# Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

XML

JAXP

SAX

DOM

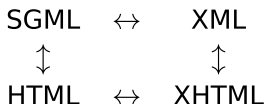
# Einführung in XML

XML ist eine *Auszeichnungssprache* zur strukturellen Beschreibung von Dokumenten.

Refenzdefinition ist die *XML Recommendation* vom W3C (World Wide Web Consortium), erste Fassung von 1998.

XML sieht zwar aus wie HTML (spitze Klammern), im eigentlich Sinne hat XML nicht direkt mit HTML zu tun.

Korrekte Analogie:



# Grundlegende Definitionen

Ein XML-Dokument heißt

**wohlgeformt** wenn es *syntaktisch* korrekt ist:

- Nur ein Wurzelelement.
- Jeder geöffnete Tag geht wieder zu.
- Kein „Überlappen" über Tag-Grenzen hinweg.
- Kein Element mit doppeltem Attribut mit gleichem Namen.

**gültig** wenn es gegen eine *semantische* Beschreibung validiert, d. h. einer inhaltlichen Beschreibung entspricht, welche Elemente ein XML-Dokument für einen bestimmten Zweck wann und wie enthalten darf.

# Schemasprachen

Eine Schemasprache beschreibt für einen bestimmten Zweck, wie ein gültiges XML-Dokument auszusehen hat.

Hier sind weit verbreitet:

DTD
: (*Document Type Definion)* alter Standard, in SGML, wenig Features (keine Wertebereiche, keine Key-Constraints)

XML Schema
: (XSD) neuerer Standard, in XML, erlaubt z. B. Angabe von Wertebereichen, Constraints für Schlüssel etc.

Relax NG
: nach Meinung der Befürworter einfach und elegant, aber wenig verbreitet.

# Verarbeitung von XML

Um XML (z. B. in einem Java-Programm) zu verarbeiten, braucht man einen *Parser*.

Traditionell existieren zwei Modelle zum Parsen und Verarbeiten:

SAX
: als *ereignisorientierte* Methode, bei der entsprechende Aktionen (*hooks*) beim Start und Ende eines Tags ausgelöst werden. Dies ist eine schnelle und schlanke Methode, da nicht das gesamte Dokument im Speicher gehalten werden muss.

DOM
: als Aufbau ein DOM-Tree (DOM = *Document Object Model*), wobei das gesamte Dokument analysiert wird und so wahlfreien Zugriff auf alle Knoten des Baums erlaubt. Aufwändiger, aber mächtiger als SAX.

# JAXP

JAXP (*Java API for XML Processing* ist die Standard-API von Java zum Parsen von Dokumenten mit SAX oder DOM.

Zusätzlich erlaubt JAXP Transformationen von XML-Dokumenten mit XSLT und die Validierung von Dokumenten anhand eines gegebenen XML Schemas.

Referenzimplementierung von JAXP ist aktuell das Projekt *Xerces-J*, entstanden als Apache-Projekt (nebenbei existiert eine Implementierung von Xerces für C++).

Webseite: `http://xerces.apache.org/xerces2-j/`

# SAX

SAX (*Simple API for XML*) ist eine ereignisorientierte Methode zum Parsen von XML-Dateien.

Dies bedeutet, das beim Parsen definierte Callbacks ausgeführt werden, wenn

- das Dokument beginnt oder endet,
- Elemente beginnen oder enden,
- Text-Inhalt gelesen wird.

Die Reihenfolge des Aufrufens der Callbacks ist fest durch die Reihenfolge der Elemente im Dokument vorgegeben.

SAX ist daher unflexibel, aber spart Ressourcen, da nicht das komplette XML-Dokument im Speicher gehalten werden muss.

