

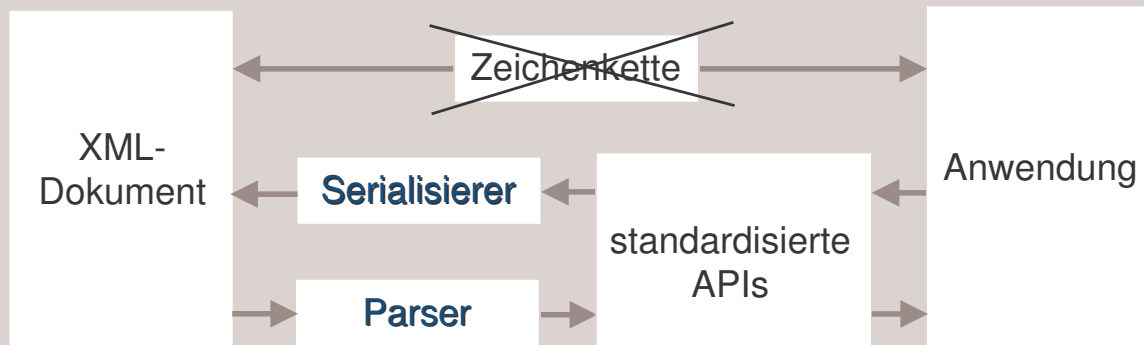
---

# XML-Parser

## Heutige Vorlesung

---

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



## Parser

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

## Serialisierer

- generiert aus Datenstruktur XML-Dokument

# Kategorien von Parser

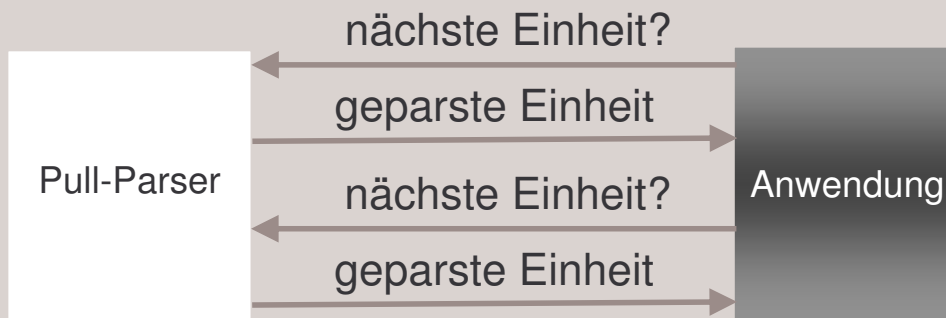
## Pull- vs. Push-Parser

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

## Einschritt- vs. Mehrschritt-Parser (*one step vs. multi step*)

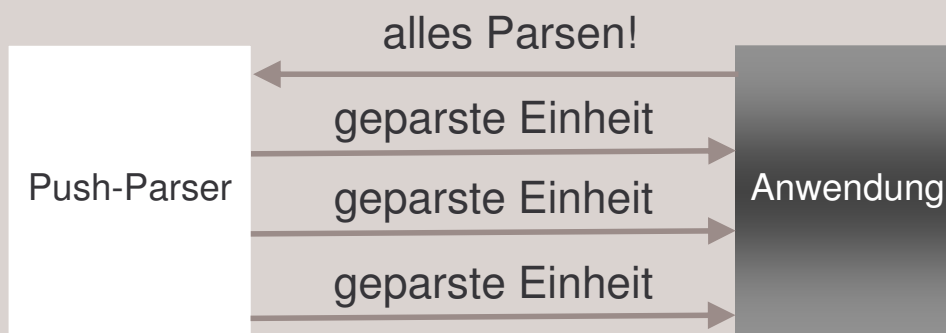
- Wird das XML-Dokument in einem Schritt vollständig geparkt oder Schritt für Schritt?
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden

