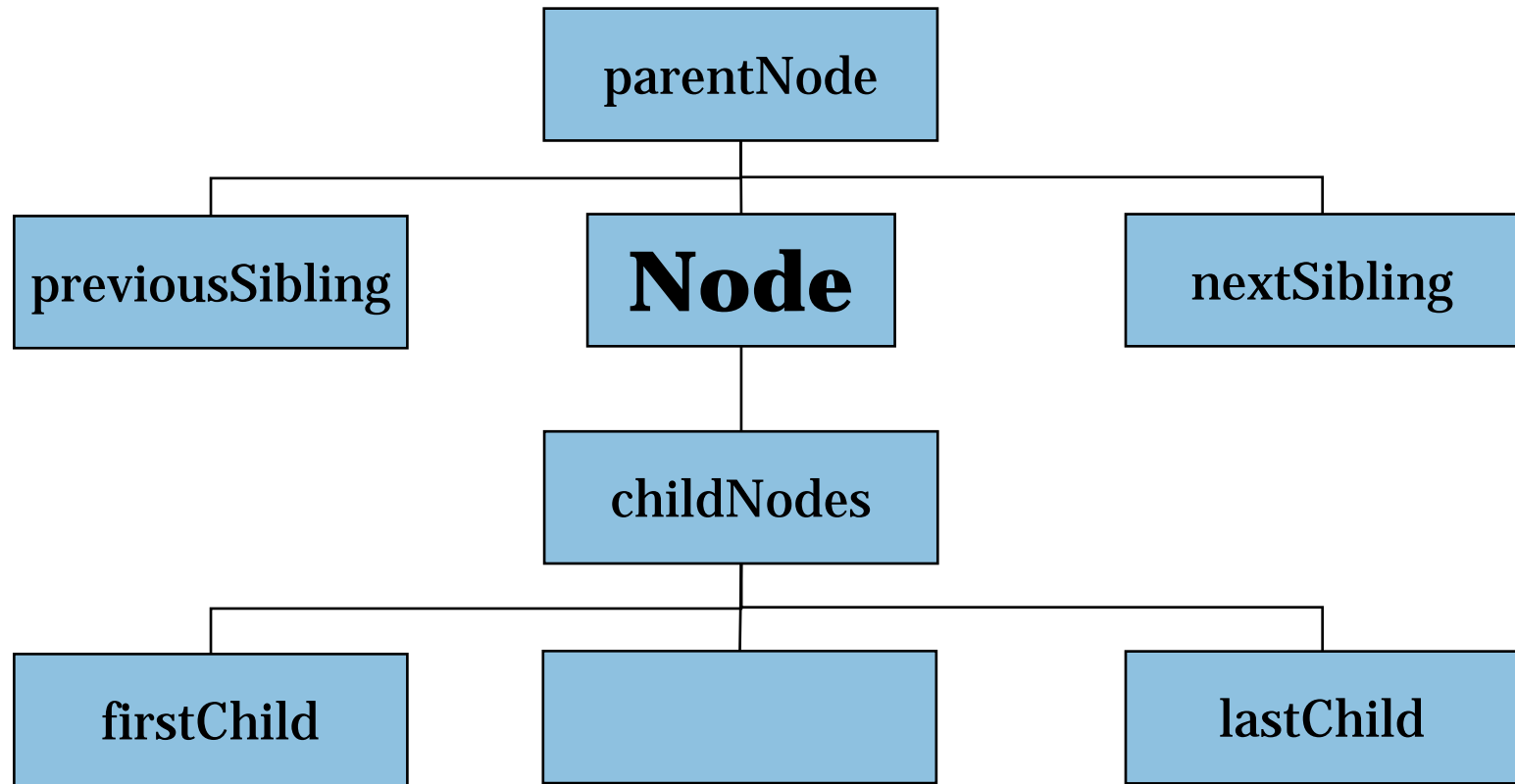
```
<price>11.95</price>
```

```
</coffee>
```

```
</priceList>
```

- Beachte: PCDATA wird als eigener Knoten dargestellt.





direkter Zugriff über Namen möglich: `getElementsByName`

```
<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?

