



Netzprogrammierung XML Dokumente und ihre Verarbeitung

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
<http://www.robert-tolksdorf.de>

1. XML Dokumententypen
2. XML Parser DOM und SAX

oder so?

```
<book>
  <title>My Life and Times</title>
  <authors>
    <author>
      <first>Paul</first>
      <last>McCartney</last>
    </author>
  </authors>
  <date>
    <year>1998</year>
    <month>July</month>
  </date>
  <isbn>94303-12021-43892</isbn>
  <publisher>McMillin Publishing</publisher>
</book>
```

```
<Book>
  <Title>My Life and Times</Title>
  <Author>Paul McCartney</Author>
  <Date>July, 1998</Date>
  <ISBN>94303-12021-43892</ISBN>
  <Publisher>McMillin
  Publishing</Publisher>
</Book>
```

so?

- ⇒ einheitliches Format nötig
- ⇒ Format sollte durch XML-Prozessor validierbar sein

- **prinzipieller Aufbau von Dokumenten:** Welche Elemente/Attribute dürfen wo verwendet werden?

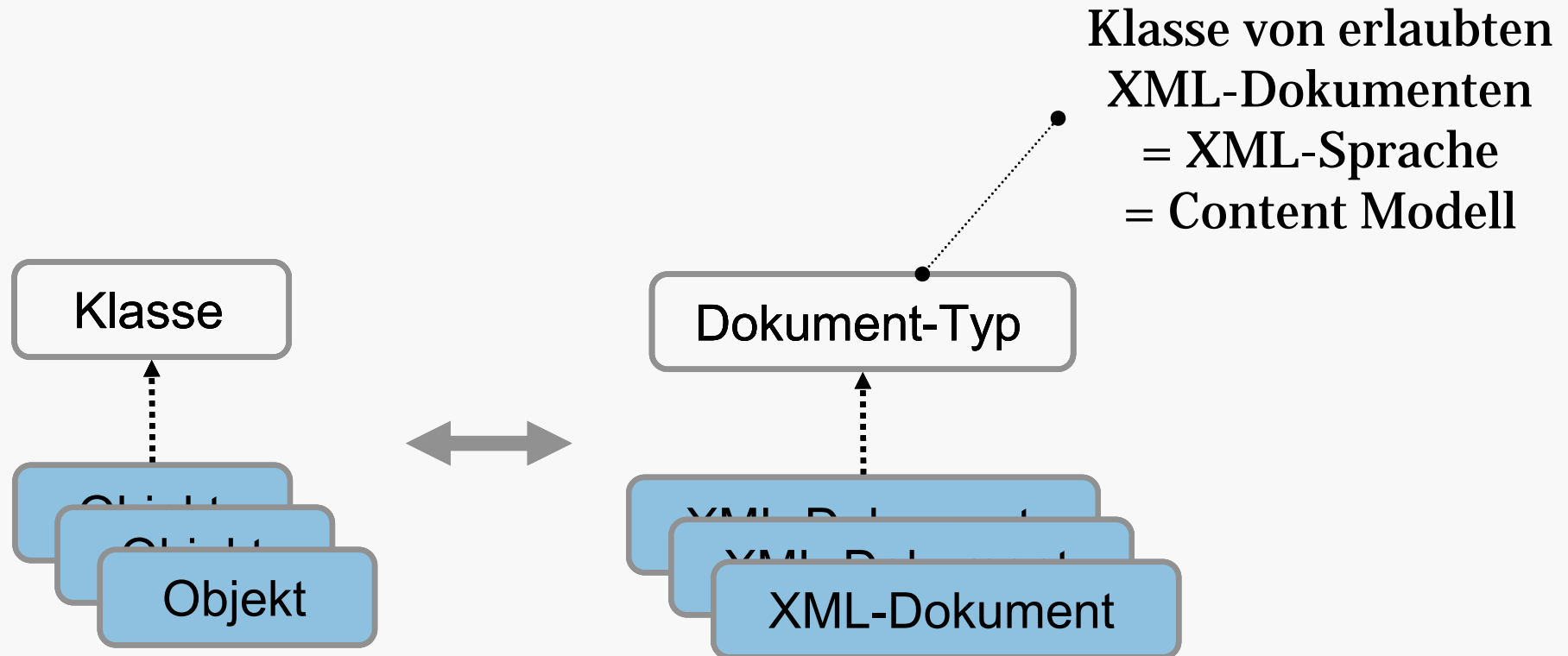
- **Datentypen der Inhalte:** Welche Inhalte sind erlaubt?

```
<Book>
  <Title> PCDATA </Title>
  <Author> PCDATA </Author>
  <Date> PCDATA </Date>
  <ISBN> PCDATA </ISBN>
  <Publisher> PCDATA
</Publisher>
</Book>
```

- konkrete Inhalte werden nicht beschrieben

⇒ Klasse von erlaubten XML-Dokumenten

- auch **Dokument-Typ, XML-Sprache, Anwendung von XML** oder **Content Modelle** genannt



- Dokument-Typ kann mit einer DTD, einem XML-Schema oder ähnlichen Formalismen definiert werden.

Document Type Definitions (DTDs)

Wie sieht eine DTD hierfür aus?

```
<BookStore>
```

```
<Book>
```

```
<Title>My Life and Times</Title>
```

```
<Author>Paul McCartney</Author>
```

```
<Date>July, 1998</Date>
```

```
<ISBN>94303-12021-43892</ISBN>
```

```
<Publisher>McMillin Publishing</Publisher>
```

```
</Book>
```

```
</BookStore>
```

- BookStore soll mindestens ein Buch enthalten.
- ISBN optional
- alle anderen Kind-Elemente obligatorisch

Die DTD für das Beispiel-Dokument

<!ELEMENT BookStore (Book+)>

<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>

<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author (#PCDATA)>

<!ELEMENT Date (#PCDATA)>

<!ELEMENT ISBN (#PCDATA)>

<!ELEMENT Publisher (#PCDATA)>

<!ELEMENT *Name Content-Modell*>

ähnelt einer
regulären
Grammatik

Element-Deklaration

verschiedene Datentypen:

1. **Element**: Element mit speziellen Symbolen + * | ?
2. **#PCDATA**: unstrukturierter Inhalt ohne reservierte Symbole < und &.

<!ELEMENT Title (#PCDATA)>

2. **EMPTY**: leerer Inhalt, Element kann aber Attribute haben

<!ELEMENT br EMPTY> →

3. **ANY**: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)

<!ELEMENT title ANY>

Datentypen wie **INTEGER** oder **FLOAT** stehen nicht zur Verfügung.

`<!ELEMENT BookStore (Book+)>`

`<BookStore>
 <Book>...</Book>
 <Book>...</Book>
</BookStore>`

- **+** bezeichnet n Wiederholungen mit $n > 0$.
- ***** bezeichnet n Wiederholungen mit $n \geq 0$.
- BookStore hat mindestens ein Kind-Element Book.
- Außer Book darf BookStore keine anderen Kind-Elemente haben.

```
<!ELEMENT BookStore (Book | (Book, BookStore))>
```

- Bookstore besteht aus genau einer der folg. Alternativen:
 - genau ein Kind-Element Book
 - zwei Kind-Elemente: Book und BookStore
- | bezeichnet **Auswahl**: genau eine der beiden Alternativen
- , bezeichnet **Sequenz** von Elementen.
- Beachte: Rekursive Deklaration nicht äquivalent zur vorherigen, iterativen Definition!

