Netzprogrammierung
9. 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

Überblick

1. XML Verarbeitung

Verarbeitung von XML Dokumenten
[Folien: Klaus Schild, NBI]

Verarbeitung von XML-Dokumenten

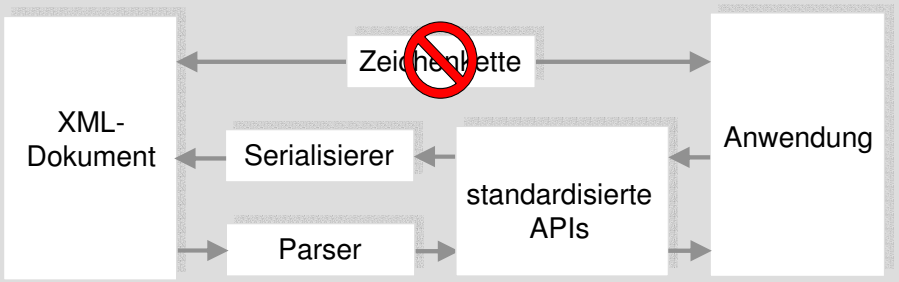
© Klaus Schild, 2003

Lernziele

- Was unterscheidet Pull- von Push-Parser?
- Was unterscheidet Einschritt- von Mehrschritt-Parser?
- Wie ordnen sich SAX und DOM bezüglich dieser Kategorien ein?
- Warum sollte immer die Anwendungslogik vom Datenzugriff getrennt werden?
- Was sind Schema-Übersetzer?

© Klaus Schild, 2003

Grundlegende Architektur



- Möglichst immer standardisierte APIs verwenden!
- Parser: Analysiert XML-Dokument und erstellt Parse-Baum mit Tags, Text-Inhalten und Attribut-Wert-Paaren als Knoten.
- Serialisierer: Generiert aus bestimmter Datenstruktur ein XML-Dokument.

© Klaus Schild, 2003

Kategorien von Parser

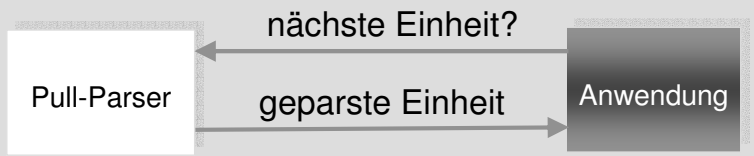
- Pull- vs. Push-Parser:

  Wer hat die Kontrolle über das Parsen, die Anwendung oder der Parser selbst?
- Einschritt- vs. Mehrschritt-Parser (engl. *one-step vs. multi-step parsing*):

  Wird das XML-Dokument in einem Schritt vollständig geparst oder werden die einzelnen syntaktischen Einheiten Schritt für Schritt analysiert?
- Beachte: Die Kategorien sind unabhängig voneinander, sie können beliebig kombiniert werden.