# Pull-Parser

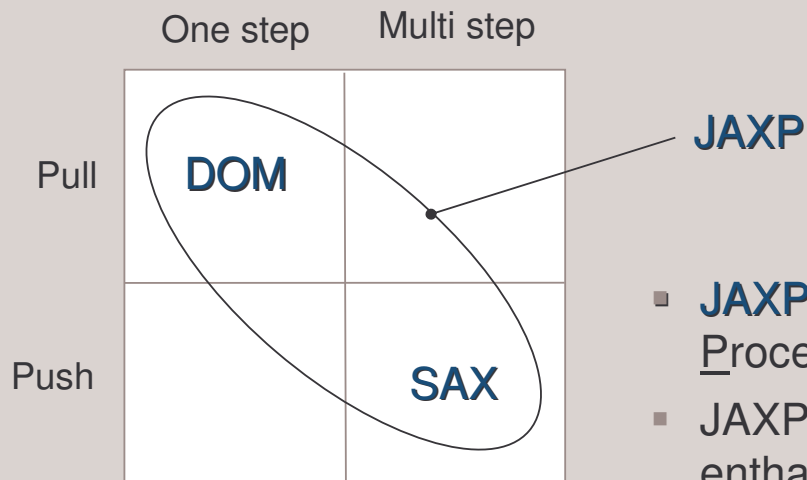


- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: „Pull“ bezieht sich auf die Perspektive der Anwendung.

# Push-Parser



- Parser hat Kontrolle über das Parsen.
- Sobald der Parser eine syntaktische Einheit analysiert hat, übergibt er die entsprechende Analyse.
- Beachte: „Push“ bezieht sich wiederum auf die Perspektive der Anwendung.



- **JAXP**: Java API for XML Processing
- JAXP 1.3 in J2SE 5.0 enthalten

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

# SAX-Parser

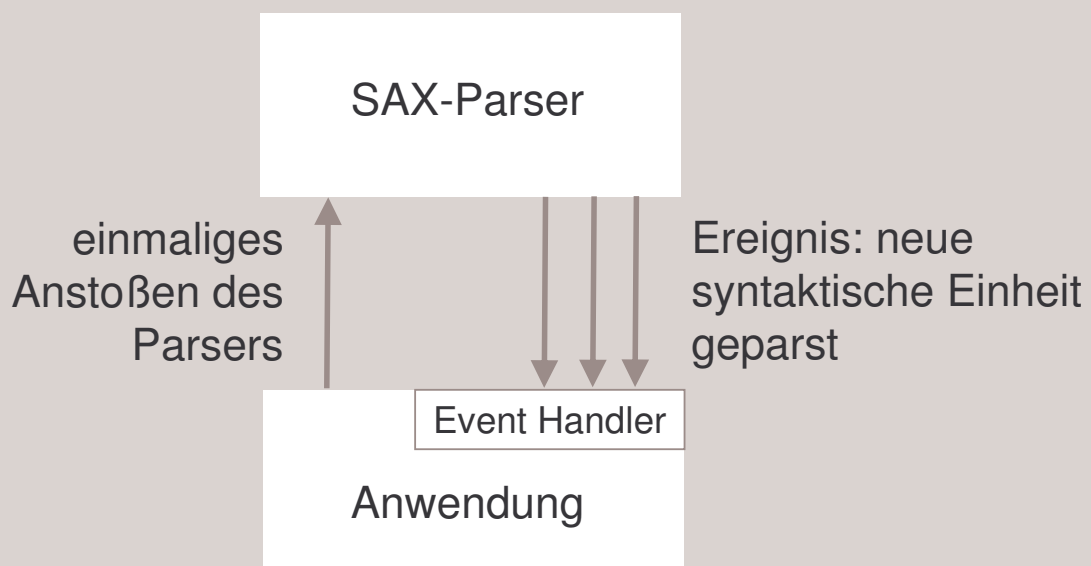
# SAX: Simple API for XML



- Mehrschritt-Push-Parser für XML
  - standardisiertes API
  - ursprünglich nur Java-API
  - inzwischen werden aber auch viele andere Sprachen unterstützt: C, C++, VB, Pascal, Perl
  - kein W3C-Standard, sondern *de facto* Standard
- <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 geparkt 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 aber überschrieben werden

