



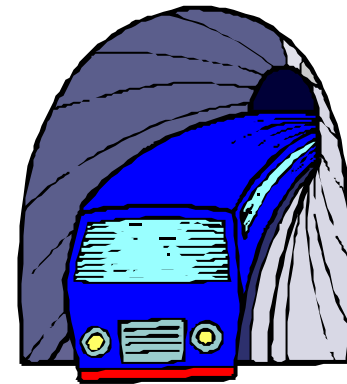
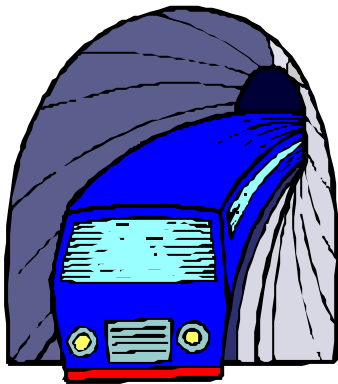
# Anbindung von XML an OOP

---

Java und C++

Vortrag am 20.06.2002

Idris Hamid Arrahmane



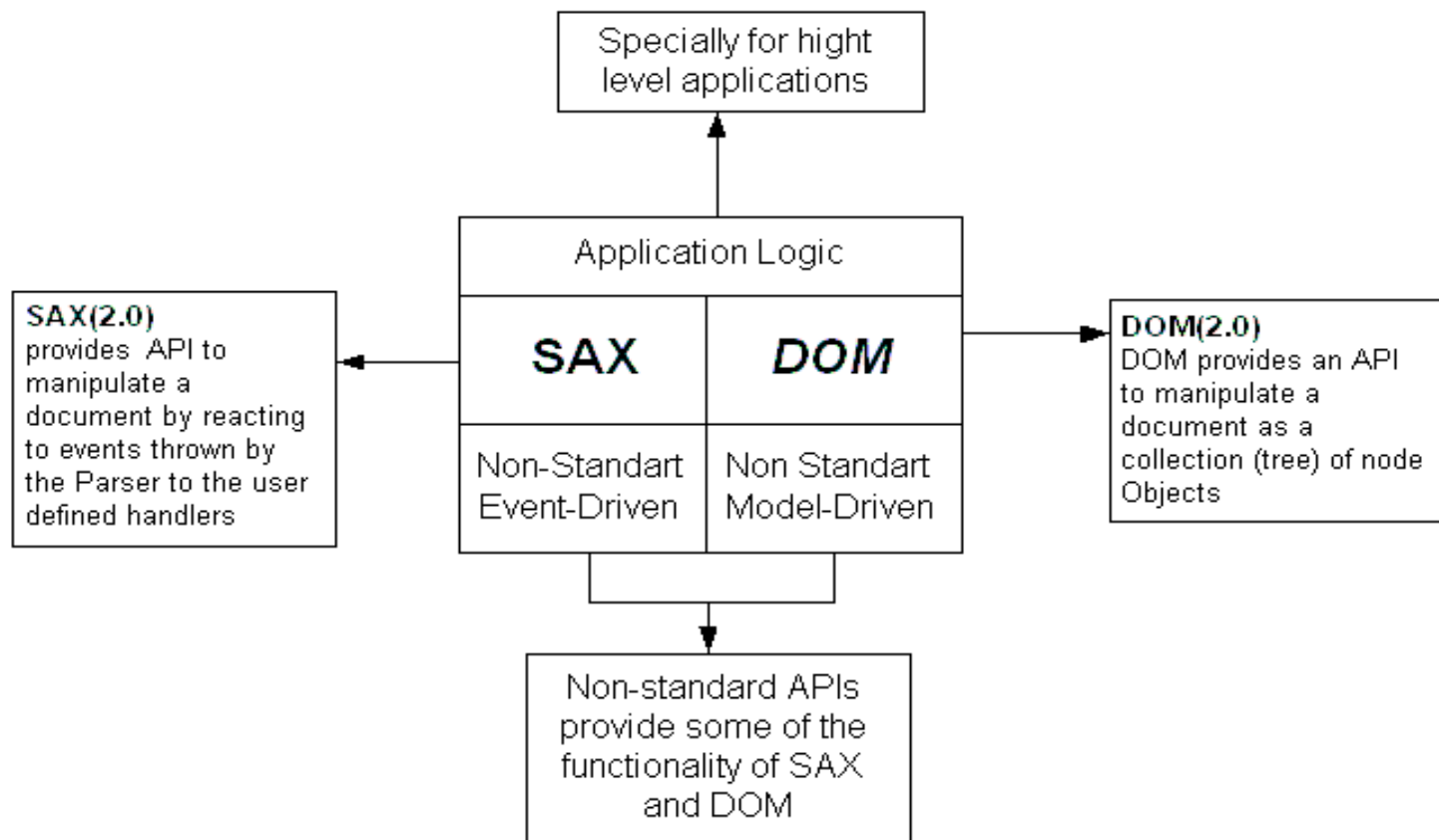


# Inhalt

---

- SAX und DOM in C++ und JAVA
- Vergleich zwischen SAX & DOM
- Der Java Class Generator
- Der C++ Class Generator