© Klaus Schild, 2003

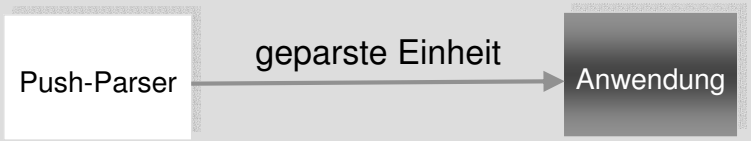
Pull-Parser



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

© Klaus Schild, 2003

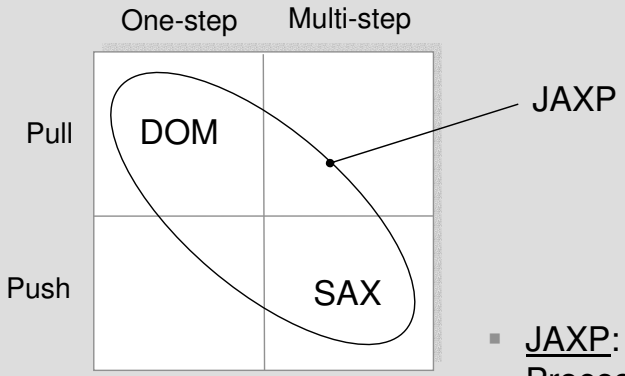
Push-Parser



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

© Klaus Schild, 2003

XML-Parser



- JAXP: Java API for XML Processing

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

© Klaus Schild, 2003

SAX-Parser

© Klaus Schild, 2003

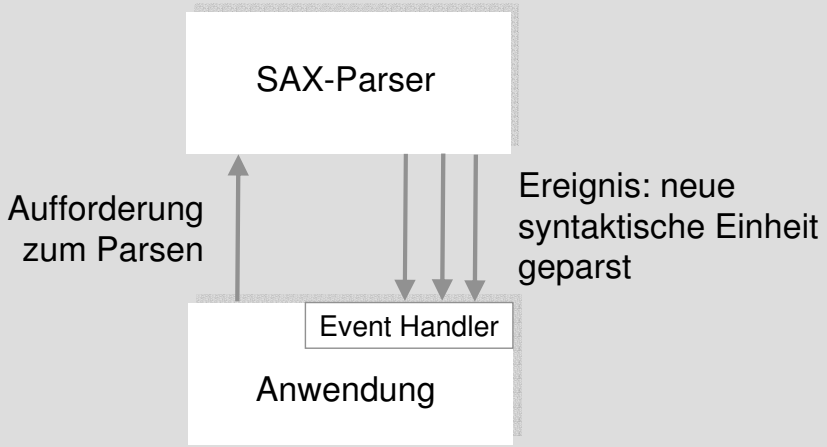
Simple API for XML (SAX)

- standardisiertes API für Mehrschritt-Push-Parsen von XML-Dokumenten
- ursprünglich nur Java-API, inzwischen werden aber auch viele andere Sprachen unterstützt
- kein W3C-Standard, sondern *de facto* Standard

➔ http://www.saxproject.org/

© Klaus Schild, 2003

Ereignisbasiertes Parsen mit SAX



© Klaus Schild, 2003

Beispiel für ereignisbasiertes Parsen

```
<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.

- Sobald eine syntaktische Einheit geparst wurde, wird die Anwendung benachrichtigt (Ereignisfluss).
- Beachte: Es wird *kein* Parse-Baum aufgebaut!

© Klaus Schild, 2003

Callback-Methoden

- startElement, endElement und characters werden Call-Back-Methoden genannt, weil sie vom Parser aufgerufen werden.
- Ein SAX-Parser hat für jede syntaktische Struktur in XML eine eigene Callback-Methode.
- Die Standard-Implementierung der Call-Back-Methoden (DefaultHandler) tut jeweils nichts.
- Die Standard-Implementierung einer Callback-Methode kann aber entsprechend den Erfordernissen überschrieben werden.

© Klaus Schild, 2003

Beispiel

```
<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 SAX-Parser
  2. passende Callback-Methoden

© Klaus Schild, 2003

Ein SAX-Parser

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse("priceList.xml", handler);
```

- SAXParserFactory.newInstance() liefert eine SAXParserFactory.
- Die Methode newSAXParser() liefert einen SAXParser mit einer Methode parse().
- priceList.xml: die zu parsende Datei (kann auch eine URL oder ein Stream sein)
- handler: Instanz von DefaultHandler, implementiert die Callback-Funktionen

© Klaus Schild, 2003

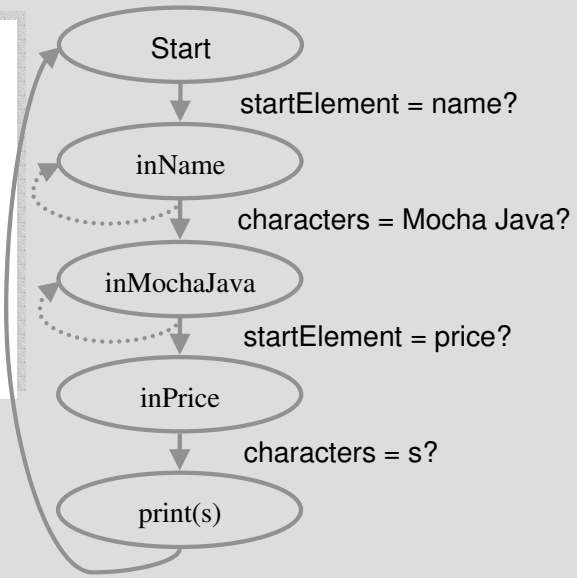
Exkurs: Factory Method

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

© Klaus Schild, 2003

Die Callback-Methoden

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



Aufgabe: Gib des Preis von Mocha Java aus!

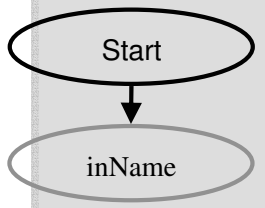
© Klaus Schild, 2003

Die Callback-Methoden

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){    inName = true;   }
else if (elementName.equals("price") && inMochaJava ){
   inPrice = true;
   inName = 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 (inPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMochaJava = false;
   inPrice = false;    }   }
```