Rekursive vs. iterative Deklaration

```
<BookStore>
```

```
  <Book>...</Book>
```

```
  <Book>...</Book>
```

```
  <Book>...</Book>
```

```
</BookStore>
```

BookStore mit 3 Büchern

```
<!ELEMENT BookStore (Book+)>
```

```
<BookStore>
```

```
  <Book>...</Book>
```

```
  <BookStore>
```

```
    <Book>...</Book>
```

```
    <BookStore>
```

```
      <Book>...</Book>
```

```
    </BookStore>
```

```
  </BookStore>
```

```
</BookStore>
```

BookStore mit 3 Büchern

```
<!ELEMENT BookStore (Book | (Book, BookStore))>
```

```
<!ELEMENT BookStore (Book+)>
```

```
<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
```

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

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

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

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

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

<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>

- Title, Author, Date, ISBN und Publisher (in dieser Reihenfolge) Kind-Elemente von Book.
- außer diesen keine anderen Kind-Elemente
- ? bedeutet optional

```
<Book>
  <Title>...</Title>
  <Author>...</Author>
  <Date>...</Date>
  <ISBN>...</ISBN>
  <Publisher>...</Publisher>
</Book>
```

```
<!ELEMENT BookStore (Book+)>
```

```
<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
```

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

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


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

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

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

Deklaration von Title etc.

```
<!ELEMENT Title (#PCDATA)>  
<!ELEMENT Author (#PCDATA)>  
<!ELEMENT Date (#PCDATA)>  
<!ELEMENT ISBN (#PCDATA)>  
<!ELEMENT Publisher (#PCDATA)>
```



```
<Title>My Life and Times</Title>  
<Author>Paul McCartney</Author>  
<Date>July, 1998</Date>  
<ISBN>94303-12021-43892</ISBN>  
<Publisher>McMillin Publishing</Publisher>
```

Verschachtelungen

- (fast) beliebige Verschachtelung von Sequenz, Auswahl
|, ?, *, + und Rekursion erlaubt
- Beispiel:

```

<!ELEMENT Chap (Title (Para | Chap)+)>
<!ELEMENT Para ANY>
<!ELEMENT Title (#PCDATA)>
  
```

```

<Chap>
  <Title>Kap1</Title>
  <Para>Ein Absatz</Para>
  <Chap>
    <Title>Kap1.1</Title>
    <Para>...</Para>
  </Chap>
  <Para>...</Para>
  <Chap>
    <Title>Kap1.2</Title>
    <Para>...</Para>
  </Chap>
</Chap>
  
```


Deklaration von Attributen

```
<!ATTLIST BookStore  
version CDATA #IMPLIED>
```

```
<!ATTLIST Name  
AttrName1 AttrTyp1 Attrbeschr1  
AttrName2 AttrTyp2 Attrbeschr2  
>
```

Attribut-
Deklarationen

```
<!ATTLIST BookStore  
    version CDATA #IMPLIED>
```



```
<BookStore version="1.0">  
    ...  
</BookStore>
```

- Element BookStore hat Attribut version.
- Außer version hat BookStore keine weiteren Attribute.
- **CDATA**: Attribut-Wert = String ohne <, & und ' bzw. "
- Beachte: nicht verwechseln mit <![CDATA[...]]>
- daher Entity References für <, & und ' bzw. " verwenden

```
<!ATTLIST Author  
    gender (male | female) "female">
```

- hier statt CDATA **Aufzählungstyp**:
- Attribut gender hat entweder Wert male oder female.
- "female" ist Standard-Wert von gender.

Zusätzlich zu CDATA (Strings) und Aufzählungstypen:

- **NMTOKEN**: String, der Namenskonventionen von XML entspricht
- **ID**: eindeutiger Bezeichner, der Namenskonventionen von XML entspricht
- **IDREF**: Referenz auf einen eindeutigen Bezeichner

```
<!ATTLIST Author  
    key ID #IMPLIED  
    keyref IDREF #IMPLIED>
```

- Wert des Attributes key muss eindeutig sein:
Zwei Attribute vom Typ ID dürfen niemals gleichen Wert haben.
- Wert des Attributes keyref muss gültige Referenz sein:
Wert von keyref muss Wert eines Attributes vom Typ ID sein.

Deklaration von Attributen

```
<!ATTLIST BookStore  
version CDATA #IMPLIED >
```

```
<!ATTLIST Name  
AttrName1 AttrTyp1 Attrbeschr1  
AttrName2 AttrTyp2 Attrbeschr2  
>
```

Attribut-
Deklarationen

```
<!ATTLIST BookStore  
        version CDATA #FIXED "1.0">
```

- **#FIXED**: Attribut hat immer den gleichen Wert.
- **#IMPLIED**: Attribut optional
- **#REQUIRED**: Attribut obligatorisch

- **"1.0"**: Standard-Wert des Attributes

```
<BookStore>
  <Book>
    <Title>Text</Title>
    <Author key="k1">Text</Author>
    <Date>Text</Date>
    <Publisher pkey="p1">Text</Publisher>
  </Book>
  <Book>
    <Title>Text</Title>
    <Author keyref="k1"/>
    <Date>Text</Date>
    <Publisher pkey="p1">Text</Publisher>
  </Book>
</BookStore>
```

Wert **k1** muss eindeutig sein: kein anderes Attribut vom Typ ID darf diesen Wert haben.

Referenz **k1** muss existieren: ein Attribut vom Typ ID muss den Wert **k1** haben.

- direkt nach XML-Deklaration einfügen:
 - vollständige DTD oder
 - Verweis auf externe DTD

`<!DOCTYPE Wurzel-Element SYSTEM "DTD">`

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE BookStore [
```

```
<!ELEMENT BookStore (Book+)>
```

```
<!ELEMENT Book (Title, Author, Date, ISBN?, Publisher)>
```

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

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

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

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

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

```
]>
```

```
<BookStore>
```

```
...
```

```
</BookStore>
```

`<!DOCTYPE Wurzel-Element SYSTEM "DTD">`

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE BookStore SYSTEM "Bookstore.dtd">
```

```
<BookStore>
```

```
...
```

```
</BookStore>
```

Dokument-Typ
Deklaration

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE Book SYSTEM "Bookstore.dtd">
```

```
<Book>
```

```
...
```

```
</Book>
```

wohlgeformt (well formed)

- XML-Dokument entspricht Syntaxregeln von XML

zulässig (valid) bzgl. einer DTD

1. Wurzel-Element des XML-Dokumentes ist in der DTD deklariert und
2. Wurzel-Element hat genau die Struktur, wie sie in der DTD festgelegt ist.

Nachteile von DTDs

- keine XML-Syntax, eigener Parser nötig
- nur sehr wenige Datentypen, insbesondere für Element-Inhalte
- keine eigenen Datentypen definierbar
- keine Namensräume:
DTDs können nur dann kombiniert werden, wenn es keine Namenskonflikte gibt!
- keine Vererbungshierarchien

- Sequenzen einfach zu definieren:

```
<!ELEMENT Book (Title, Author)>
```

⇒ starre Struktur in XML-Dokumenten

- soll Reihenfolge der Kind-Elemente egal sein, müssen alle Permutationen explizit aufgezählt werden:

```
<!ELEMENT Book ((Title, Length) | (Length, Title))>
```

- nicht praktikabel: bei n Kind-Elementen $n!$ Permutationen

DTD deklariert das erlaubte Vokabular

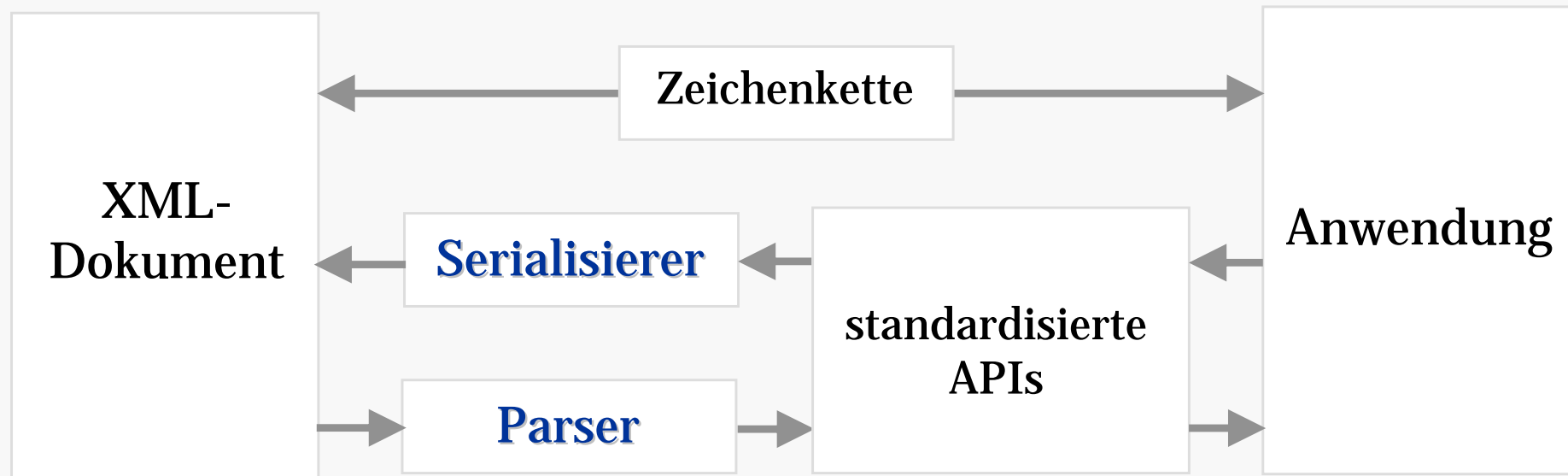
DTD definiert für jedes Element ein Content-Modell

- Content-Modell legt fest:
 - Elemente oder Daten
 - Reihenfolge & Anzahl von Elementen/Daten innerhalb eines Elements
 - Pflicht oder optionale Elemente

DTD deklariert für jedes Element eine Menge von erlaubten Attributen



XML-Parser



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?

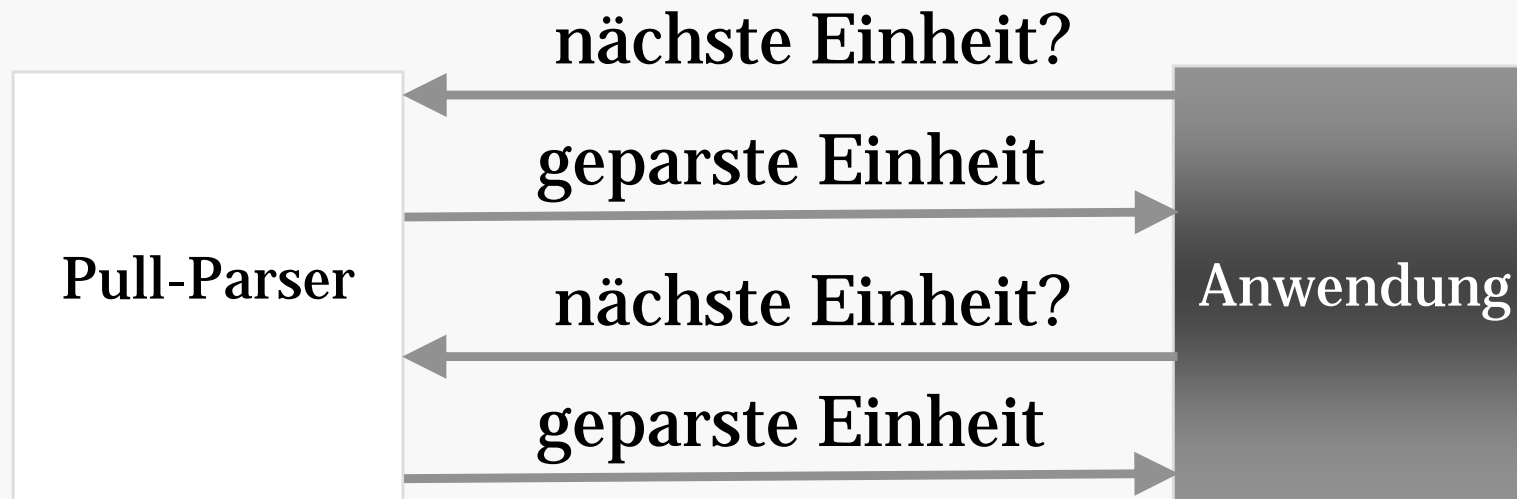