# Basis-APIs für XML



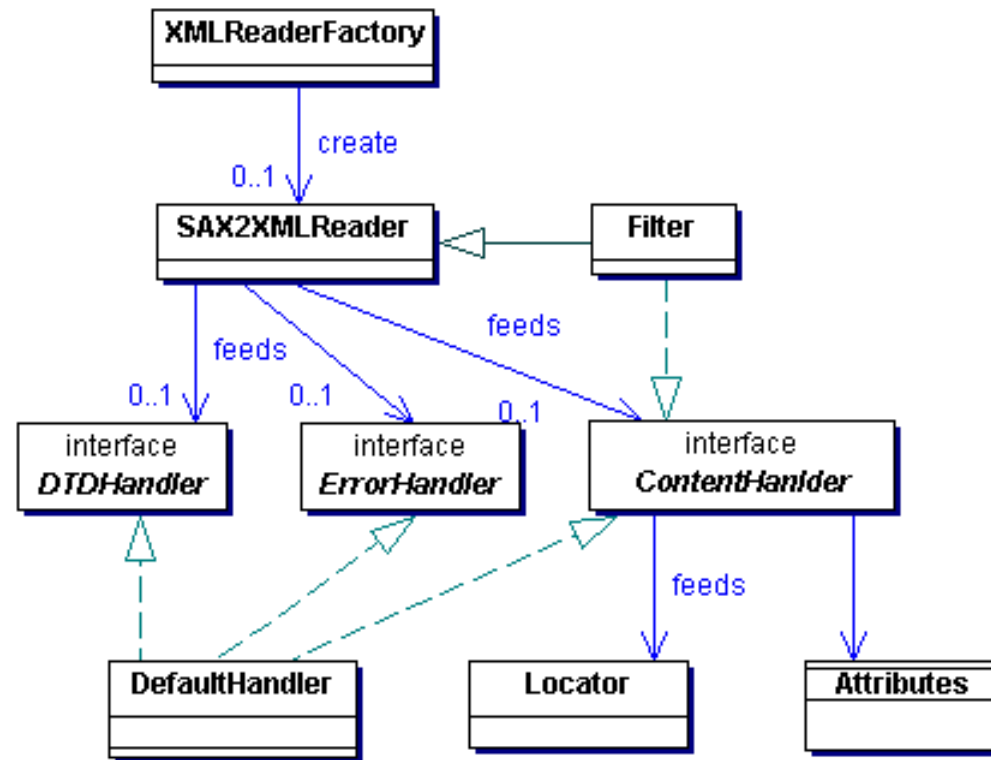


## SAX C++

---

- SAX C++ Interfaces können in folgenden Gruppen klassifiziert werden:
  - Klassen, die vom Parser implementiert sind:  
( Parser, Attributlist, Locator )
  - Klassen, die von der Anwendung implementiert werden: (ContentHanlder, ErrorHandler, Entity Resolver)
  - Standard SAX-Klassen: InputSource, SAXException, SAXParseException und HandlerBase

# SAX2-MAP





# ContentHandler

---

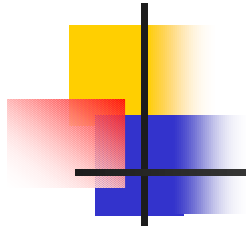
- `void startDocument()`  
// wird am Anfang des Parsens aufgerufen
- `void startElement (Const XMLCh URI, const XMLCh const localname, const XMLCh qname AttributeList& atts)`  
uri: identifiziert den Namespace  
URI  
localname: enthält den Namen des Elements ohne das Namespace Präfix  
qname: Elem-Namen mit dem Präfix

- Atts : enthält eine Kollektion von Attribut

Beispiel:

```
<movies xmlns:Kubrick="http://www.univ-trier.de/Buecher">  
<Kubrick:movie>"EYES WIDE SHUT"</kubrick>  
</movies>
```

1. URI "<http://www.uni...>"
2. Localname = "movie"
3. Qname = „kubrick:movie“
4. Atts = Emptyattributs



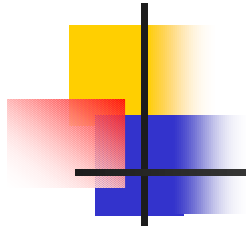
- `void endElement(XMLCh* const name)`  
// wird beim End-Tage eines Elements mit Übergabe des Element-Name aufgerufen
- `void endDocument()`  
// wird am Ende des Parsens aufgerufen

- `virtual void startPrefixMapping(const XMLCh* prefix, const XMLCh const URI)`  
// wird aufgerufen, wenn ein neues Präfix mapping gefunden wird

Präfix mapping hat die Form: (Spezielles Attr)  
Xmlns:[optional name]

```
<student Xmlns:Thomas="http://cip.uni-trier.de/Thomas">....</student>
```

- `void endPrefixMapping(const XMLCh const prefix)`



- `void characters (const XMLCh* chars, const unsigned int length)`  
// ermöglicht den  
eigentlichen Inhalt eines  
Elements auszulesen.
- `void ignorableWhitespace(const XMLCh const chars, const unsigned int length)`  
// Wird für jeden Durchlauf mit  
übergehbaren Leerzeichen  
aufgerufen
- `void processingInstruction(const XMLCh * target, const XMLCh* const data)`  
// wird für jede  
Verarbeitungsanweisung  
aufgerufen
- `void setDocumentLocator(const Locator const locator)`  
// dient zur genauen  
PositionsBestimmung des  
Parsens innerhalb des  
Dokuments



# DTD Handler

---

- Enthält die folgenden zwei Methoden:
  1. void notationDecl(const XmlCh name, const XMLCh publicId, const XMLCh SystemId)

//wird aufgerufen, wenn der Parser auf eine Deklaration einer Notation in der DTD stößt

Beispiel:

```
<! Notation:ISO3166 PUBLIC " ISO/IEC 3166:1993//  
  NOTATION Codes for the Representation of  
  the Languages//
```

```
  EN" " ie3166">
```

- name = "ISO3166"
- publicId = "ISO/IEC .....//EN"
- systemId = "ie3166"

2. void unparsedEntityDecl(const XMLCh name,const XMLCh publicId, const XMLCh SystemId, const XMLCh const notationName)