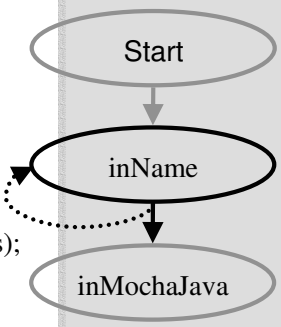
© Klaus Schild, 2003

Die Callback-Methoden

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") && inMochaJava ){
   inPrice = true;
   inName = 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 (inPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMochaJava = false;
   inPrice = false;     }   }
```



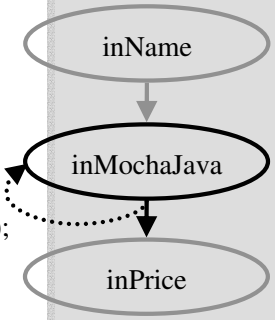
© Klaus Schild, 2003

Die Callback-Methoden

```
public void startElement(..., String elementName, ...){
if (elementName.equals("name")){     inName = true;   }
else if (elementName.equals("price") && inMochaJava ){
   inPrice = true;
   inName = 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 (inPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMochaJava = false;
   inPrice = false;     }   }
```



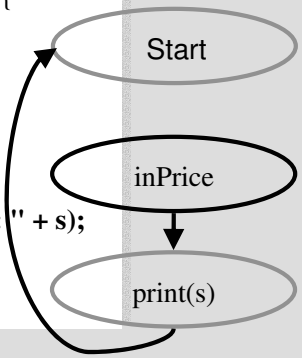
© Klaus Schild, 2003

Die Callback-Methoden

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

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

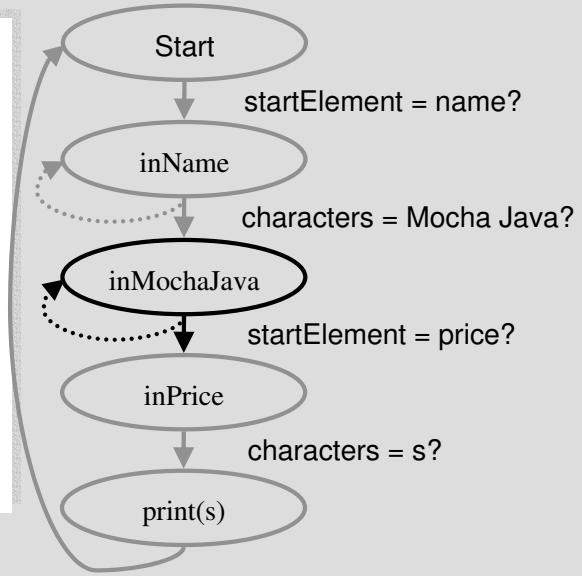
```
public void characters(char [] buf, int offset, int len) {
String s = new String(buf, offset, len);
if (inName && s.equals("Mocha Java")) {
   inMochaJava = true;
   inName = false;   }
else if (inPrice) {
   System.out.println("The price of Mocha Java is: " + s);
   inMochaJava = false;
   inPrice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



© Klaus Schild, 2003

Fehlerbehandlung

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



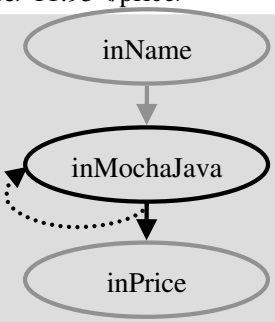
© Klaus Schild, 2003

Fehlerbehandlung

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

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

- inMochaJava erwartet price-Element
- Kommt ein name-Element, wird der Zustand einfach nicht verändert.
- Kommt danach ein price-Element, wird der aktuelle Zustand inPrice.
- Dadurch wird der Preis von MS Java ausgegeben!