Pull- vs. Push-Parser

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

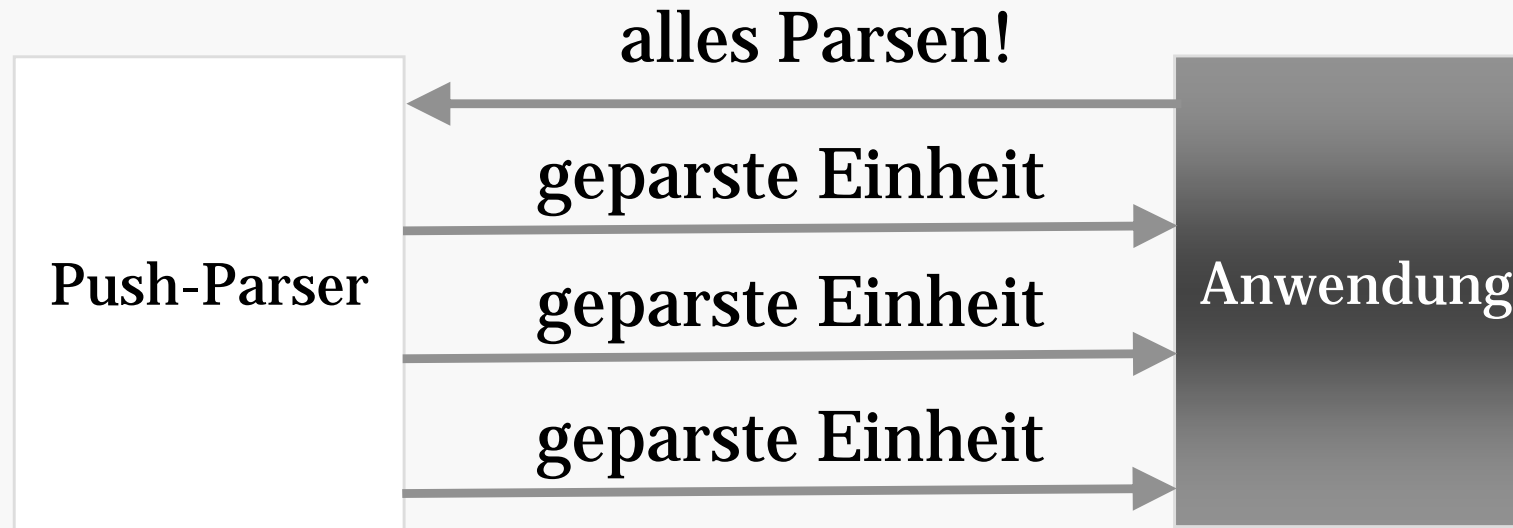
Einschritt- vs. Mehrschritt-Parser

- Wird das XML-Dokument in einem Schritt vollständig geparkt oder Schritt für Schritt?

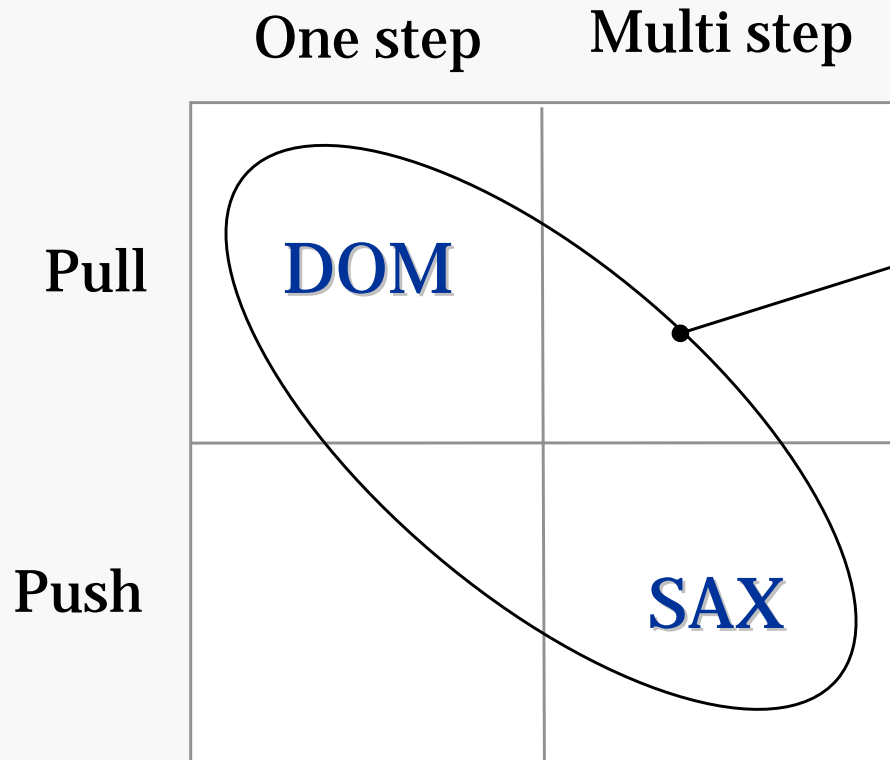
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden



- 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

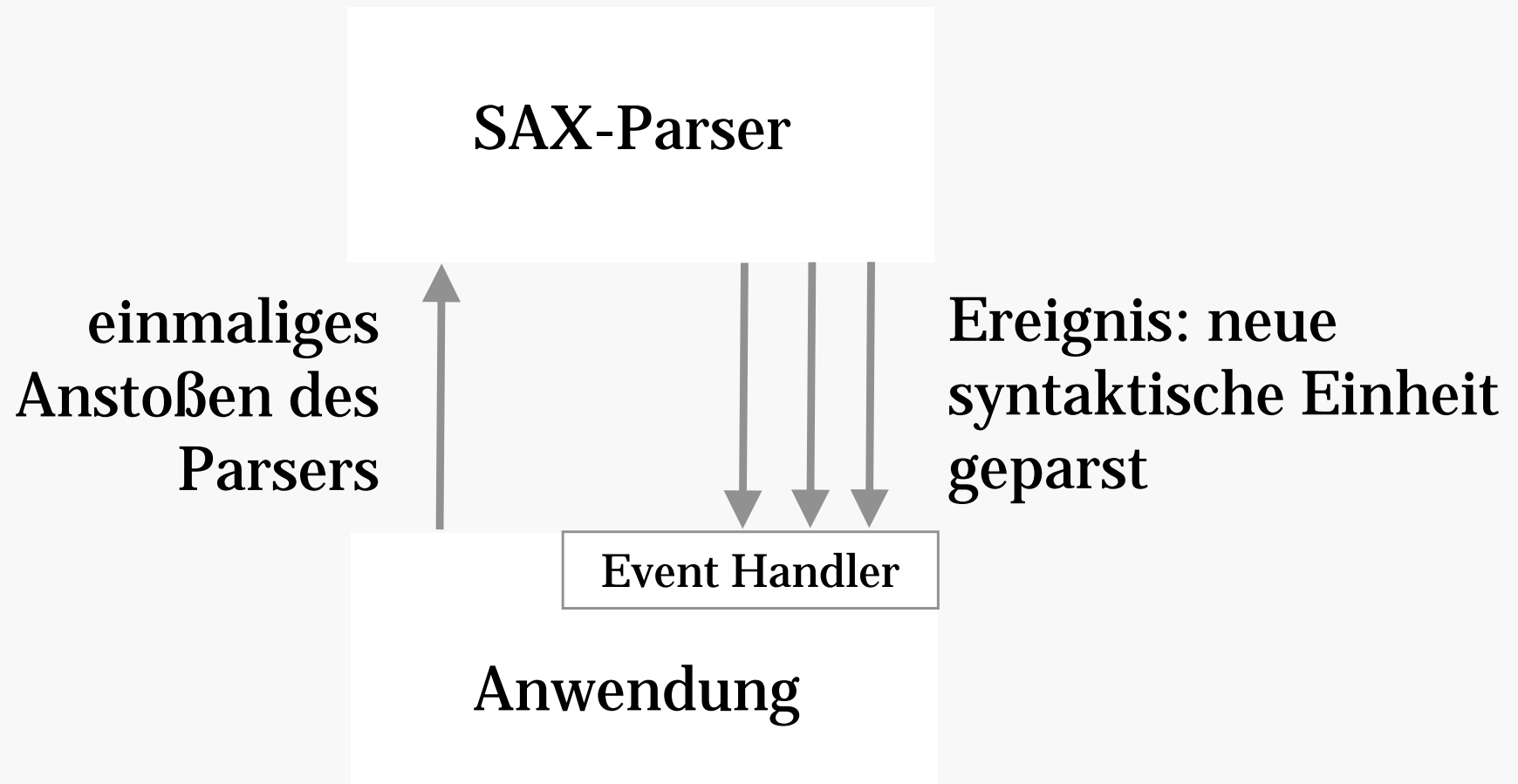
- **DOM**: Document Object Model
- **SAX**: Simple API for XML



SAX-Parser

- **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