Beispiel:

```
<!NOTATION gif PUBLIC "Graphics Interchange  
  Format//EN">
```

```
<! ENTITY SAX2MAP SYSTEM "/SAX2MAP.gif"  
  NDATA gif >
```

Dann ruft der Parser diese Methode mit den folgenden Parametern:

- name = " SAX2MAP"
- publicId = null
- systemID = "/SAX2MAP"
- notationName = "gif"



## ERROR Handler

---

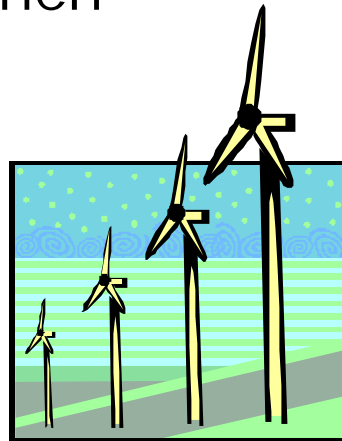
- Drei Methoden zur Behandlung von Fehlern: (errors, fatal errors und warnings):
  - 1- `void warning(const SAXParseException& exception)`  
// wird aufgerufen beim auftreten eines Problems, das nicht als Fehler angesehen wird.
  - 2- `virtual void error(const SAXParseException& exception)`  
// im Fall, dass ein nicht Fataler Fehler auftritt
  - 3- `void fatalError(const SAXParseException& exception)`  
// Dokument ist ungültig



# DefaultHandler

---

- Defaulte Implementierung für alle Handler Interfaces
- Gut geeignet für einfache Programme, die nur wenig SAX-Methoden brauchen





## Sources: SAX2XMLReader

---

- Quelle von Parsing-events
- SAX2XMLReader ↔ XMLReader in Java
- Enthält Methoden, die vom Parser implementiert werden müssen



# SAX2XMLReader-Methoden

---

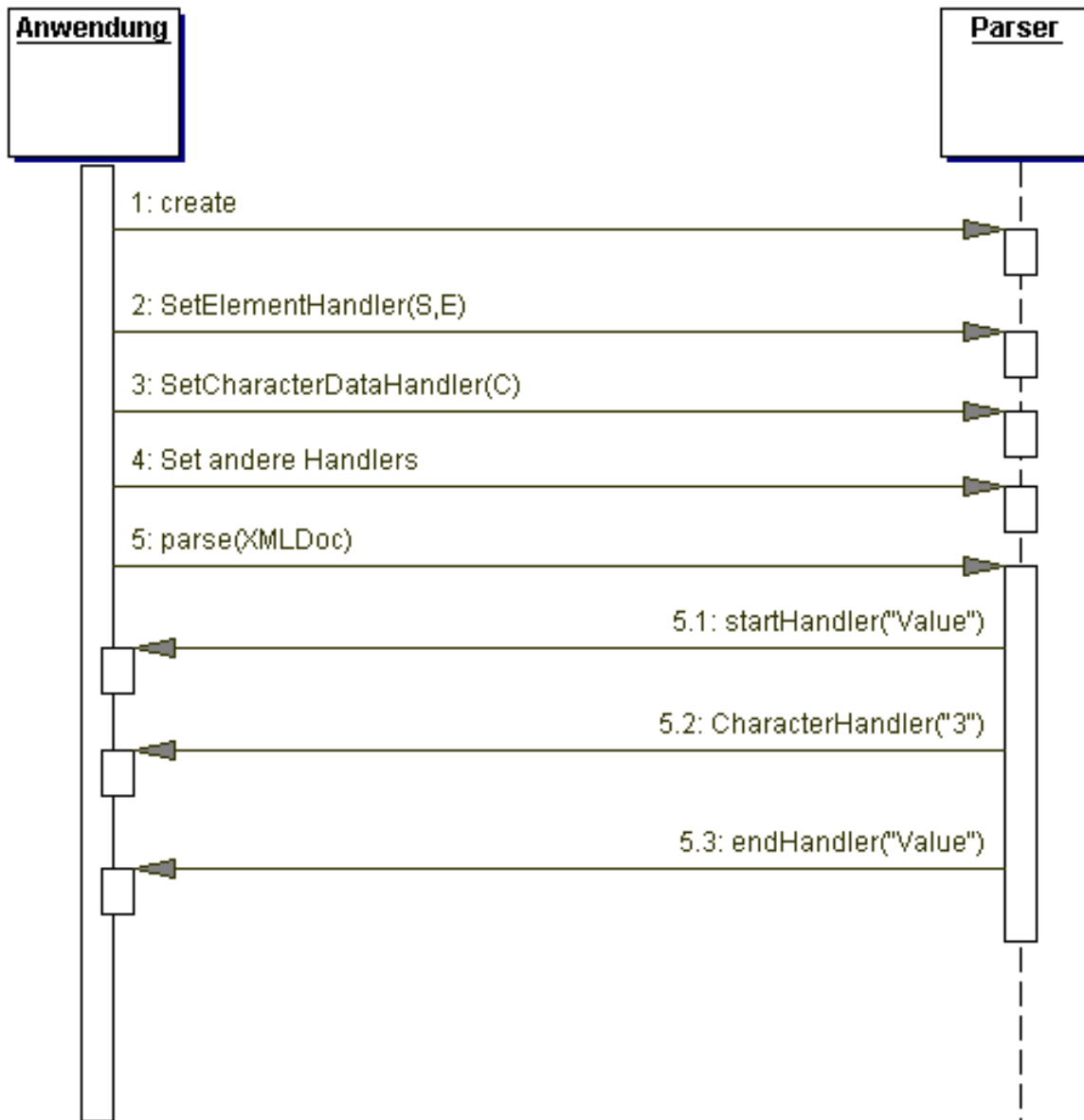
- Handler-Registrierung Methoden:
  1. ContentHandler setContentHandler()
  2. DTDHandler setDTDHandler()
  3. ErrorHandler setErrorHandler()
- Parsen eines Dokuments
  1. void parse(const XMLCh const systemId)



