XML-Grundlagen:

XML ist casesensitive

tags

Starttag <...>
EndeTag </....>
leeres Tag <...../> entspricht <...> (nix) </...>

Attribute
<tagname attrName="wert" attrName="Wert">.....</tagname>

Wann Attribute und wann eingebettete Tags verwendet werden, ist nicht festgelegt.

- verwendete Parser: org.apache.xerces
- Literatur: Brett Mc Laughlin: Java & XML O'Reilly, Elliote Rusty Harold & W. Scott Means XML in a Nutshell, O'Reilly

Start von xml01:

```
java -classpath ./xerces-1_4_4/xerces.jar:. xml01 http://www.htw-dresden.de/~beck/test.xml
```

Bsp.:

```
<?xml version="1.0" ?>
<AddrPhones>
<Person>
<Name> Beck</Name>
<FistName> Arnold </FistName>
<Street> Fr-List-Platz 1 </Street>
<ZIP> 01069 </ZIP>
<Town> Dresden </Town>
<Phone>
    <Nr> 0351/462-2130</Nr>
    <Note> Dienst </Note>
</Phone>
<Phone>
    <Nr> 0351/462-3564</Nr>
    <Note> Fax </Note>
</Phone>
</Person>
<Person>
<Name>Huckebein</Name>
<FistName>Hans</FistName>
<Street>Wilhem-Bush-Str. 73</Street>
<ZIP>31719</ZIP>
  <Town>Wiedensahl</Town>

</Person>
</AddrPhones>
```

SAX Parser

Ereignisgesteuerter Parser, der bei Erkennen eines XML-Items eine Callbackfunktion in einem Handlerobjekt aufruft. (Ähnlich Eventhandler bei AWT)

- Der eigentliche Parser wird wie folgt erzeugt:

```
private String vendorParserClass="org.apache.xerces.parsers.SAXParser";
XMLReader myReader = XMLReaderFactory.createXMLReader(vendorParserClass);
```

- Festlegen der Datenquelle

```
InputSource S=new InputSource(I);

InputSource()                          // Zero-argument default constructor.

InputSource(InputStream byteStream)  // Create a new input source with a byte stream.

InputSource(Reader characterStream)  // Create a new input source with a character stream.

InputSource(String systemId)         // Create a new input source with a system identifier
                                     // systemId enthält ein URI URL/Dateiname
S.setSystemId(I);                    // URI der XML-Quelle angeben
```

- Contenthandler angeben

```
myReader.setContentHandler (new org.xml.sax.ContentHandler() { ... });
```

- Starten des Parsers

```
myReader.parse(S);
```

Method Summary

void	**characters**(char[] ch, int start, int length) Receive notification of character data.
void	**endDocument**() Receive notification of the end of a document.
void	**endElement**(String namespaceURI, String localName, String qName) Receive notification of the end of an element.
void	**endPrefixMapping**(String prefix) End the scope of a prefix-URI mapping.
void	**ignorableWhitespace**(char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	**processingInstruction**(String target, String data) Receive notification of a processing instruction.
void	**setDocumentLocator**(Locator locator) Receive an object for locating the origin of SAX document events.
void	**skippedEntity**(String name) Receive notification of a skipped entity.
void	**startDocument**() Receive notification of the beginning of a document.
void	**startElement**(String namespaceURI, String localName, String qName, Attributes atts) Receive notification of the beginning of an element.
void	**startPrefixMapping**(String prefix, String uri) Begin the scope of a prefix-URI Namespace mapping.

Beispiel für eine Implementierung eines ContentHandlers

```
Locator L; //Locator wird von Sax benötigt,stellt Zeilen-/Spalteninformationen u.a. bereit

myReader.setContentHandler (new org.xml.sax.ContentHandler()
{
  public void characters(char[] ch, int start, int length)
  {
    if (ch[start]!='\n'&&length>1)
      System.out.println("Chars: "+new String (ch,start,length)+"\n");
  }
  public void endDocument(){System.out.println("EndDoc\n");}
  public void endElement(String namespaceURI, String localName, String qName)
  {
    System.out.println("EndElem: "+namespaceURI+' '+localName+' '+qName+"\n");
  }
  public void endPrefixMapping(String prefix)
  {
    for (Iterator i=NameSpaces.keySet().iterator();i.hasNext();)
    {
       String Uri =(String)i.next();
       String Pref=(String)NameSpaces.get(Uri);
       if (Pref.equals(Pref)){NameSpaces.remove(Uri);break;}
    }
  }
  public void ignorableWhitespace(char[] ch, int start, int length){}
  public void processingInstruction(String target, String data)
  {
  }
```