```
<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!

- 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

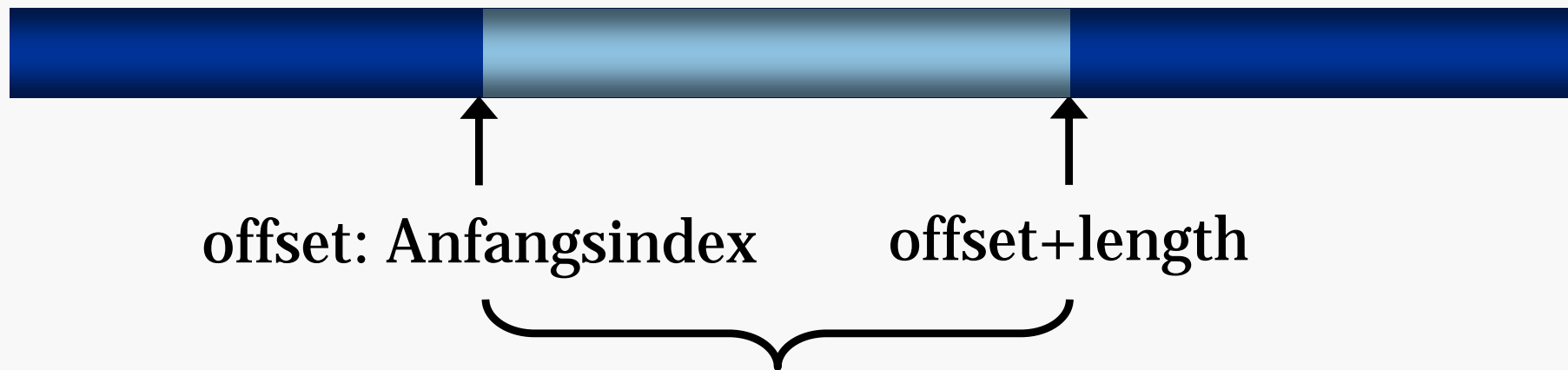
- **startDocument** – einmalig zu Beginn des Parsens
- **endDocument** – einmalig am Ende des Parsens aufgerufen
- **startPrefixMapping** – wenn eine Präfixbindung für einen Namensraum beginnt
- **endPrefixMapping** – wenn eine Präfixbindung für einen Namensraum endet
- **startElement** – wenn ein Starttag 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

```
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

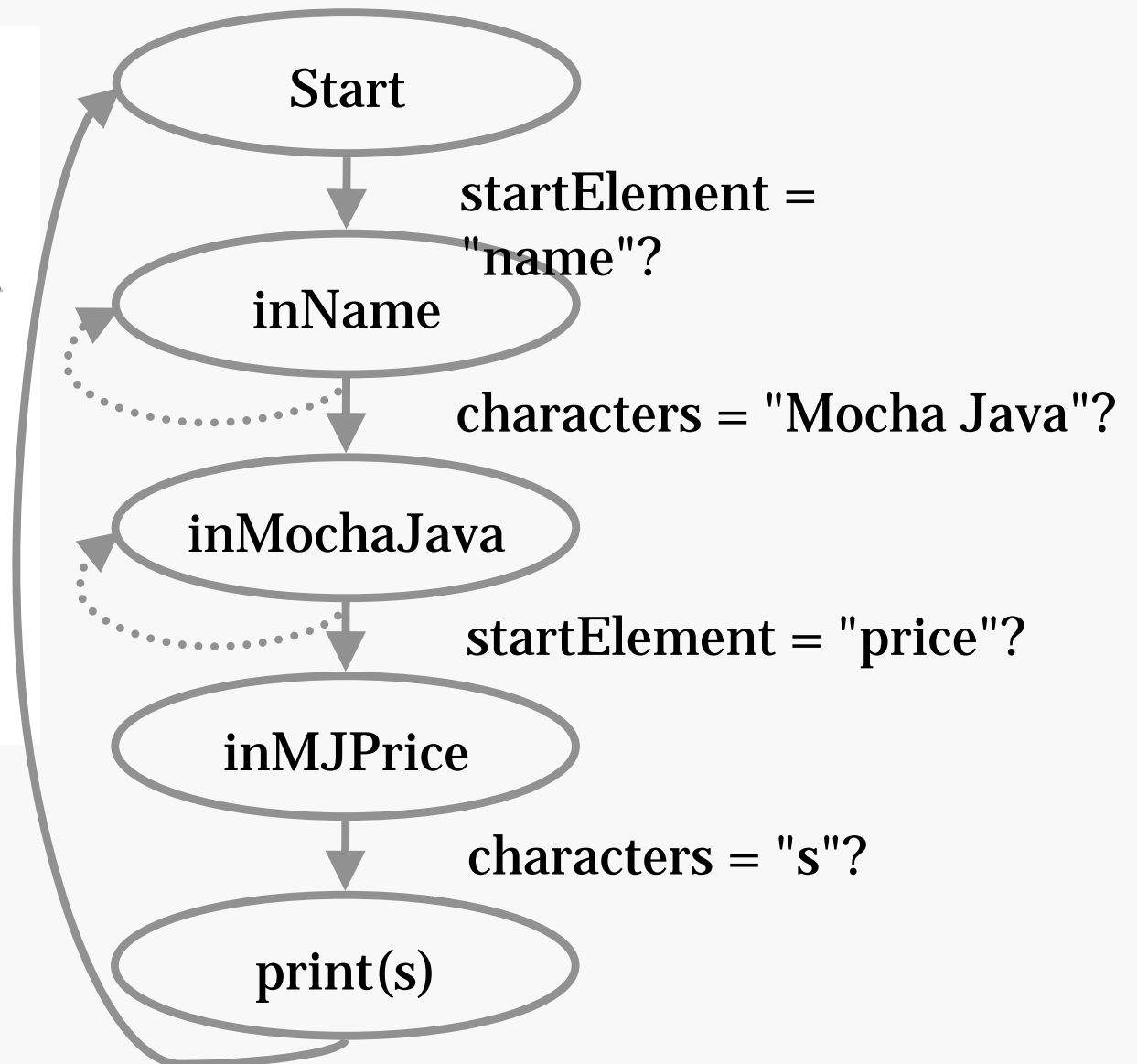
- Entwurfsmuster aus „Design Patterns“ von Gamma, Helm, Johnson, Vlissides (1995)
- liefert ein Objekt
- Objekt ist Instanz einer abstrakten Klasse oder einem Interface.
- abstrakte Klasse / Interface von mehreren Klassen implementiert
- Beispiel: `Iterator i = list.iterator();`
- Beispiel: `SAXParser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

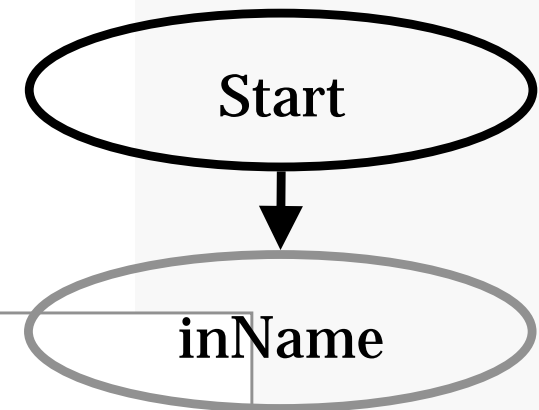
```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){ inName = true; }
else if (elementName.equals("price") && inMochaJava ){
    inMJPrice = true;
    inMochaJava = false; } }
```

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

Start

- Anfangszustand
- keine eigene Zustandsvariable
- alle Zustandsvariablen = false

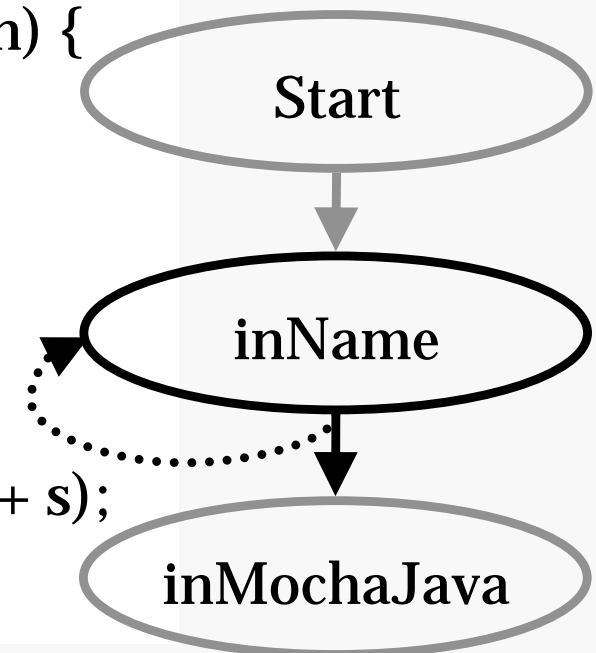


inName: Auf "Mocha Java" warten

```
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>
```

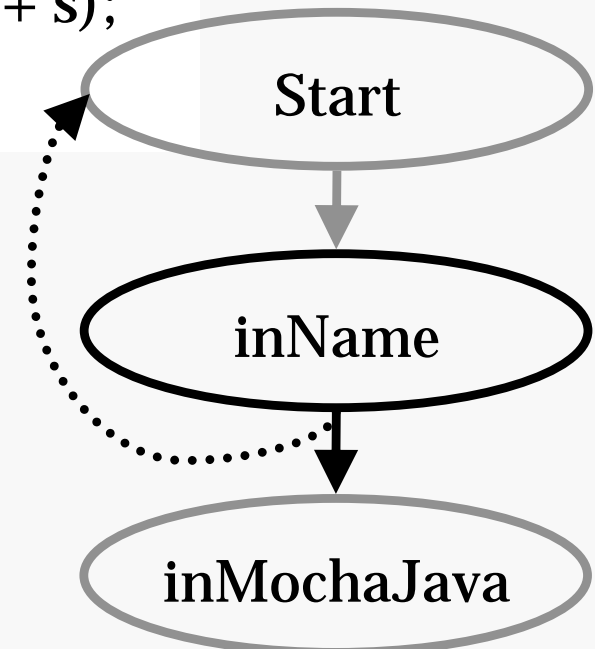
```
public void characters(char [] buf, int offset, int len) {  
    String s = new String(buf, offset, len);  
    if (inName && s.equals("Mocha Java")) {  
        inMochaJava = true;  
        inName = false; }  
    else if (inMJPrice) {  
        System.out.println("The price of Mocha Java is: "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