© Klaus Schild, 2003

Fehlerbehandlung

- Ein SAX-Parser überprüft immer die Wohlgeformtheit eines XML-Dokumentes.
- Hat der SAX-Parser eine DTD oder ein Schema, kann er auch noch die *Zulässigkeit* des XML-Dokumentes überprüfen.
- Syntax- und Strukturfehler fängt bereits der SAX-Parser ab.
- Callback-Methoden können von einem wohlgeformten und zulässigen Dokument ausgehen.

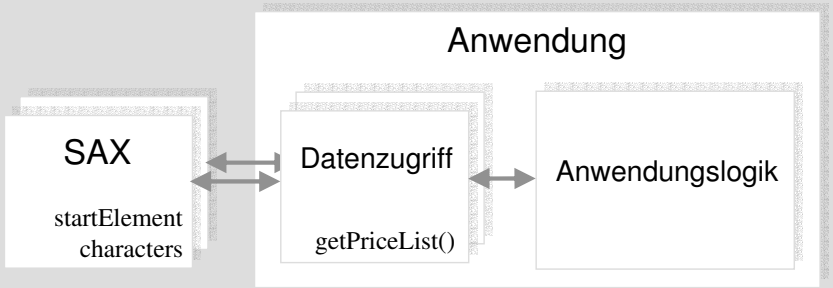
© Klaus Schild, 2003

Vor- und Nachteile von SAX

+ sehr effizient, auch bei großen XML-Dateien
– Kontext (Parse-Baum) muss von der Anwendung selbst verwaltet werden.
– Abstrahiert nicht von der spezifischen XML-Syntax
– nur Parsen von XML-Dokumenten möglich, keine Modifikation oder Erstellung von Dokumenten

© Klaus Schild, 2003

Zusätzliche Schicht zum Datenzugriff



- Immer Anwendungslogik durch spezielle Schicht von dem 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 der XML-Syntax abstrahieren.

© Klaus Schild, 2003

DOM-Parser

© Klaus Schild, 2003

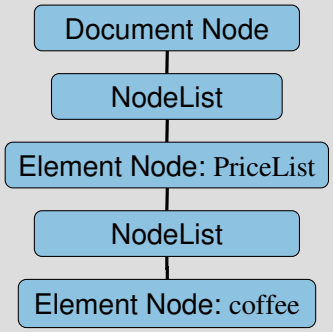
Document Object Model (DOM)

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

© Klaus Schild, 2003

Parse-Bäume

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



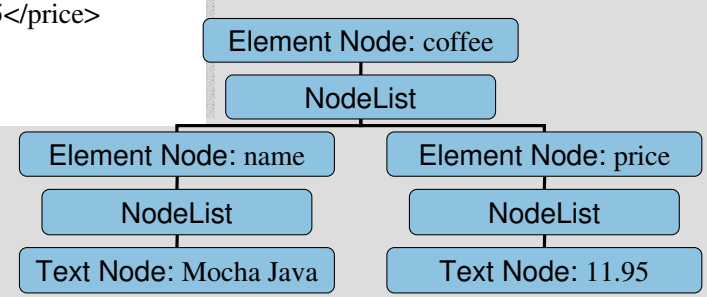
- Beachte: In DOM ist *nicht* priceList die Dokument-Wurzel.
- DOM führt virtuelle Dokument-Wurzel ein, um auch syntaktische Einheiten außerhalb von priceList (wie version="1.0") repräsentieren zu können.