## startElement



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

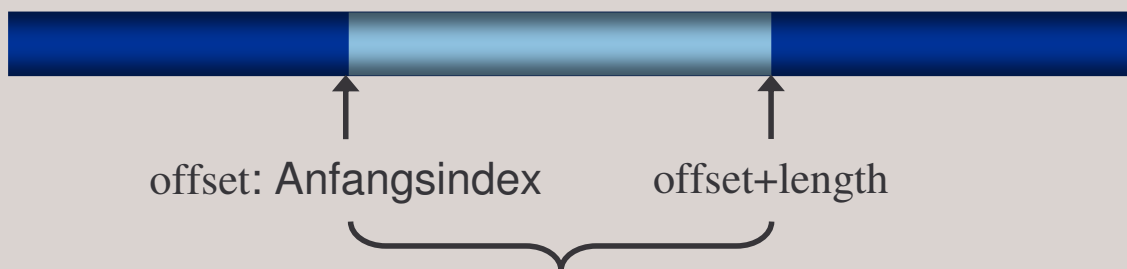
- **uri**: Namensraum-Bezeichner oder leerer String
- **localName**: lokaler Name ohne Präfix oder leerer String
- **qName**: Name mit Präfix oder leerer String
- **attributes**: zu dem Element gehörige Attribute
- Attribute können über ihre Position (Index) oder ihren Namen zugegriffen werden
- **endElement**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

## 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

# Exkurs: Factory Method

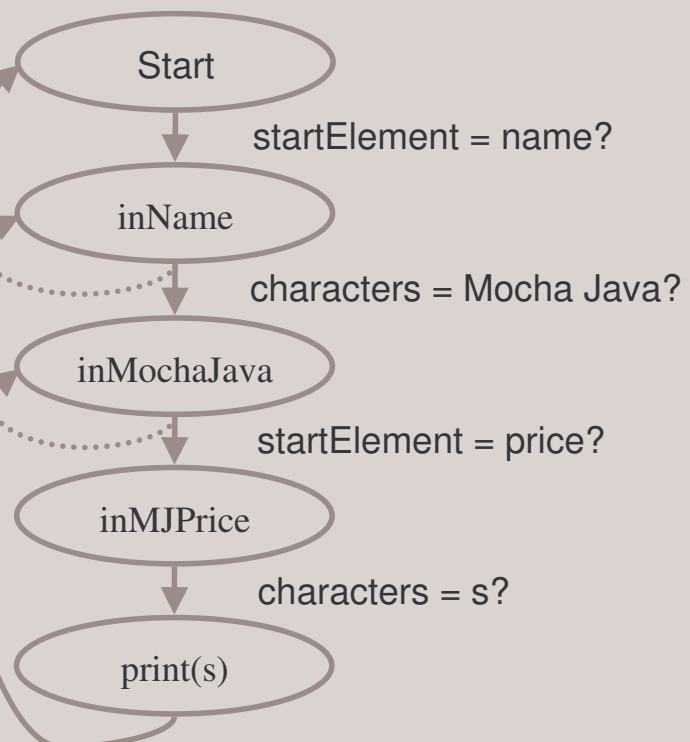


- Entwurfsmuster aus „Design Patterns“ von Gamma, Helm, Johnson, Vlissides (1995)
- liefert ein Objekt
- Objekt ist Instanz einer abstrakten Klasse oder einem Interface.
- abstrakte Klasse / Interface von mehreren Klassen implementiert
- Beispiel: Iterator `i = list.iterator()`;
- Beispiel: SAXParser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