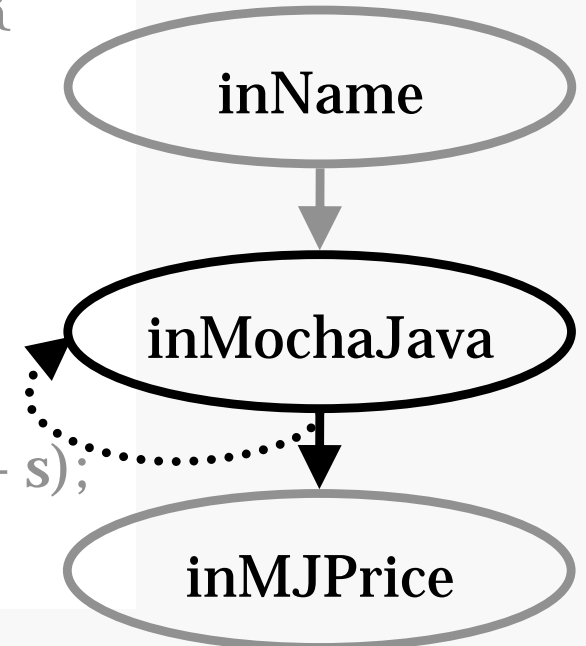


inMochaJava: Auf <price> warten

```
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;  } }
```

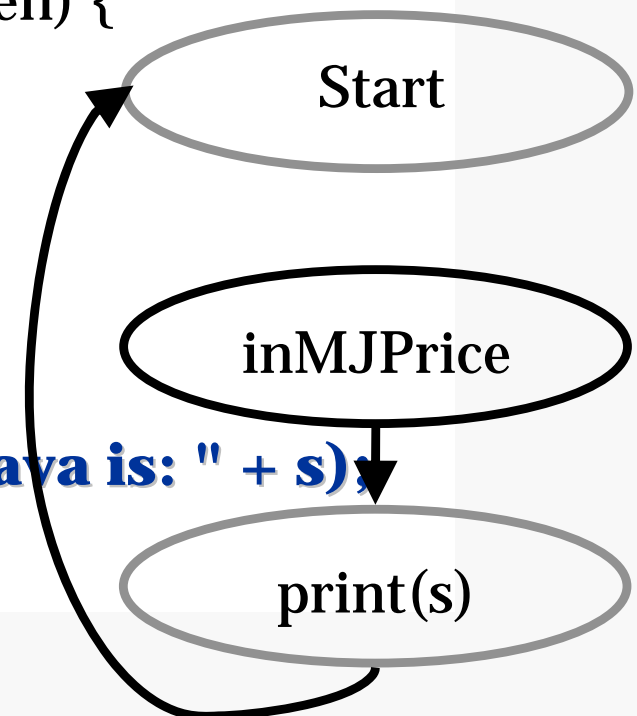


inMJPrice: Preis ausgeben

```
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;  } }
```

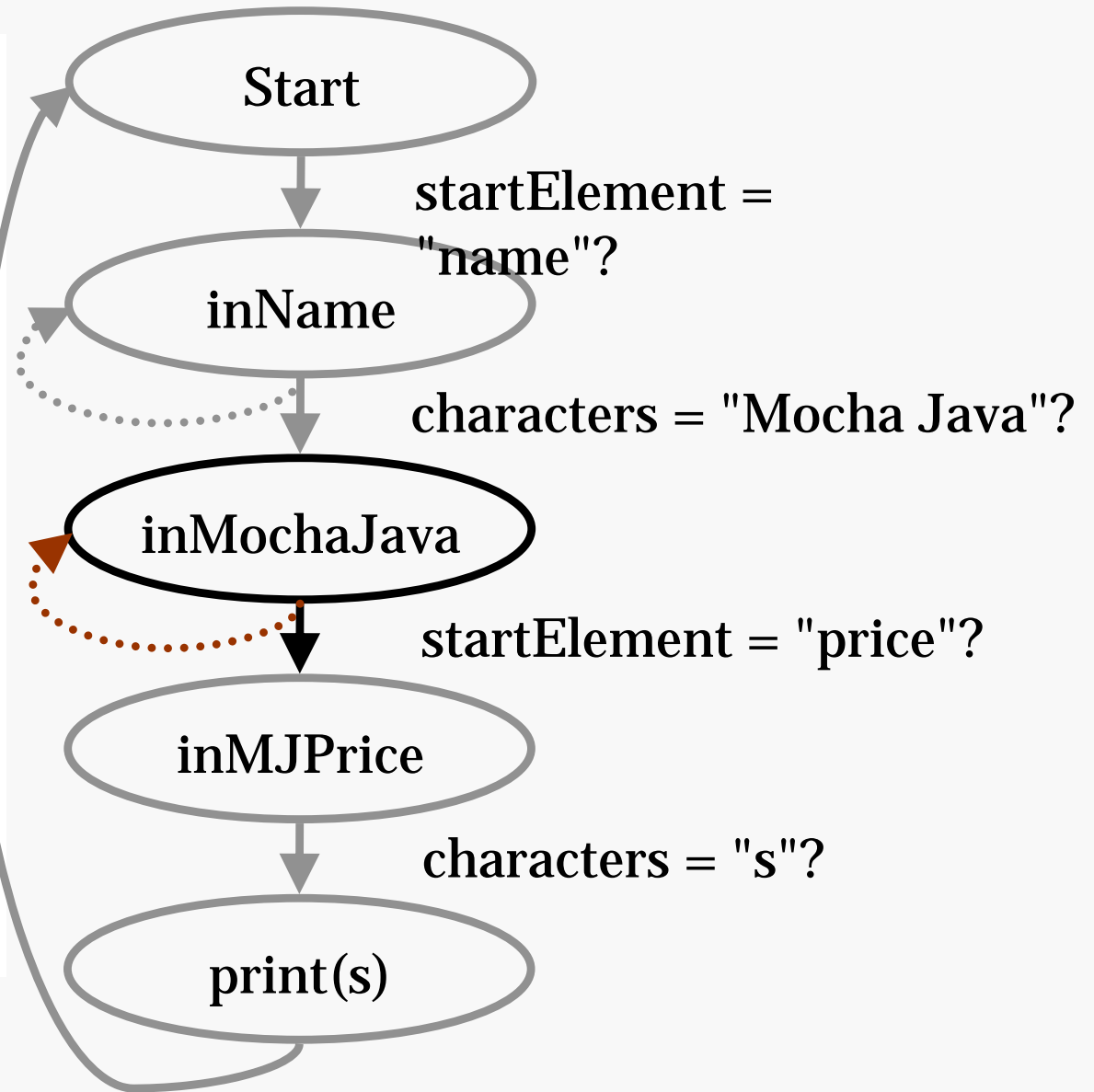


```
<priceList>  
<coffee>  
  <name>  
    Mocha Java  
  </name>  
  <name>  
    MS Java  
  </name>  
  <price>  
    11.95  
  </price>  
</coffee>  
</priceList>
```