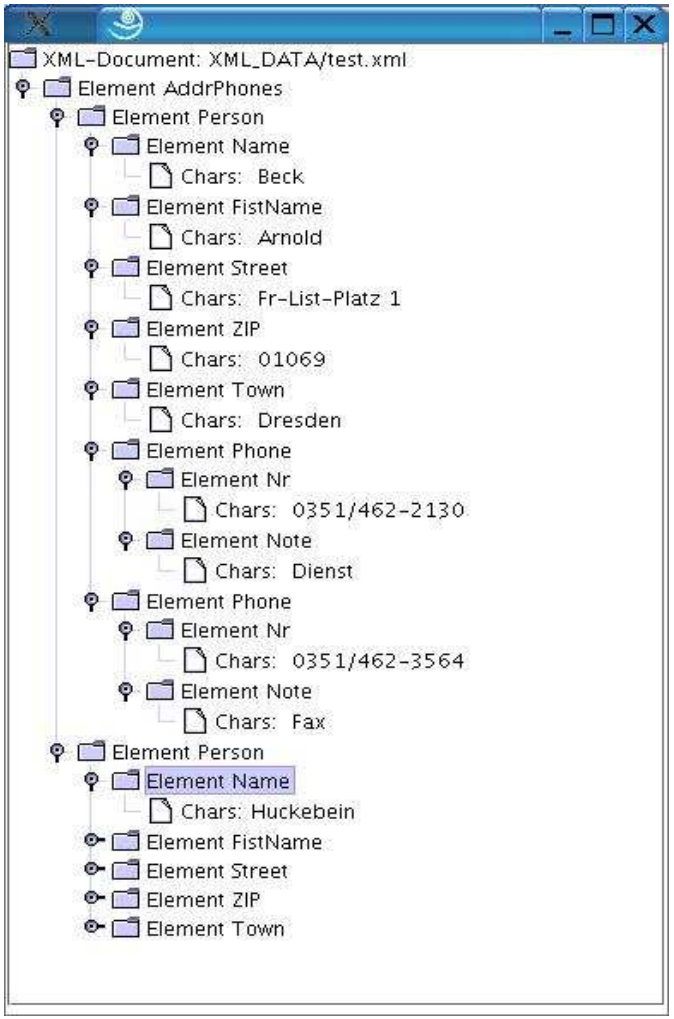
```
  public void setDocumentLocator(Locator locator){L=locator;}
  public void skippedEntity(String name){System.out.println("Skipped: "+name+"\n");}
  public void startDocument(){System.out.println("StartDoc\n");}
  public void startElement(String namespaceURI,
                           String localName,
                           String qName,
                           Attributes atts)
  {
    System.out.println("StartElem: "+namespaceURI+' '+localName+' '+qName+' '+atts+"\n");
  }
  public void startPrefixMapping(String prefix, String uri){NameSpaces.put(uri,prefix);}
});
```

```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;


class xmlsax extends JPanel
{
  JTree T;
  String S="";
  int idx=0;
  private String vendorParserClass="org.apache.xerces.parsers.SAXParser";
  //private String vendorParserClass="org.xml.sax.Parser";
  Map NameSpaces=new HashMap();

  DefaultMutableTreeNode Curr;
```

```
xmlsax(String I) // Name einer XML-Datei oder URL (URI)
{
  DefaultMutableTreeNode base=new DefaultMutableTreeNode("XML-Document: "+I); //Wurzelelement eines Tree
  T=new Jtree(base); // neuen Baum mit base als Wurzelelement erzeugen

  Curr =base;

  try

  {
    InputSource S=new InputSource(I);
    S.setSystemId(I);
    XMLReader myReader = XMLReaderFactory.createXMLReader(vendorParserClass); // XML-Reader (SAX-Parser) erzeugen
    myReader.setContentHandler (new org.xml.sax.ContentHandler()
    {
      Locator L;
      public void characters(char[] ch, int start, int length)
                {
                if (ch[start]!='\n'&&length>1)
                {
                  System.out.println("Chars: "+new String (ch,start,length)+"\n");
                  DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Chars: "+new String (ch,start,length));
                  Curr.add(pi);
                  }
              }
      public void endDocument(){System.out.println("EndDoc\n");}
      public void endElement(String namespaceURI, String localName, String qName)
                {
                System.out.println("EndElem: "+namespaceURI+' '+localName+' '+qName+"\n");
                Curr=(DefaultMutableTreeNode)Curr.getParent();
              }
```

```
      public void endPrefixMapping(String prefix)
                {
                for (Iterator i=NameSpaces.keySet().iterator();i.hasNext();)
                  {
                    String Uri =(String)i.next();
                    String Pref=(String)NameSpaces.get(Uri);
                    if (prefix.equals(Pref)){NameSpaces.remove(Uri);break;}
                  }
                }
      public void ignorableWhitespace(char[] ch, int start, int length){}
      public void processingInstruction(String target, String data)
                {
                DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Instruction: target: "+target+" Data: "+data);
                Curr.add(pi);
               }
      public void setDocumentLocator(Locator locator){L=locator;}
      public void skippedEntity(String name){System.out.println("Skipped: "+name+"\n");}
      public void startDocument(){System.out.println("StartDoc\n");}
      public void startElement(String namespaceURI, String localName, String qName, Attributes atts)
                {
                System.out.println("StartElem: "+namespaceURI+' '+localName+' '+qName+' '+atts+"\n");
                DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Element "+localName);
                Curr.add(pi);  Curr=pi;
               }
      public void startPrefixMapping(String prefix, String uri){NameSpaces.put(uri,prefix);}
    });
    setLayout(new BorderLayout());
    myReader.parse(S);
    add(new JScrollPane(T));
  }
  catch (Exception e)
  { System.out.println(e); }
}
```

```
  public static void main(String args[])
  {
    Frame F=new Frame();
    try
    {

      xmlsax P=new xmlsax(args[0]);
      F.add (P);
      F.setSize(500,200);
      F.setVisible(true);
      F.addWindowListener(new WindowAdapter()
            {public void windowClosing(WindowEvent e)
               {System.exit(0);}
            });
      System.out.println("ready");
    }
    catch (Exception e)
    { System.out.println(e); }
  }
}
```