```
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

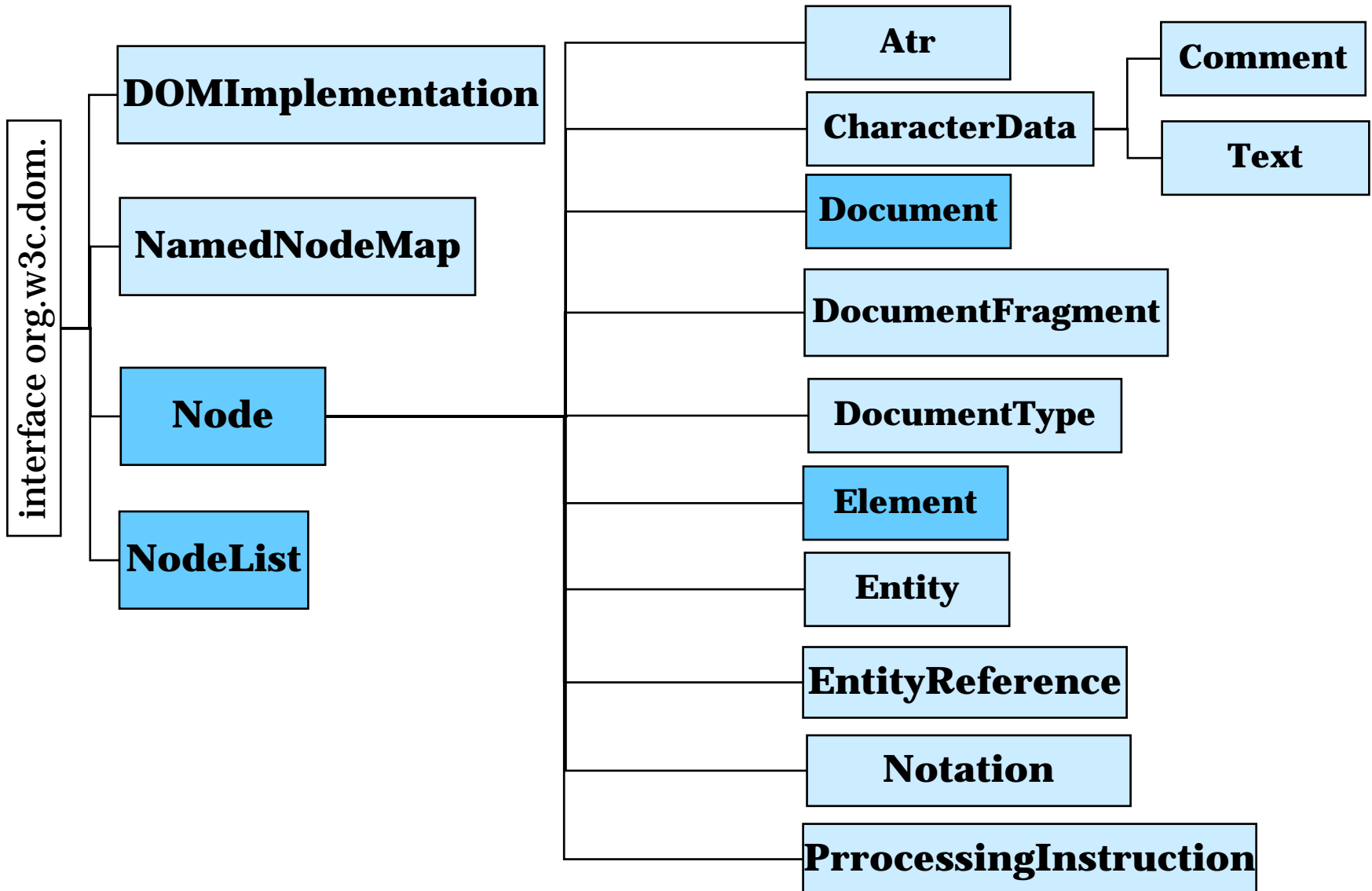
# Wie sehen die Zugriffsmethoden aus?

```
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



# Interface Hierarchy



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

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

- ***Element getElement()***  
→ Referenz auf das Wurzelement des Dokuments
- ***Element createElement(String tagName)***  
→ neuer Element-Knoten mit angegebenem Namen
- ***Text createTextNode(String data)***  
→ neuer Text-Knoten
- ***DocumentType getDoctype()***  
→ Document Type Declaration des Dokuments

- ***NodeList getElementsByTagName(String name)***  
→ Liste von Kindelementen, die einen bestimmten Namen haben
- ***String getAttribute(String name)***  
→ Wert des Attributs mit dem angegebenen Namen
- ***Attr getAttributeNode(String name)***  
→ Referenz auf das Attribut mit dem angegebenen 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!

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

coffeeNodes.item(0)

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

# 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; } }
```