Mocha Java

MS Java

11.95

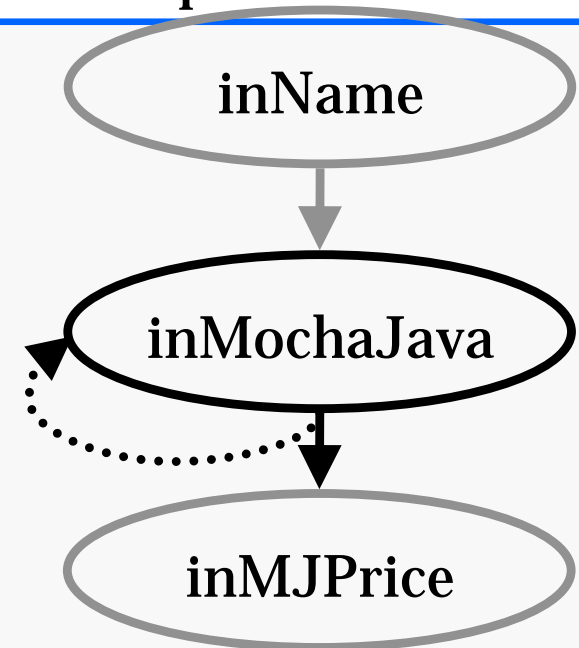


Fehlerbehandlung

```
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!



- 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.

- + 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

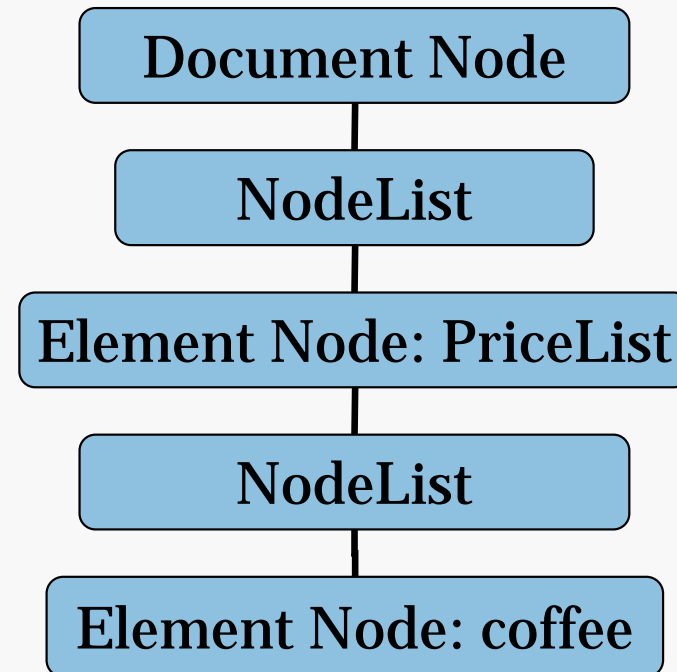


DOM-Parser



- 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**

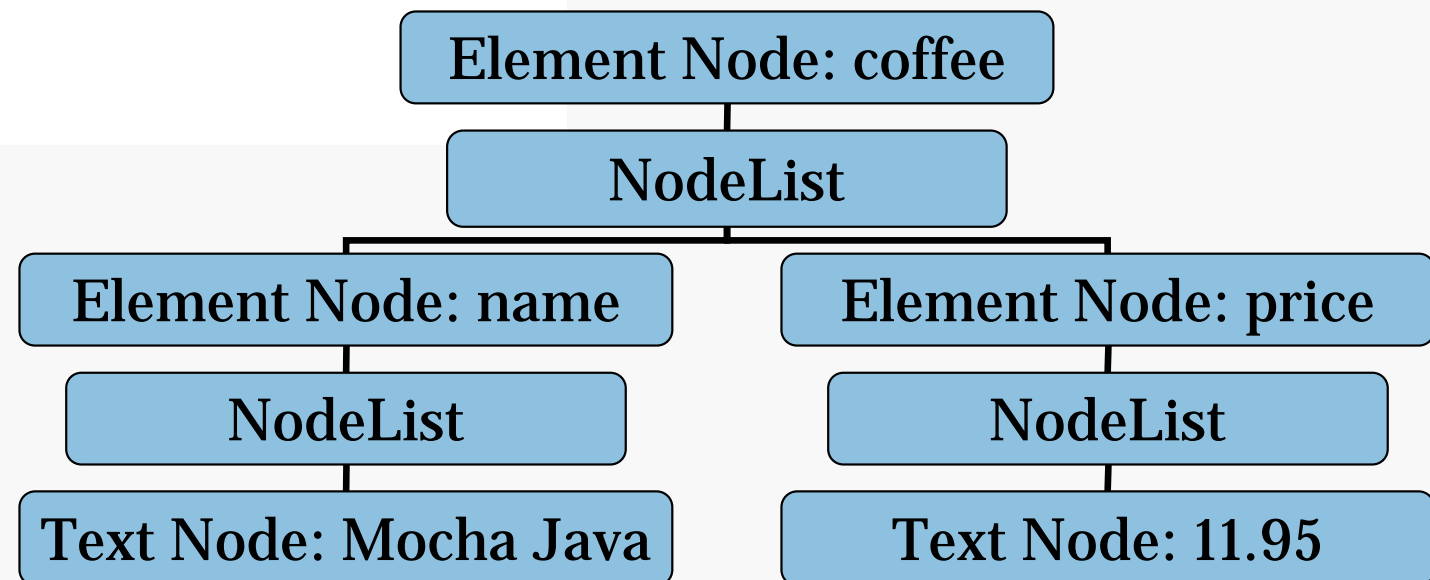
```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

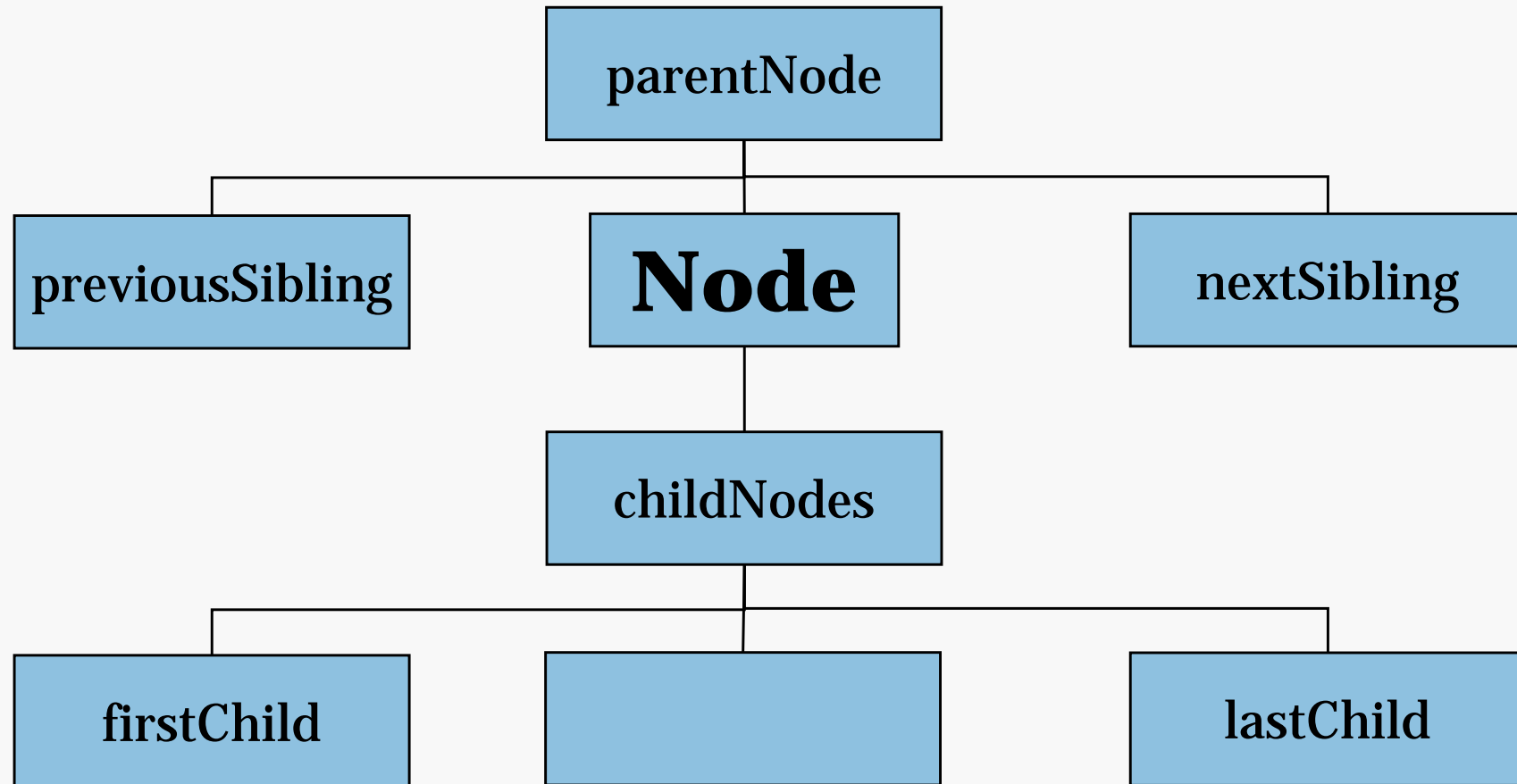


- Beachte: Dokument-Wurzel (Document Node) \neq priceList
- **Document Node**: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer Node**List** als Kind

```
<?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: `getElementByTagName`