# Vorgehen einer Ereignis-basierten Anwendung

---

1. Erzeugen einer neuen Parser-Instanz
2. Registrierung des Handlers im Parser
3. Beginn des Parsens
4. Sequentielles Durchgehen des Dokuments plus Aufruf des Handlers für jeden vorkommenden Event
5. Bearbeitung der Daten



Die Datei enthält  
das folgende  
XML-Dokument:  
<Value>3</Value>



# SAXBeispiel (extrahiert Informationen aus einem XML-DO)

```
import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXHandler extends HandlerBase {

    public static void main(String[] argv) {
        try {
            // get an instance of the parser
            Parser parser = new SAXParser();

            // create a SAX event Handler and register it with the
            parser
            SAXHandler handler = new SAXHandler();
            parser.setDocumentHandler(handler);

            // convert file to InputSource and parse
            InputSource xmldoc = new InputSource (new
            FileInputStream(argv[0]));
            parser.parse(xmldoc);
        }

        catch (Exception e)
        {
            System.out.print(e.toString());
        }
    }

    /*******Simple Implementation of DocumentHandler
    interface******/
    public void startElement(String name,AttributeList atts)
    throws SAXException
    {
        System.out.println("StartElement:" + name);
        for(int i=0;i<atts.getLength();i++)
        { String aname = atts.getName(i);
          String type = atts.getType(i);
          String value = atts.getValue(i);

            System.out.println("
            "+aname+"("+type+"")+"="+value);
        }
    }

    public void characters(char[] buf,int start,int len)
    {
        System.out.print("Characters:");
        System.out.println(new String(buf,start,len));
    }
}
```



# Das Eingabedokument

---

```
<BOOKCATALOG>  
  <BOOK ISBN "7654">  
    <TITLE>La memoire d´un roi</TITLE>  
    <AUTHORNAME>MOI</AUTHORNAME>  
    <PRICE>1.00</PRICE>  
  <BOOK>  
</BOOKCATALOG>
```



# Ausgabe

---

- StartElement: BOOKCATALOG
- StartElement: BOOK
- ISBN(CDATA) = 7654
- StartElement: TITLE
- Characters: La memoire d´un roi
- StartElement: AUTHORNAME
- Characters: MOI
- StartElement: PRICE
- Characters: 1.00