DOM

- DOM ist im eigentlichen Sinne kein Parser für XML-Dokumente. Beim Einlesen eines XML-Dokumente wird dieses jedoch geparst.

- DOM hält im Speicher den kompletten Datenbestand eines XML-Dokumentes in Form eines Baumes ist also eher eine interne Datenrepräsentation von XML-Daten

- Man kann durch die Daten eines DOM-Baumes navigieren

- Man kann einen DOM-Baum auch synthetisieren und danch als XML-Dokument speichern

- Um aus einer Datei ein Dokument zu erhalten gibt es 2 Wege:

  Old Style:

```
DocumentBuilder B=DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document D=B.parse(I);
```

  New Style (Xerces)

```
DOMParser B=new DOMParser();
B.parse(I);
Document D=B.getDocument();
```

- Die einzelnen Knoten des Baumes erhält man mit:

```
NodeList NL=D.getDocumentElement().getChildNodes();
```

- Wichtige Klassen/Interfaces sind:

Klasse/Interface	*Verwendung*
Document	Repräsentiert den gesamten DOM-Baum
Element	Knoten im DOM-Baum mit ggf. Attributen
Node	Knoten im Baum, es gibt viele verschiedene Knotentypen
NodeList	Eine Sammlung von Knoten

- Knotentypen
  - **DOCUMENT_NODE**
  - **ELEMENT_NODE**
  - **TEXT_NODE**
  - CDATA_SECTION_NODE
  - COMMENT_NODE
  - PROCESSING_INSTRUCTION_NODE
  - ENTITY_REFERENCE_NODE
  - DOCUMENT_TYPE_NODENODE

- Ein Node vom Typ ELEMENT_NODE lässt sich nach Element casten, da Element von Node abgeleitet ist.