© Klaus Schild, 2003

Parse-Bäume

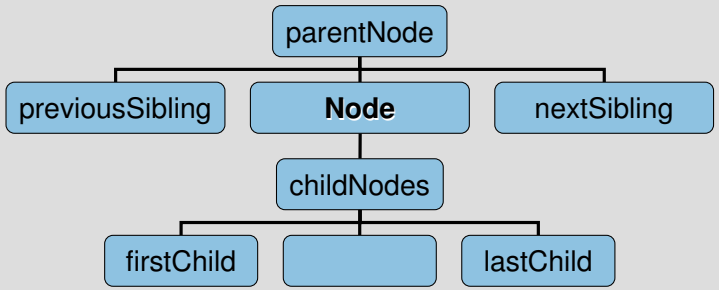
```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

- Beachte: Text-Inhalte werden als eigene Knoten dargestellt.



© Klaus Schild, 2003

Navigationsmodell



© Klaus Schild, 2003

Beispiel

```
<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

© Klaus Schild, 2003

Ein DOM-Parser

```
DocumentBuilderFactory factory =  DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse("priceList.xml");
```

- DocumentBuilderFactory.newInstance() liefert eine DocumentBuilderFactory.
- Die Methode newDocumentBuilder() von DocumentBuilderFactory liefert einen DOM-Parser.
- Der DOM-Parser hat eine Methode parse().

© Klaus Schild, 2003

Zugriffsmethoden

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

- Direkter Zugriff auf bestimmte Elemente mit getElementsByTagName möglich.
- Resultat ist eine NodeList.

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

© Klaus Schild, 2003

Zugriffsmethoden

```
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)

© Klaus Schild, 2003

Zugriffsmethoden

```
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>
```

© Klaus Schild, 2003

DOM-Methoden

```
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; } }
```

- Beachte: getFirstChild() liefert den Text-Knoten, nicht dessen Inhalt „Mocha Java“!

firstChild

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

© Klaus Schild, 2003

DOM-Methoden

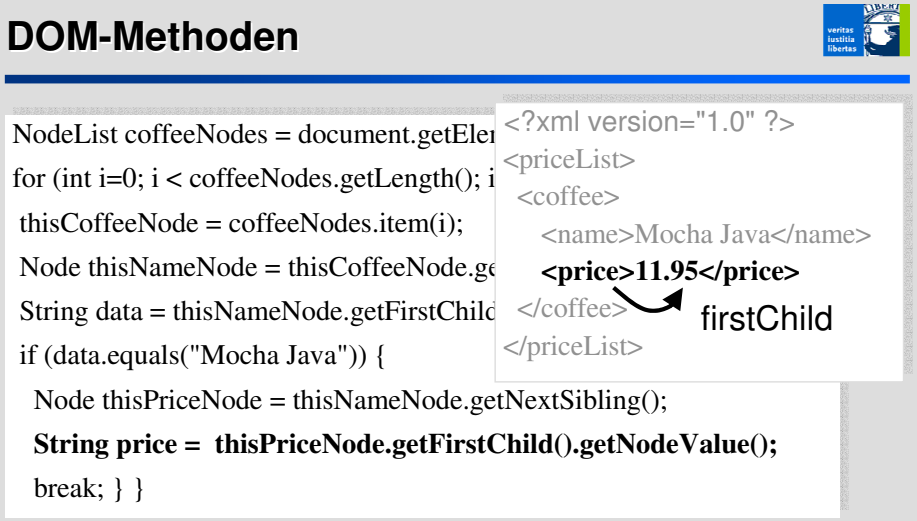
```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.getFirstChild();
 String data = thisNameNode.getFirstChild().getNodeValue();
 if (data.equals("Mocha Java"))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

© Klaus Schild, 2003

DOM-Methoden



```
NodeList coffeeNodes = document.getEle
for (int i=0; i < coffeeNodes.getLength(); i
 thisCoffeeNode = coffeeNodes.item(i);
 Node thisNameNode = thisCoffeeNode.ge
 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>
```


© Klaus Schild, 2003

Vor- und Nachteile von DOM

+ Kontext (Parse-Baum) muss *nicht* von der Anwendung verwaltet werden.
+ Auch direkter Zugriff auf Elemente und Attribute über ihre Namen (unabhängig von deren Position) möglich.
+ Nicht nur Parsen von XML-Dokumenten möglich, sondern auch Modifikation und Erstellung von Dokumenten.
– speicherintensiv, insbesondere bei großen XML-Dateien