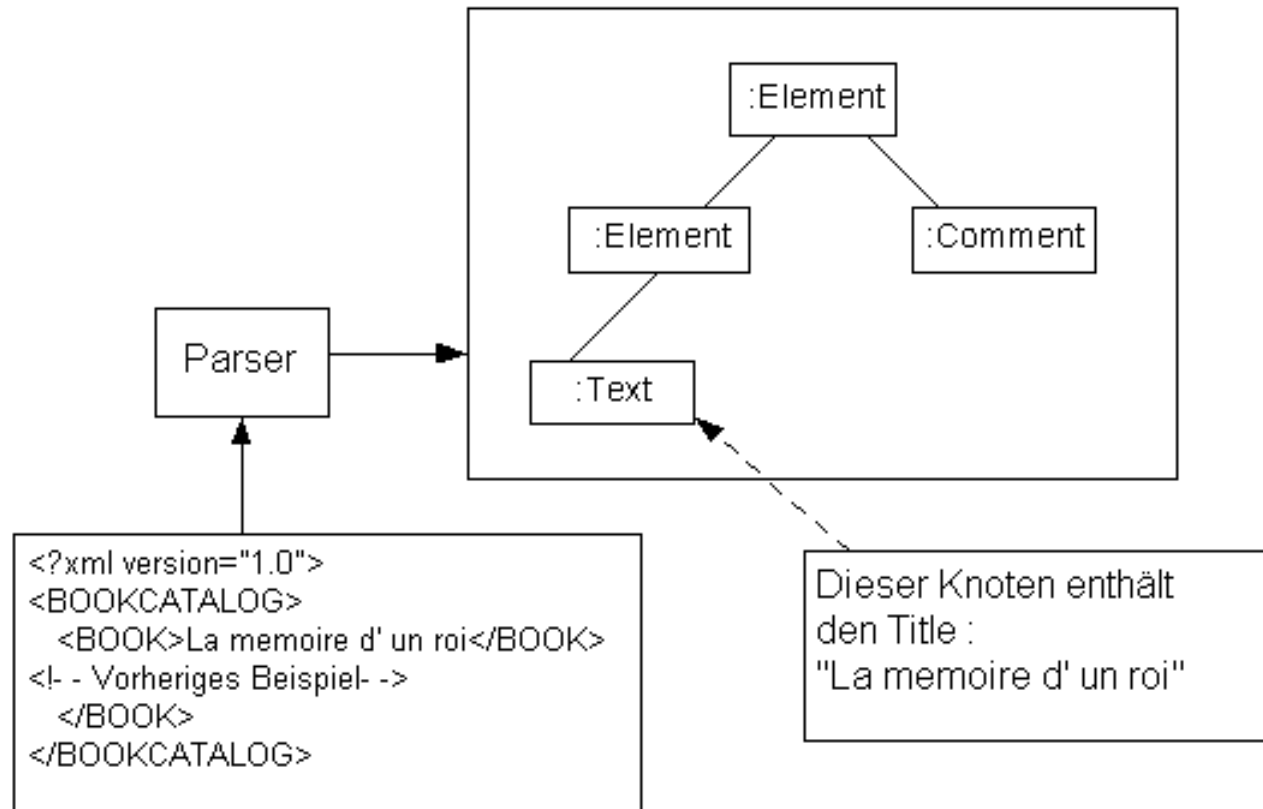


# DOM

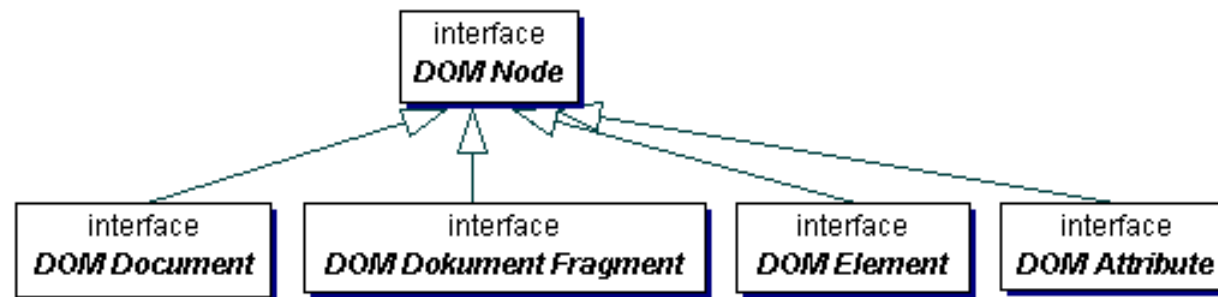
---

- DOM steht für Document Object Model
- Wurde von W3C 1998 als Standard recommendation verabschiedet
- DOM ist eine Objektorientierte API
- Diese API erzeugt aus einem XML-Dokument einen Baum und ermöglicht Zugriff darauf

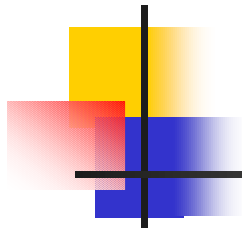
# Darstellung des Dokuments vor und nach dem Parsen



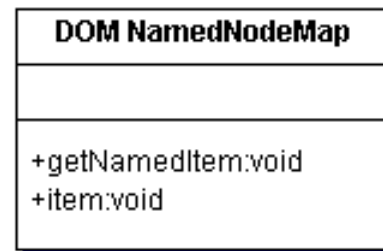
# DOM-Basisinterfaces



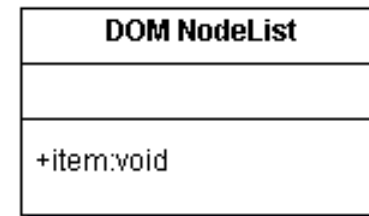
Dieses UML\_Diagram entspricht Apache C++ Implementierung. Bei Microsoft-Toolkit ist das selbe nur mit dem Prefix IXML statt DOM\_ (zb:IKMLDocument



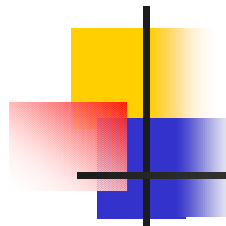
# Kollektionen



NamedNodeMap provides an unordered list of nodes, accessible by name. Note that you can also access them by index, but only to facilitate iteration, not because it is ordered