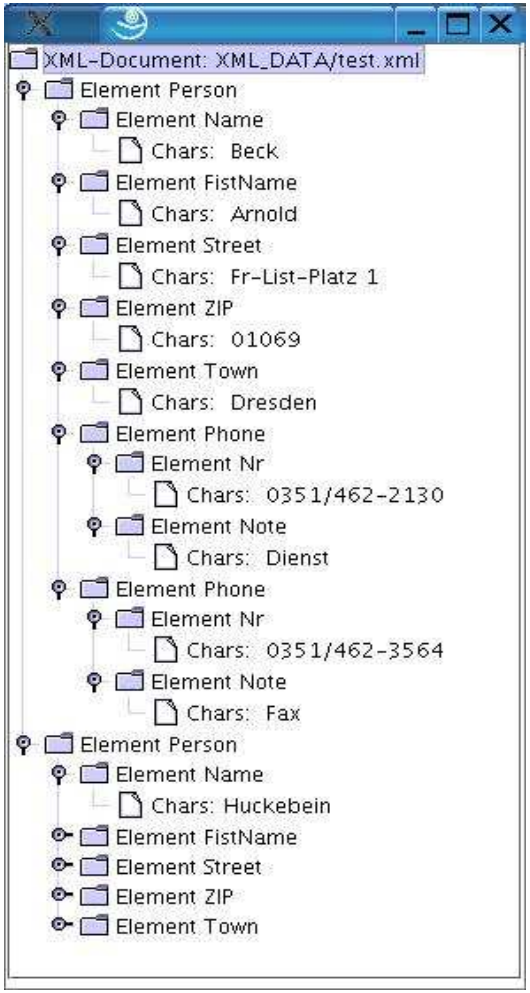
```
void outNodeList(NodeList NL, DefaultMutableTreeNode base)
{
    DefaultMutableTreeNode Curr=base; int idx; Node N=null;
    for (idx=0;(N=NL.item(idx))!=null; idx++)
    {
      switch (N.getNodeType())
      {
       case Node.TEXT_NODE:
       {
         System.out.println("Text:"+N.getNodeName()+" "+N.getNodeValue());
         if(!(onlyWhiteSpace(N.getNodeValue())))
         {
          DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Chars: "+(String)N.getNodeValue());
          Curr.add(pi);
         } break;
       }
       case Node.ELEMENT_NODE:
       {
        DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Element "+N.getNodeName());
        Curr.add(pi);  if (N.hasChildNodes()) outNodeList(N.getChildNodes(), pi );
       } break;
       default:
       {
        DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Other");
        Curr.add(pi);
       } break;
   }
  }
 }
```

```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;

class xml01 extends JPanel
{
  Document D;
  DocumentBuilder B;
  JTree T;

  boolean onlyWhiteSpace(String S)
  {
    int i=0;
    try
    {
    while(Character.isWhitespace(S.charAt(i))) i++;
    }
    catch (Exception e){return true;}
    return false;
  }
```

```
void outNodeList(NodeList NL, DefaultMutableTreeNode base)
{
    DefaultMutableTreeNode Curr=base;  int idx;  Node N=null;
    for (idx=0; (N=NL.item(idx))!=null; idx++)
    {
      switch (N.getNodeType())
      {
        case Node.TEXT_NODE:
        {
          if(!(onlyWhiteSpace(N.getNodeValue())))
          {
            DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Chars: "+N.getNodeValue());
            Curr.add(pi);
          }
        }break
        case Node.ELEMENT_NODE:
        {
          DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Element "+N.getNodeName());
          Curr.add(pi);
          if (N.hasChildNodes()) outNodeList(N.getChildNodes(), pi );
        } break;
        default:
        {
          DefaultMutableTreeNode pi=new DefaultMutableTreeNode("Other");
          Curr.add(pi);
        } break;
     }
    }
}
```

```
xml01(String I)
{
  try
  {
    DefaultMutableTreeNode base=new DefaultMutableTreeNode("XML-Document: "+I);
    T=new JTree(base);
    setLayout(new BorderLayout());
    add(new JScrollPane(T),BorderLayout.CENTER);

    //OldStyle:   B=DocumentBuilderFactory.newInstance().newDocumentBuilder();
    //            D=B.parse(I);

    // New Style (Xerces)
    DOMParser B=new DOMParser();
    B.parse(I);
    D=B.getDocument();
    NodeList NL=D.getDocumentElement().getChildNodes();
    outNodeList(NL,base);
  }
  catch (Exception e)
  { System.out.println(e); }
}
```

```
  public static void main(String args[])
  {
    JFrame F=new JFrame();
    try
    {

      xml01 P=new xml01(args[0]);
      F.getContentPane().add (P);
      F.setSize(500,200);
      F.setVisible(true);
      F.addWindowListener(new WindowAdapter()
             {public void windowClosing(WindowEvent e)
               {System.exit(0);}
             });
      System.out.println("ready");
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }
}
```