# Funktionsweise 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;  } }  
}
```

## Start: Auf <name> warten



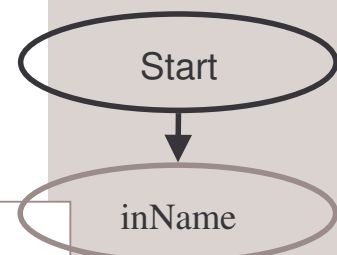
```
public void startElement(..., String elementName, ...){  
    if (elementName.equals("name")){    inName = true;  }  
    else if (elementName.equals("price") && inMochaJava ){  
        inMJPrice = true;  
        inMochaJava = false;  } }  
}
```

```
<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 price of Mocha Java is: " + s);  
        inMJPrice = false;  
    }  
}
```

**Start**

- Anfangszustand
- keine eigene Zustandsvariable
- alle Zustandsvariablen = false



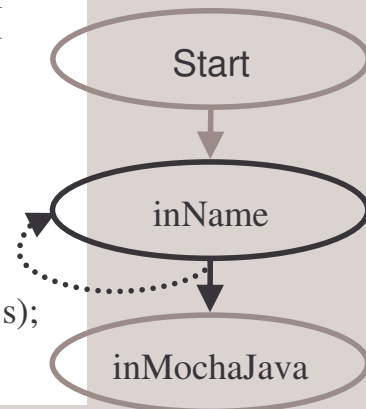
# inName: Auf "Mocha Java" warten



```
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;  } }  
}
```

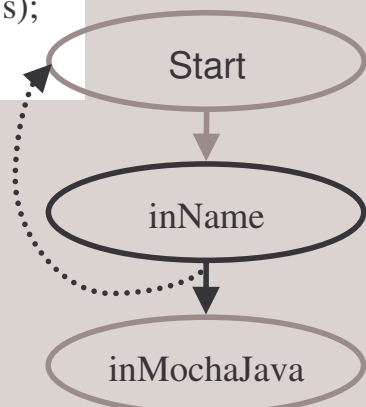


# Eine bessere Alternative



```
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>
```



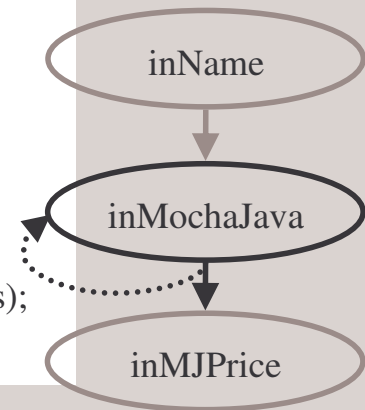
# inMochaJava: Auf <price> warten



```
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;  }  }
```



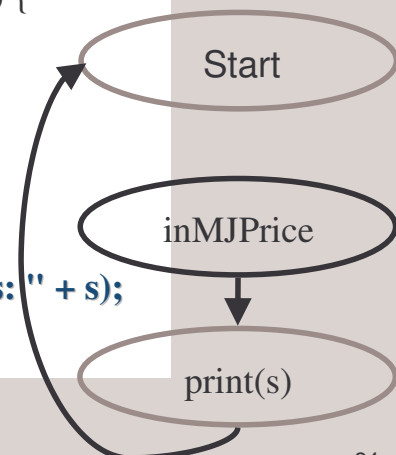
# inMJPrice: Preis ausgeben



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

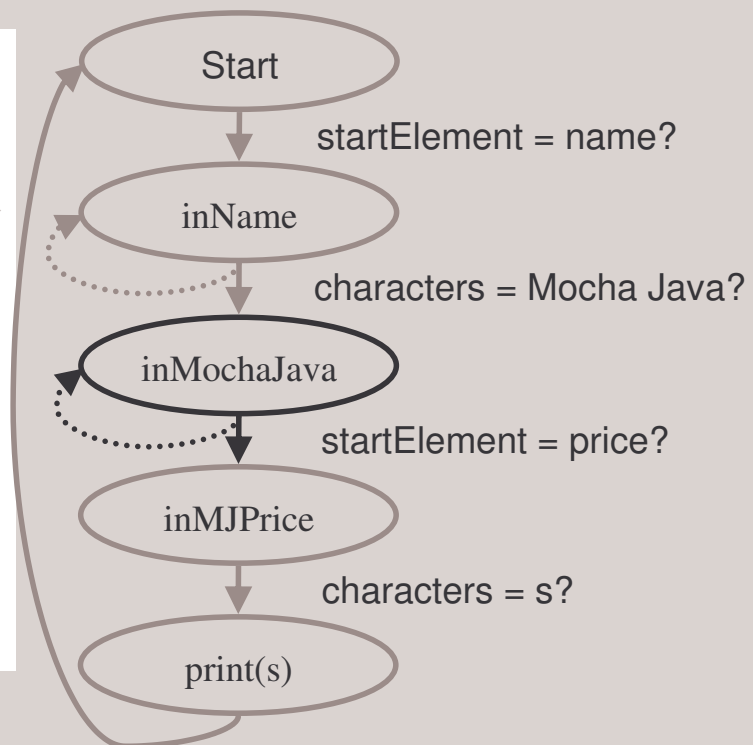
```
<name>Mocha Java</name>  
<price>11.95</price>
```