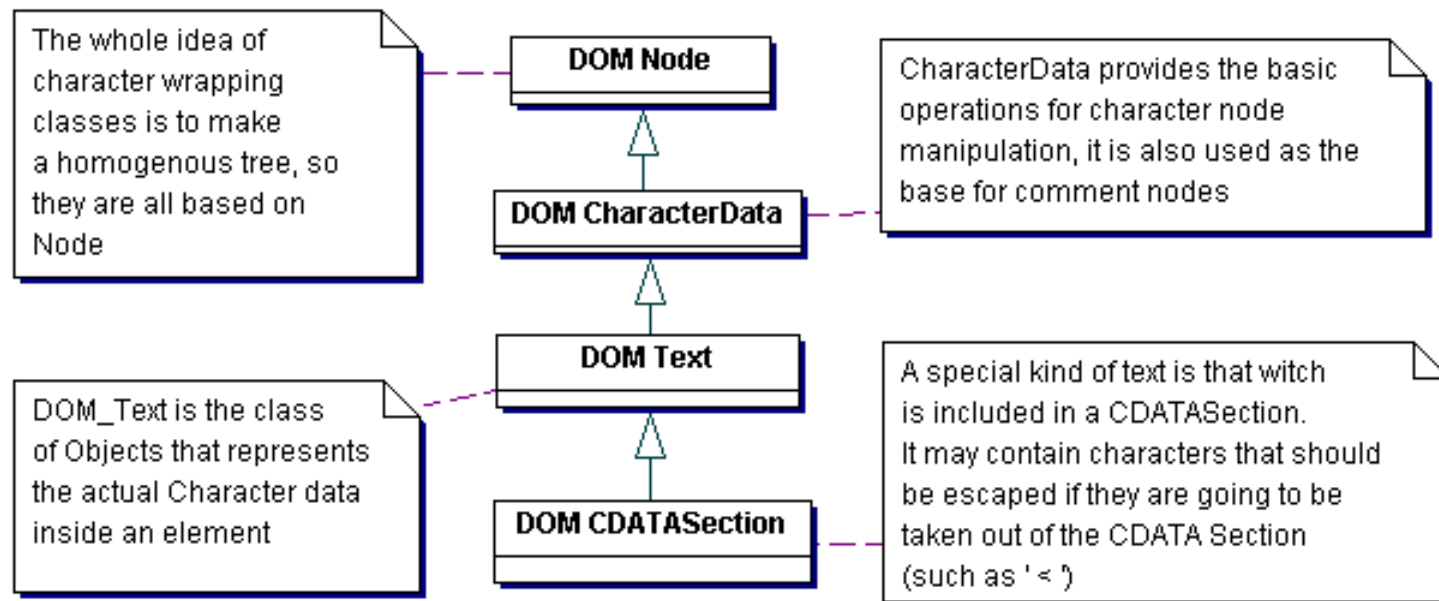


Node List provides an ordered list of nodes, accessible by index



# Text

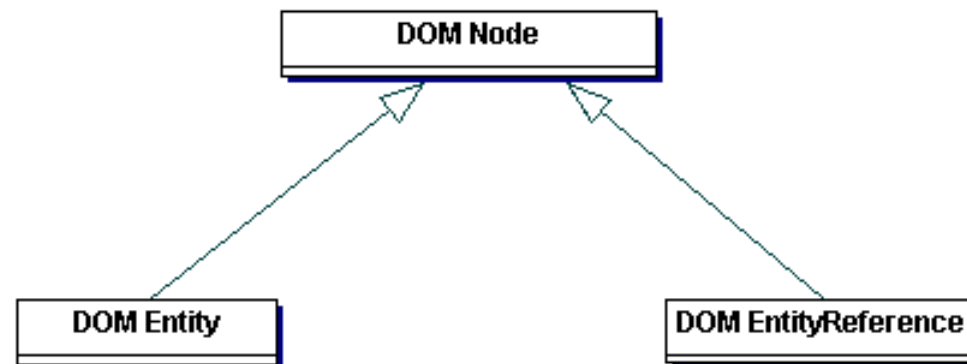
---





# Entities

---



Die Methoden in diesen Schnittstellen dienen zur Bearbeitung von Entities und Ihren Referenzen



# DOM mit JAVA

---

- In DOM gibt' s Methoden zur Manipulation von:
  - Knoten && Knoten-Objekten
  - Elementen
  - Attributen
  - CharacterData





# Methoden eines Knotens-Objekts

---

- `Short getNodeType();` // gibt den Typ als Zahl zurück
- `String getNodeName();` // Namen des Knotens zurück
- `String getNodeValue();` // den Wert zurück
- `void setNodeValue(String nodeValue)` //Wert ändern
- `boolean hasChildNodes();`
- `NamedNodeMap getAttributes();` // Liste aller  
Attributen zurück.



# Manipulation von Knoten

---

- Node `removeChild(Node OldChild)` throws `DOMException`;  
// entfernt einen Knoten
- Node `InsertBefore(Node newChild, Node OldChild)`  
// fügt einen oberhalb eines anderen ein
- Node `appendChild(Node newChild)`  
// hängt einen neuen an einen anderen an
- Node `replaceChild(Node newChild, Node OldChild)`  
// ersetzt einen durch einen anderen
- Node `cloneNode(boolean deep)`:  
//kopiert einen Knoten



# Elemente

---

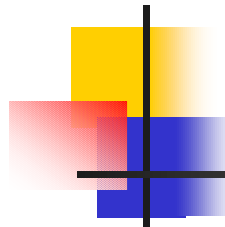
- `String getTagName();`  
    // gibt den Namen eines Elements zurück
  
- `void normalize();`  
    // fasst verschiedene Knoten im Baum  
    zusammen



# Attribute

---

- `String getAttribute(String name);`
- `void SetAttribute(String name,String value);`  
`// erzeugt ein neues Attribut`
- `void removeAttribute(String name);`
- `Attr getAttributeNode(String name);`  
`// gibt ein Attribut eines Elements zurück`
- `void setAttributeNode(Attr newAttr)`  
`// hängt an ein Element ein neues Attribut an`
- `void setValue(String value)`  
`// überschreibt den Wert eines Attributes`



# Text

---

- `String getData(); // Text auslesen`
- `void setData(); // Text ersetzen`
- `Int getlength(); // Länge des Textes einlesen`
- `Void appendData(String arg);`  
`// Ein neues Text an das Ende eines anderen anhängen`



# Navigation

---

- `Node getParentNode();` // einen Knoten weiter nach Oben
- `NodeList getChildNodes();` // alle KinderKnoten als Liste
- `Node getFirstChild();` // das linke Kind
- `Node getLastChild();` // das rechte Kind