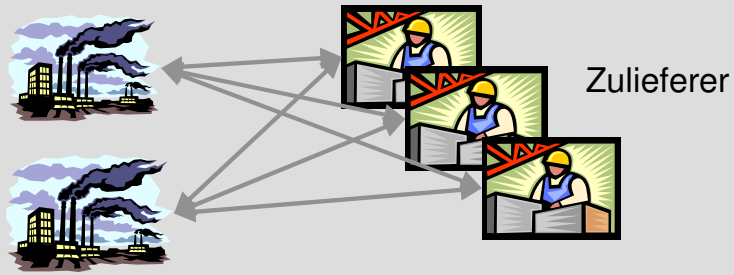
© Klaus Schild, 2003

SAX oder DOM?

- SAX ist geeignet, um gezielt bestimmte Teile von XML-Dokumenten herauszufiltern, ohne dass zu einem späteren Zeitpunkt andere Teile des Dokumentes benötigt werden.
- DOM ist geeignet, um auf unterschiedliche Teile eines XML-Dokumentes zu verschiedenen Zeitpunkten zuzugreifen.
- DOM ist zum Erstellen und Modifizieren von Dokumenten geeignet, SAX ist hierfür ungeeignet.

© Klaus Schild, 2003

Typisches E-Business-Projekt



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

© Klaus Schild, 2003

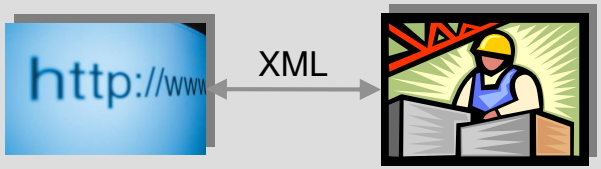
Typisches E-Business-Projekt: Phase I

- Welche Geschäftsdaten sollen ausgetauscht werden?
- Gibt es bereits einen passenden Branchenstandard (nicht unbedingt XML-basiert)?
- Wie sollen diese Geschäftsdaten in XML repräsentiert werden?
- Gibt es bereits einen geeigneten XML-Standard?
- Es muss ein gemeinsames Verständnis der verwendeten XML-Syntax hergestellt werden.

Software-Architekten entwickeln gemeinsam einen XML-basierten Branchenstandard (= XML-Schema).

© Klaus Schild, 2003

Typisches E-Business-Projekt: Phase II



- Branchenstandard liegt in Form eines XML-Schemas vor.
- Es gibt ein gemeinsames Verständnis der verwendeten XML-Syntax.
- Aufgabe: Realisierung der Schnittstelle zwischen betriebsinterner Software und dem XML-Standard.

Programmierer können Schema-Übersetzer einsetzen und von der XML-Syntax abstrahieren.

© Klaus Schild, 2003

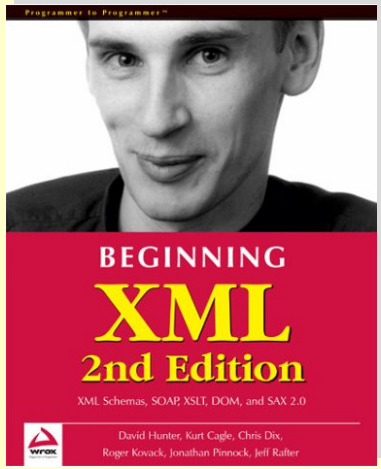
Warum sich noch mit XML beschäftigen?

- 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.

© Klaus Schild, 2003

Literatur

- Hunter, David; Cagle, Kurt; Dix, Chris: Beginning XML XML Schemas, SOAP, XSLT, DOM, and SAX 2.0 2nd ed. 2001. Wrox Press 1-86100-559-8



- Empfehlenswertes Skript einer anderen XML-Vorlesung: http://www.jeckle.de/vorlesung/xml/
- mehrere Interaktive XML-Kurse http://www.zvon.org