```
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>
    
```



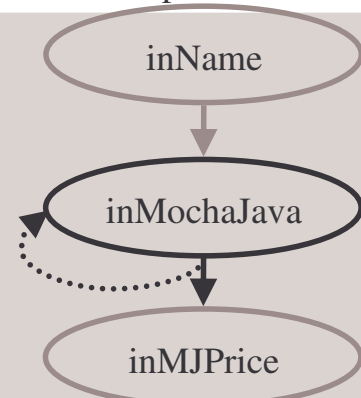
```

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

```

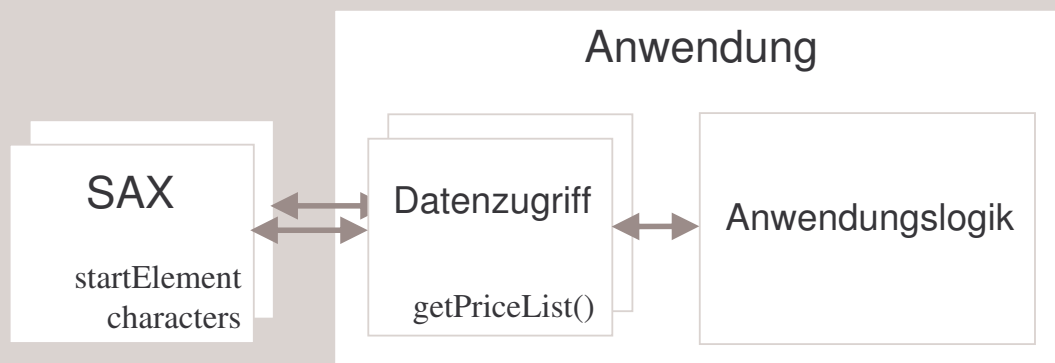
<name>Mocha Java</name>
<name>MS Java</name>
<price>11.95</price>
    
```

- 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



- Immer 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)

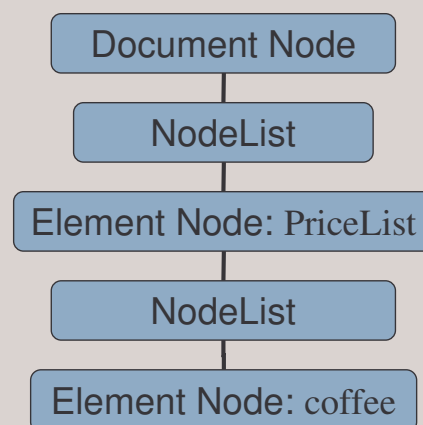


- genau genommen *kein* Parser, sondern abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- W3C-Standard
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis aber Einschritt-Pull-Parser