# Methoden der Klasse Document

---

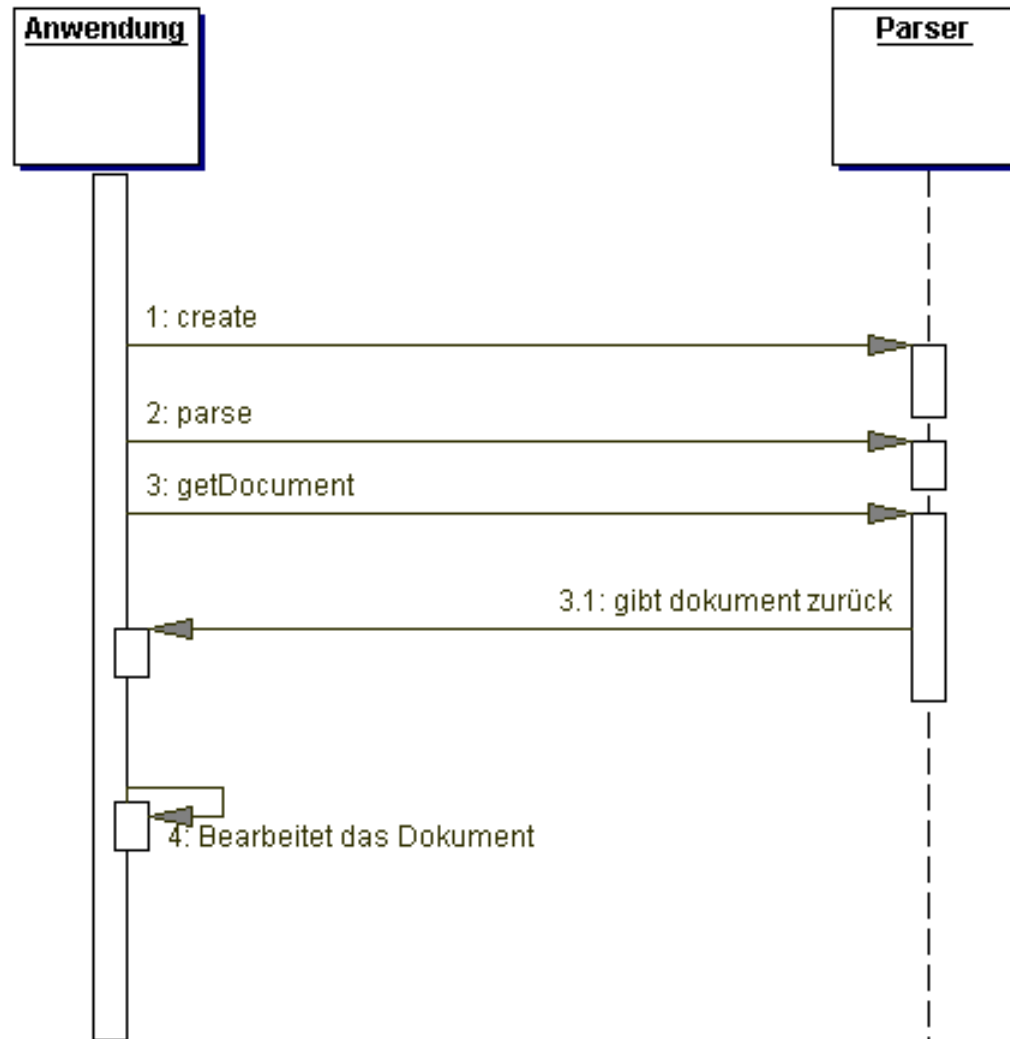
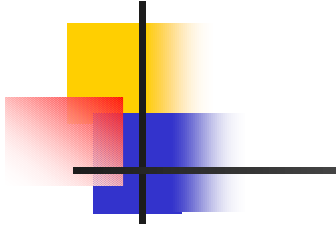
- `DocumentType getdocType();` // Type des Dokuments
- `DomImplementation getImplementation();`  
//informiert über die Fähigkeit der Implementierung
- `Element getDocumentElement();` // Verweis auf die Wurzel
- `NodeList getElementbyTagName();`  
// erlaubt die Suche nach einem bestimmten Knoten



# Vorgehen

---

- Erzeuge ein FileInputStream von der Eingabe
- Erzeuge eine DOMParser Instanz
- Parsen des Dokuments
- Parser beauftragen ein Handle an die Root des DokumentenObjektmodells zurückzuliefern
- Zugreifen auf jeden Teil des Dokuments mit dem Handle





# DOMBeispiel (Einige Informationen ausgeben)

```
public class DOMExample{
public static void main(String[] argv)
{
    FileInputStream xmldoc = new
    FileInputStream(argv[0])
    Domparger = new DOMParser();

    parser.parse(xmldoc);

    Document doc parser.getDocument();

    printElementAttributes(doc);

    static void printElementAttributes(Document
    doc)
    {
        NodeList n1=
        doc.getElementsByTagName("*");
        Element e;
        XMLAttr Attr;

        String attrname,attrva;
        NamedNodeMap nnm;

        for (int j=0; j<n1.getLength() + ":")
        {
            e= (Element) n1.item(j);

            System.out.println(e.getTagName()+":");

            e.getAttributes();
            if(nnm!=null)
            {
                for( int i=0;i<nnm.getLength();i++)
                {
                    Attr = (XMLAttr) nnm.item(i);

                    attrname = Attr.getName();

                    attrval = Attr.getNodeValue();
                    System.out.println(" "+ attrname+ "=" +
                    attrval);
                }
            }
            System.out.println();
        }
    }
}
```



# Ein und Ausgabe

---

```
<BOOKCATALOG>
  <BOOK ISBN "7654">
    <TITLE>La memoire d´un
      Roi
    </TITLE>

    <AUTHORNAME>MOI
  </AUTHORNAME>

    <PRICE Remarque = " Prix
  convenable">1.00
  </PRICE>
  <BOOK>
</BOOKCATALOG>
```

- <BOOKCATALOG>
- BOOK:
- ISBN = 7654
- TITLE:
- AUTHORNAME:
- PRICE:
- Remarque = Prix convenable



## Vergleich SAX & DOM

---

- SAX ist schneller als DOM
- DOM ermöglicht den freien Zugriff auf die Elemente
- SAX ist eine Leistungsärmere Schnittstelle gegen über DOM
- SAX ist Speicher-effizienter als DOM
- SAX ist gut geeignet für gute strukturierte Daten