Erzeugung eines DOM-Baumes

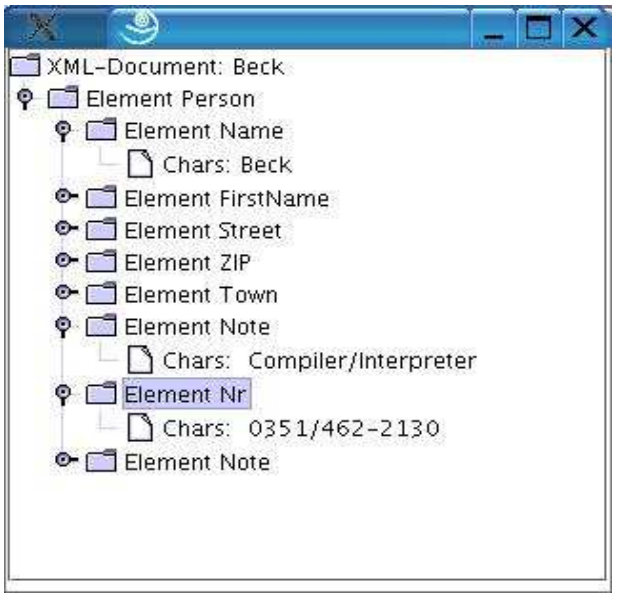
Die Erzeugung eines DOM-Baumes wird nun in das xmlDom-Beispiel aufgenommen und ersetzt das Einlesen einer XML-Datei.

Daten:
```
Beck, Arnold, Friedrich-List-Platz 1, 01069, Dresden, Compiler/Interpreter, 0351/462-2130, Dienst
```

Import:
```
import org.apache.xerces.dom.DOMImplementationImpl;
```

```
  String X[]={"Name","FirstName","Street","ZIP","Town","Note","Nr","Note"};

  xml02(String I)
  {
    try
    {
      DefaultMutableTreeNode base=new DefaultMutableTreeNode
                    ("XML-Document: "+new StringTokenizer(I,",").nextToken());
      DefaultTreeModel DTM=new DefaultTreeModel(base);
      T=new JTree(DTM);
      TM   =TM;

      setLayout(new BorderLayout());
      add(new JScrollPane(T),BorderLayout.CENTER);
```

```
  //Aufbauen DOM-Baum

  DOMImplementation domImpl=new DOMImplementationImpl();
  D = domImpl.createDocument(null,"PersList",null);
  Element root = D.getDocumentElement();
  StringTokenizer T=new StringTokenizer(I,",");
  int i=0;
  Element Pers=D.createElement("Person");
  root.appendChild(Pers);
  while (T.hasMoreElements())
  {
    Element ex=D.createElement(X[i++]);
    Pers.appendChild(ex);
    String  S=T.nextToken();
    Text    tx=D.createTextNode(S);
    System.out.println(S);
    ex.appendChild(tx);
  }

  // Ausgabe DOM-Baum

  DOMParser B=new DOMParser();
  NodeList NL=D.getDocumentElement().getChildNodes();
  Node N=null;
  outNodeList(NL,base);
}...
```



Die erzeugte XML-Datei:

```
<?xml version="1.0"?>
<PersList>
<Person>
<Name>Beck</Name>
<FirstName> Arnold</FirstName>
<Street> Friedrich-List-Platz 1</Street>
<ZIP> 01069</ZIP>
<Town> Dresden</Town>
<Note> Compiler/Interpreter</Note>
<Nr> 0351/462-2130</Nr>
<Note> Dienst</Note>
</Person>
</PersList>
```

Java & JDBC

Ziel:

Umwandeln eines beliebigen Resultsets in eine XML-Repräsentation
Beispiel:
Nach dem SQL-Statement erhält man ein ResultSet

PersID	***Name***	***FName***	***Street***	***Zip***	***Town***	***Note***
xxx	Beck	Prof. Dr.-Ing Arnold	Fr.-List-Platz 1	01069	Dresden	Dienst
yyy	Beck	Augustin	...	...	...	