```
<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");
```

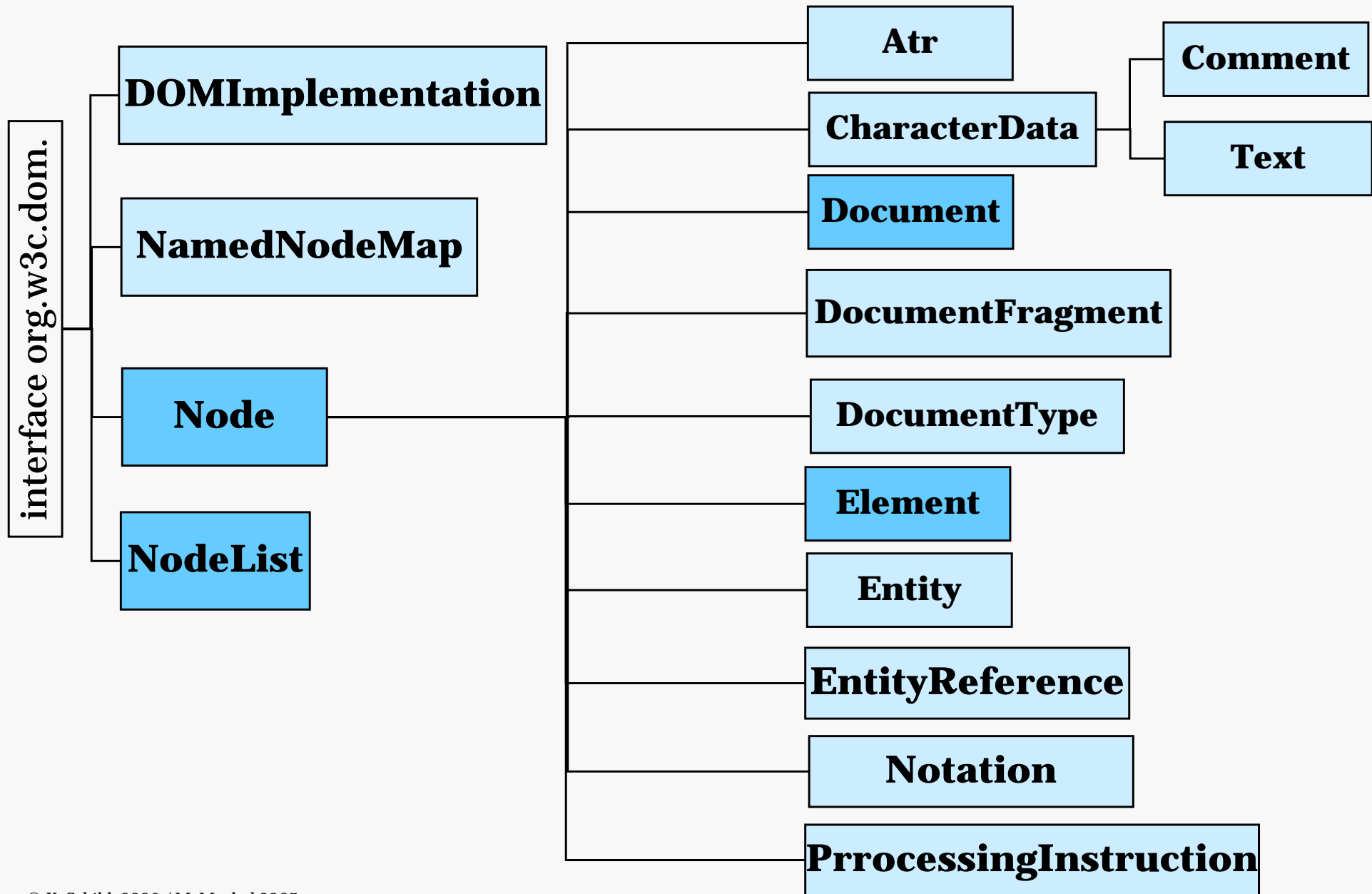
- DOM-Parser hat Methode parse().
- liefert in einem Schritt kompletten DOM-Parse-Baum

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 Hierarchie



- 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 getElementElement()***
→ 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; } }
```

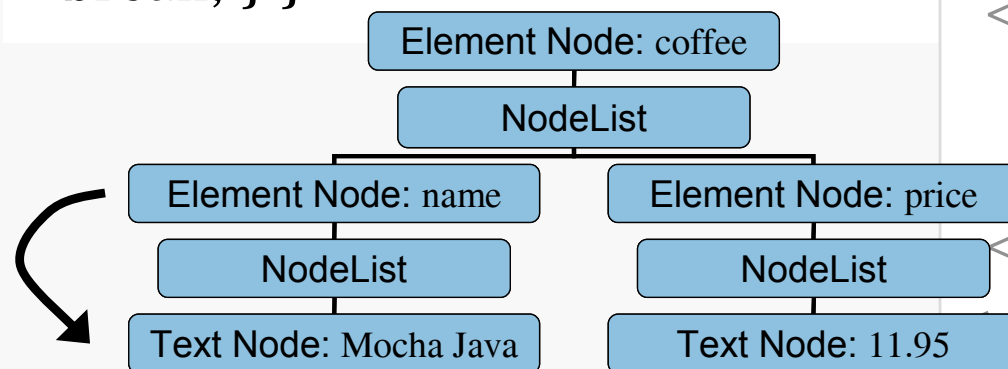
firstChild

```
<?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; } }
    
```



```

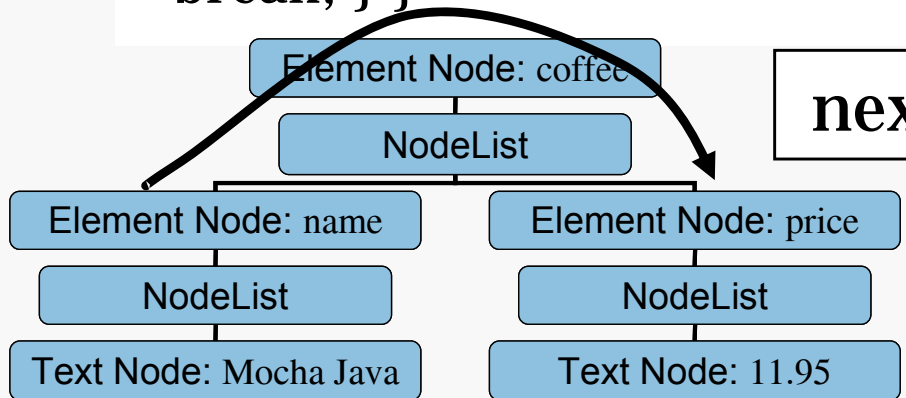
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
    
```

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!


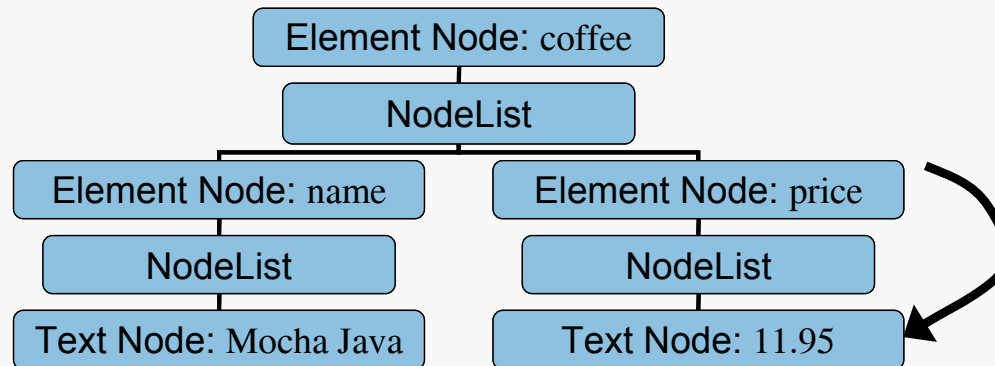
```

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

- 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

SAX oder DOM?

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