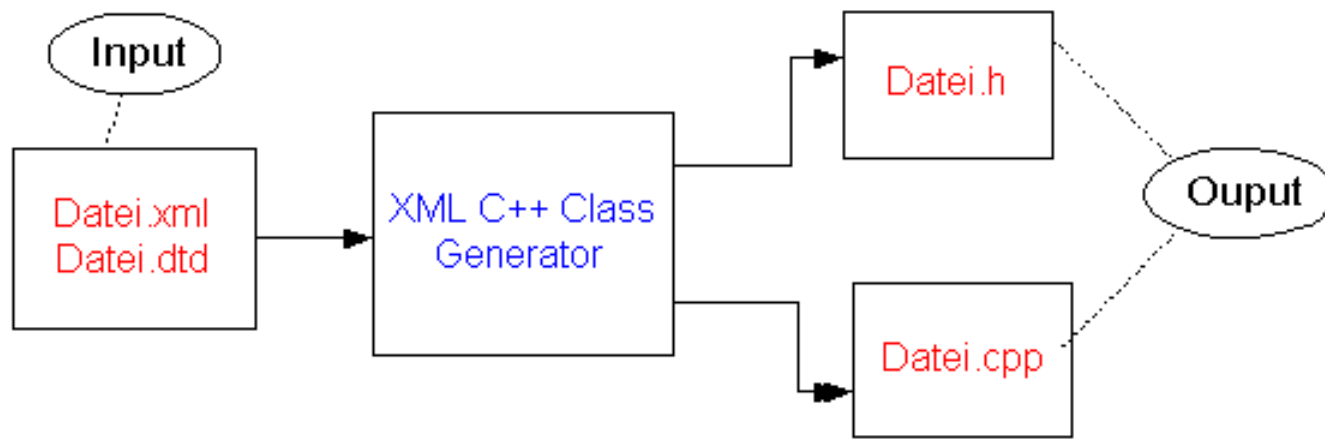
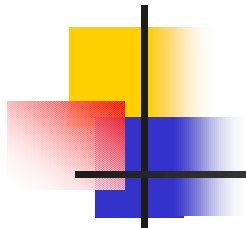


# XML C++ Class Generator

---

- Der Class Generator erzeugt C++-Klassen aus einer DTD erzeugen
- XML C++ Class Generator arbeitet mit dem Oracle XML Parser für C++
- <http://otn.oracle.com/tech/xml> (unter dieser URL verfügbar)
- Der Class Generator ist ein einzelnes Befehlszeilenprogramm (xmlcg) oder eine Bibliothek (libxmlg8.a)

```
Class XMLClassGenerator{ // Das ist die wichtigste Klasse in der
                        Bibliothek
    Public:
    uword generate(DocumentType* dtd,char* outdir = (char *) 0);
    .
    .// die Methode generate liefert einen numerischen Fehler
    }
```





# Vorgehen

---

1. Parse das Dokument oder die DTD
2. Rufe die DTD mit `XMLParser::getDOCType` auf
3. Rufe die Methode `generate` auf
4. Die generierten Klassen in einem C++ Programm verwenden



# Beispiel einer generierten Klasse

---

- `<! ELEMENT A (#PCDATA | B)* >`
- `class A: public Element`
- `{`
- `public:`
- `A(Document* doc);`
- `A(Document* doc, String data);`
- `A(Document* doc, B* kindelement);`
- `void addNode(B* Kindelement);`
- `void addData(Document* doc, String data);`
- `}`



# Der Java Class Generator

---

- Der XML Class Generator erzeugt Java-Quelldatei aus einer XML-DTD, um gültige Dokumenteninstanz aus dynamischen Daten zu erstellen.
- Der Class Generator arbeitet in Verbindung mit dem Parser, der die DTD parst und die geparste DTD an den Class Generator übergibt.



## Vorgehen

---

- Der Class Generator fragt die DTD nach allen Elementen ab.
- Eine Java Klasse wird für jedes Element erzeugt
- Die Root-Element-Klasse besitzt zusätzliche Methoden zur Auswertung und Ausgabe des Erstellten XML-Dokuments



# Die Eingabe DTD

---

- `<!-- DTD for Book Data -->`
- `<! ELEMENT BOOKCATALOG (BOOK)* >`
- `<! ELEMENT BOOK (TITLE, AUTHORNAME, PRICE?) >`
- `<! ATTLIST BOOK ISBN CDATA #REQUIRED >`
- `<! ELEMENT TITLE (#PCDATA) >`
- `<! ELEMENT AUTHORNAME (#PCDATA) >`
- `<! ELEMENT PRICE (#PCDATA) >`



# Verarbeitung der DTD

---

- Public class SampleMain
- {
- public static void main (String args[])
- {
- // open the External DTD File
- XMLParser parser = new XMLParser();
  
- String doctype\_name = null;
  
- parser .parseDTD( fileToURL (args[2]), args[1]);
  
- DTD dtd = (DTD) parser.getDoctype();
- doctype\_name = args[1];
- 
- ClassGenerator generator = new ClassGenerator(); // generate the Java files....
  
- generator.setGeneratecomments(true);     // set generate comments to true
- generator.setOutputDirectory(".");         // set OutputDiretory
- generator.setValidationMode(true);        // Set validation Mode to true
- generator.generate(dtd, doctype\_name);    // generate Java source
- .
- .
- .
- }



# Erzeugen XML-Dokument aus Java-Klassen

---

- Public class CreateBooks
- {
- try
- {
- BOOKCATALOG booklist = new BOOKCATALOG();
- // New book Mybook
- BOOK Mybook = new BOOK("7654"); // create book1 and set ISBN
  
- TITLE title1 = new TITLE("La memoire d'un roi");
  
- AUTHORNAME authorname1 = new AUTHORNAME("MOI");
  
- PRICE price = new PRICE("1.00");
- Mybook.addnode(isbn1); // Add data as tree nodes to Mybook
- Mybook.addnode(title);
- .
- .
- bookList.addNode(Mybook); // Add book as tree node to bookList doc root
- bookList.validateContent();
- bookList.print(System.out);
- catch.....
- }

# Das erzeugte XML-Dokument

- `<?xml version = "1.0"?>`
- `<!DOCTYPE BOOKCATALOG SYSTEM " bookcatalog.dtd" >`
- `<BOOKCATALOG>`
- `<BOOK ISBN "7654">`
- `<TITLE> La memoire d´un roi </TITLE>`
- `<AUTHORNAME> MOI </AUTHORNAME>`
- `<PRICE> 1.00 </PRICE>`
- `<BOOK>`
- `/BOOKCATALOG>`