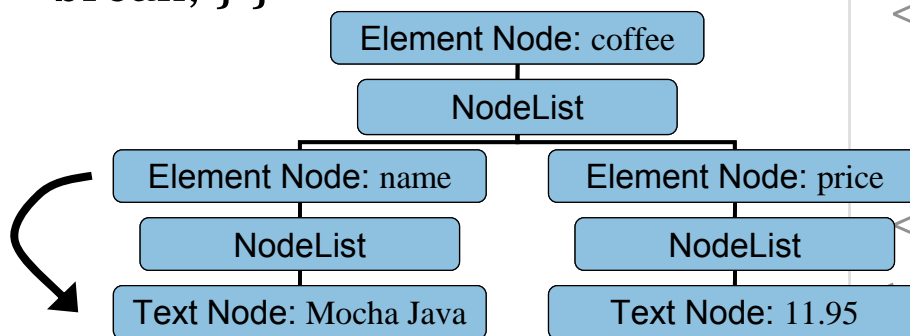
firstChild

```
<?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; } }
```

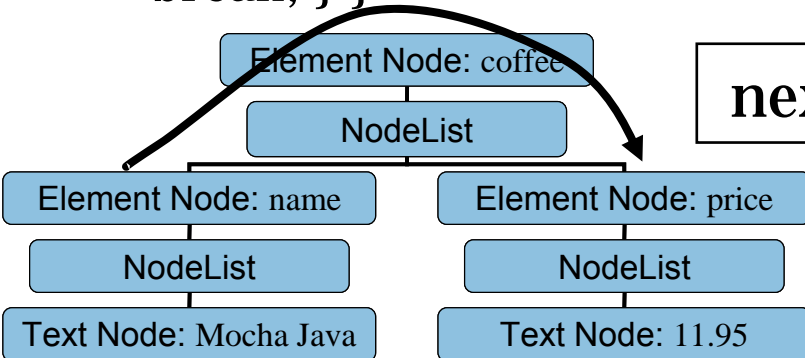


```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

**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.getFirst
break; } }
```