## DOM-Parse-Bäume



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



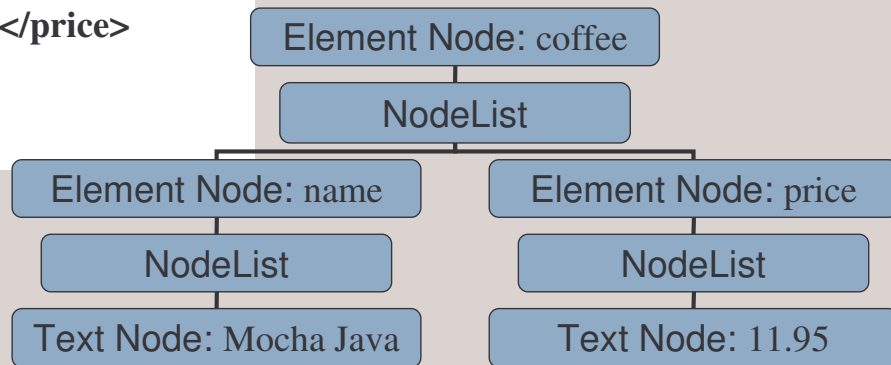
- Beachte: Dokument-Wurzel (Document Node)  $\neq$  priceList
- **Document Node**: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer NodeList als Kind

# Rest des Parse-Baumes

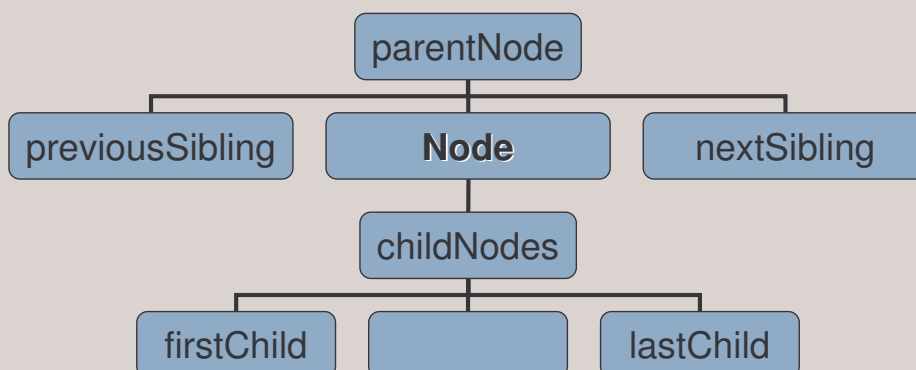


```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

- Beachte: PCDATA wird als eigener Knoten dargestellt.



# Navigationsmodell



- Direkter Zugriff über Namen auch möglich:  
getElementsByTagName

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

- Aufgabe: Gib des 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

# Gib mir die Liste aller 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 alle 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-Element 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()  
    break; } }
```

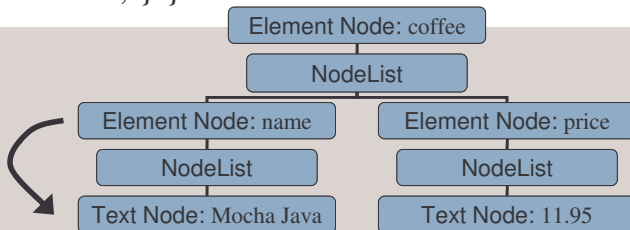
firstChild

```
<?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(
            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(
            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.getElementsById("coffeeList");
for (int i=0; i < coffeeNodes.getLength(); i++) {
    Node 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; } }
```