Für die Telefonnummer erhält man ebenfalls je Person ein Resultset

PersID	***TelID***	***Phone***	***Note***
xxx	n1	0351 / 462 - 2130	
xxx	n2	0351 / 462 – 3564	fax

Draus entsteht nun die XML-Repräsentation:

```
<?xml version="1.0"?>
<Result>
. . .
<Person>
<Name>Beck</Name>
<FName>Prof. Dr.-Ing. Arnold</FName>
<Street>Friedrich-List-Platz 1</Street>
<ZIP>01069</ZIP>
<Town>Dresden</Town>
<Note>Dienst HTW</Note>
<TeleB>
<Phone>0351 / 462-2130</Phone>
<Note></Note>
</TeleB>
<TeleB>
<Phone>0351 / 462-3564</Phone>
<Note>fax</Note>
</TeleB>
</Person>
. . .
</Result>
```

```
// Funktion, die ein Telefonnummern Resultset in XML umwandelt
// eignet sich, um aus einem „beliebigen“ Resultset XML zu erzeugen

void resultSetToXML(ResultSet rs, Document D,Element e)
{
  try
  {
    // Ermitteln der ResultsetMetadaten (Tabellenname, Spaltennamen, Spaltentypen)
    ResultSetMetaData RM=rs.getMetaData();
    int ColCount=RM.getColumnCount();
    String ColNames  []=new String[ColCount];
    String ClassNames[]=new String[ColCount];

    for (int i=1;i<=ColCount;i++)
    {
        ColNames[i-1]=RM.getColumnName(i);
        ClassNames[i-1]=RM.getColumnClassName(i);
        System.out.println("\n"+ColNames[i-1]+"\n"+ClassNames[i-1]);
    }
```

```
    while(rs.next())
    {
      int ID=0;
      Element e1=D.createElement(RM.getTableName(1));
      e.appendChild(e1);
      for (int ic=0;ic<ColNames.length; ic++)
      {
       // Resultset auslesen und Ergebnisse in XML-Baum schreiben
       String  S="";
       if (ClassNames[ic].endsWith("Long"  )) S=""+(ID=(int)(rs.getInt(ic+1))); else
       if (ClassNames[ic].endsWith("String")) S=rs.getString(ic+1);       else
       if (ClassNames[ic].endsWith("Date"  )) S=rs.getString(ic+1);
       if (S==null)S="< nix >";
       if (ClassNames[ic].endsWith("String")) // Nur Strings in DOM-Baum übernehmen
       {
         Element e2=D.createElement(ColNames[ic]);
         e1.appendChild(e2);
         Text    tx=D.createTextNode(S);
         e2.appendChild(tx);
       }
       System.out.println(ColNames[ic]+":"+S);
      }
    }
  }catch (Exception ex){System.out.println(ex); ex.printStackTrace();}
}
```

Suche nach Informationen im DOM-Baum

Testszenario:
- Nach einem Query (Name=Beck) in der Adresstabelle wird ein Resultset erzeugt.
- Dieses Resultset wird in einen DOM-Baum umgewandelt.
- Der DOM-Baum wird nach den Tags „FName" durchsucht.
- Das Ergebnis ist eine Nodeliste aller Fname-Tags. Im KindNode steht dann jeweils der eigentliche Vorname.
- Über die Methode getParentNode können dann weitere Informationen erreicht werden
- D ist das Document

```
NodeList NL=D.getElementsByTagName("FName");
String S="";
for(int i=0; i<NL.getLength();i++)
  S+="\n->"+NL.item(i).getFirstChild().getNodeValue();
System.out.println(S);


public Node getParentNode()
```