nextSibling

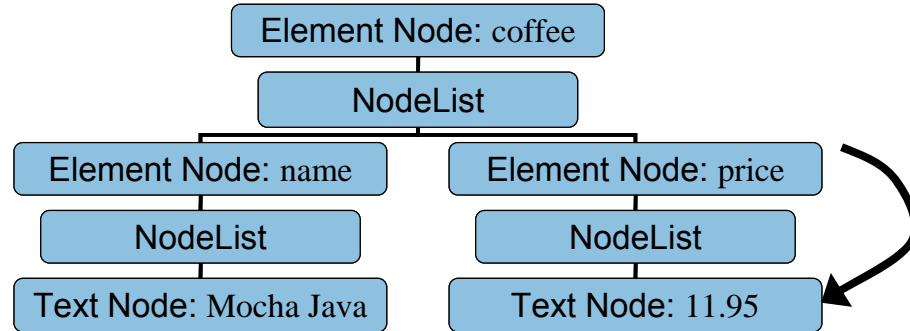

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

# Gib mir den Inhalt von price!

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

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

**firstChild**

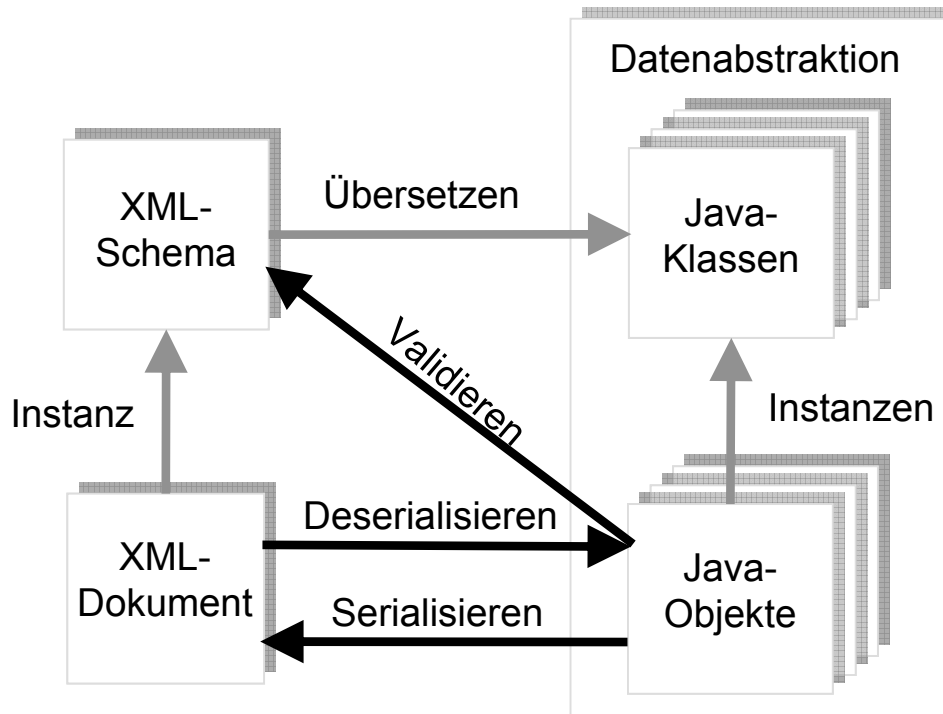


- + Kontext (Parse-Baum) muss nicht von Anwendung verwaltet werden.
- + einfache Navigation im Parse-Baum
- + direkter Zugriff auf Elemente über ihre Namen
- + nicht nur Parsen, sondern auch Modifikation und Erstellung von XML-Dokumenten
  
- speicherintensiv
- abstrahiert nicht von XML-Syntax

# SAX oder DOM?

SAX	DOM
<b>ereignis-orientierter</b> Ansatz	<b>modell-orientierter</b> Ansatz
	vollständige Umsetzung in eine <b>Baumstruktur</b>
	mehrere Verarbeitungsmöglichkeiten
XML-Dokument als <b>Eingabestrom</b> (Streaming-Verfahren)	XML-Dokument <b>vollständig im Speicher</b> (Baummodell des Dokuments )
<b>schnelle</b> Verarbeitung von großen XML-Dokumenten	<b>langsame</b> Verarbeitung von großen XML-Dokumenten
<b>wenig</b> Hauptspeicher benötigt	<b>mehr</b> Hauptspeicher benötigt
	nach dem Einlesen kann auf alle Teilstrukturen des XML-Dokuments zugegriffen werden