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

firstChild

# Vor- und Nachteile von DOM



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

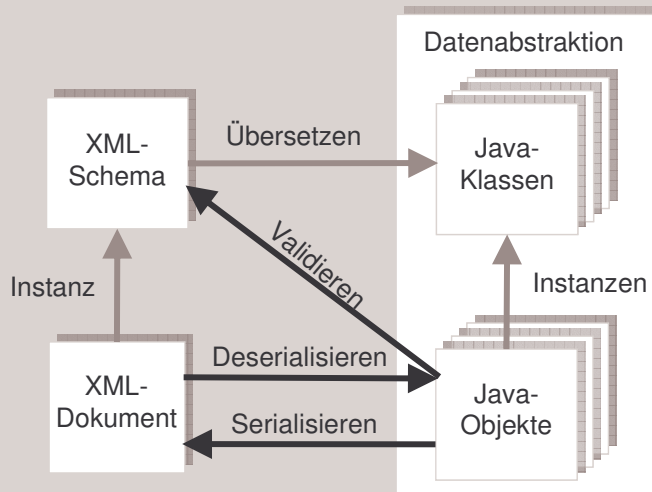
## SAX

- 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

# 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 Web Services Developer Pack

## Beispiel

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

PriceList-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(); }
```

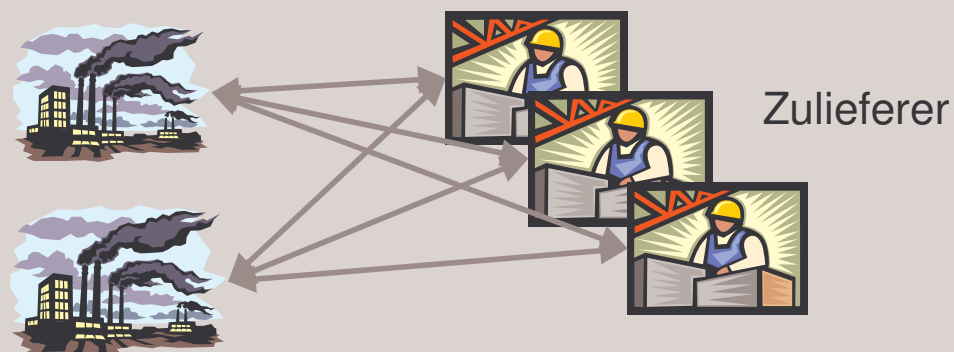
# Warum noch XML lernen?



- 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.

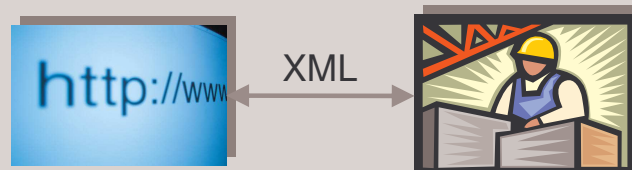
## Typisches E-Business-Projekt: Phase I



- 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.

## Typisches E-Business-Projekt: Phase II



### gegeben

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

### Aufgabe

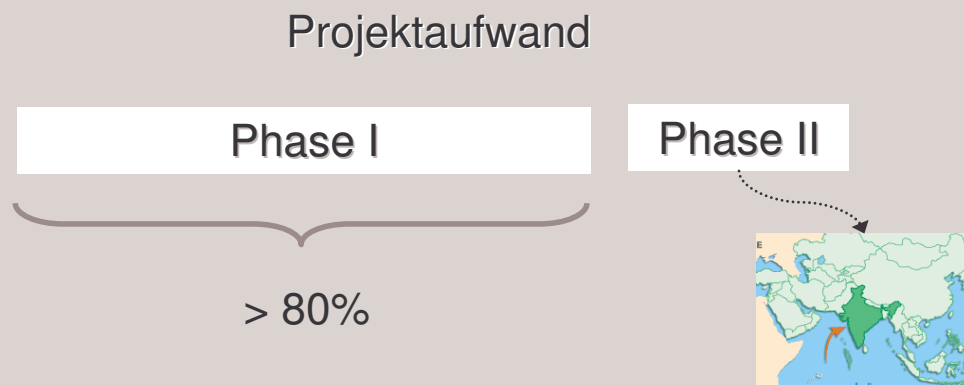
- Realisierung der Schnittstelle zwischen betriebsinterner Software und XML-Standard.

Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.

# Warum sich noch 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- und DOM-Parser
- ☑ Vor- und Nachteile von SAX und DOM
- ☑ Schema-Übersetzer

## heutige Übung

- 2. Übung (XML-Schema), 2. Gruppe

## Vorlesung nächste Woche

- Transformation von XML-Dokumenten mit XSLT