# 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

```
<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="comr
```

```
<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

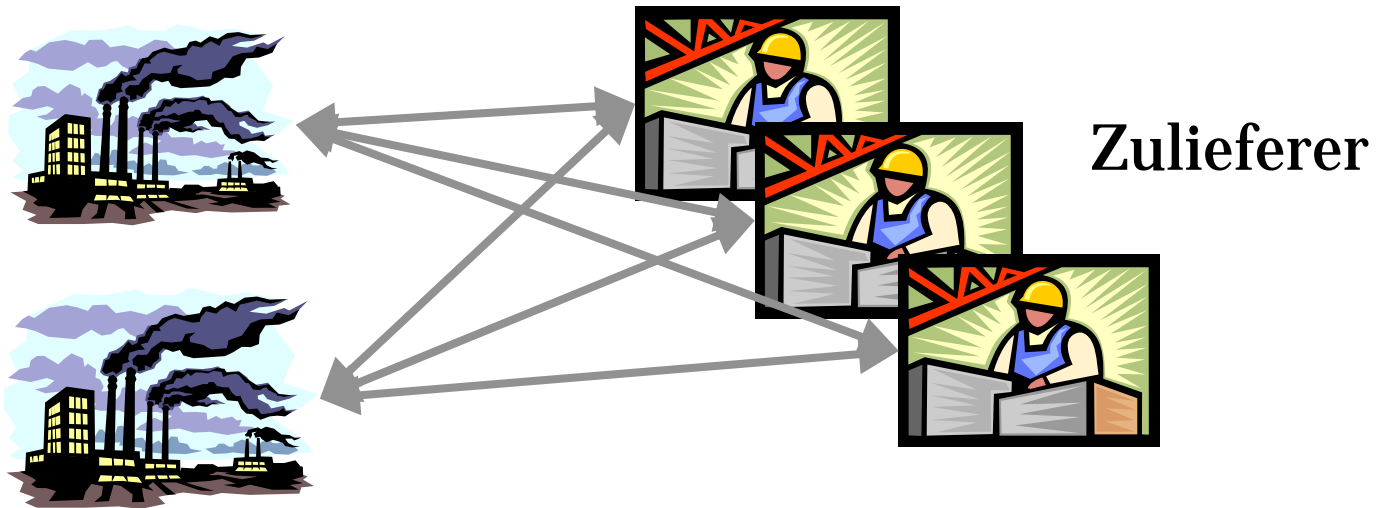
- + bessere Performance bei komplexen Aufgaben
- + übersichtlicher und robuster → weniger fehleranfällig
- + geringerer Wartungsaufwand
- + bessere Skalierbarkeit für steigenden Funktionsumfang.
- + Schnittstelle, die von XML abstrahiert
- Keine Unterstützung der Typsubstitution

# Warum 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?

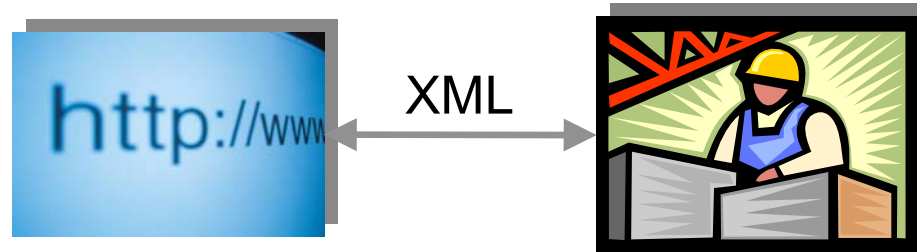
# Typisches E-Business-Projekt



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

- 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.

## Projektaufwand

Phase I

Phase II



> 80%





# Wie geht es weiter?

---

## heutige Vorlesung

- ☑ SAX- und DOM-Parser
- ☑ Vor- und Nachteile von SAX und DOM
- ☑ Schema-Übersetzer

## Übung nächste Woche

- 3. Übung: XML Schema

## Vorlesung nächste Woche

- Transformation von XML-Dokumenten mit XSLT