# Beispiel-Schema kontakt.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="kontakte" type="rootType"></xsd:element>
    <xsd:complexType name="rootType">
        <xsd:sequence>
            <xsd:element name="person" type="personType"
                maxOccurs="unbounded" minOccurs="0">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="personType">
        <xsd:sequence>
            <xsd:element name="vorname" type="xsd:string"/>
            <xsd:element name="nachname" type="xsd:string"/>
            <xsd:element name="firma" type="xsd:string"
                maxOccurs="1" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="vip" type="xsd:boolean"
            use="optional" default="false"/>
    </xsd:complexType>
</xsd:schema>
```

# Beispiel-Daten personen.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kontakte xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="kontakt.xsd">
    <person vip="true">
        <vorname>Klaus</vorname>
        <nachname>Höppner</nachname>
        <firma>GSI</firma>
    </person>
    <person>
        <vorname>Karl</vorname>
        <nachname>Schmidt</nachname>
    </person>
    <person>
        <vorname>Maria</vorname>
        <nachname>Mustermann</nachname>
        <firma>Merck</firma>
    </person>
</kontakte>
```

# Das Interface ContentHandler

Beim Parsen mit SAX wird dem Parser ein `ContentHandler` übergeben. Im Interface `ContentHandler` sind dabei die Methoden abstrakt definiert, die beim Auftreten der entsprechenden Ereignisse aufgerufen werden sollen:

startDocument beim Beginn des XML-Dokumentes,

endDocument beim Ende des XML-Dokumentes,

startElement beim öffnenden Tag eines Elementes, hierbei werden der Name und die Attribute des Elements übergeben,

endElement beim schließenden Tag eines Elemtes,

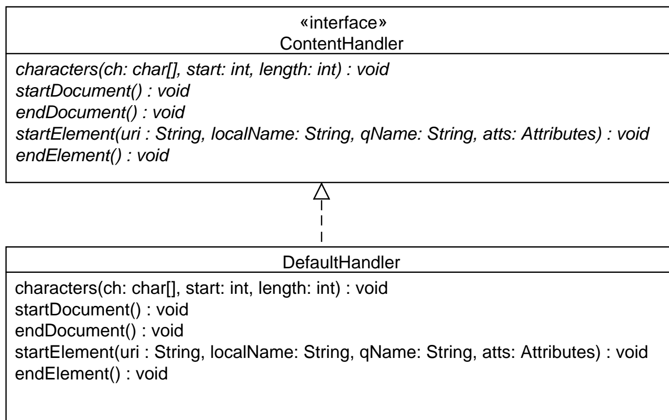
characters für normalen Text.

# Die Klasse `DefaultHandler`

Wie bereits an anderen Stellen in Java existiert eine Klasse, die das Interface implementiert, indem die fraglichen Methoden „leer“ definiert werden.

In diesem Fall implementiert die Klasse `DefaultHandler` das Interface `ContentHandler`. Hiervon erben selbst definierte Klassen für Handler, in denen dann die Methoden nach Wunsch überladen werden können.

# UML-Diagramm



# Grundlegendes Beispiel

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class MyXMLParser1 extends DefaultHandler {
    public static void main(String[] args) {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse("personen.xml", new MyXMLParser1());
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

# Grundlegendes Beispiel (Forts.)

```java
@Override
public void endDocument() throws SAXException {
    System.out.println("Ende des Dokumentes");
}

@Override
public void endElement(String uri, String localName,
                                       String qName) {
    System.out.format("Ende von Element %s\n", qName);
}

@Override
public void startDocument() throws SAXException {
    System.out.println("Start des Dokumentes");
}
```

# Grundlegendes Beispiel (Forts.)

```
    @Override
    public void startElement(String uri, String localName,
        String qName, Attributes atts) throws SAXException {
        System.out.format("Start von Element %s\n",qName);
        for (int i=0; i<atts.getLength(); i++) {
            System.out.format("Attr: %s=%s\n",
                atts.getQName(i), atts.getValue(i));
        }
    }

    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        System.out.println(String.copyValueOf(ch, start, length));
    }
}
```

# SAX-Beispiel: HTML-Tabelle

Im Folgenden soll ein SAX-Contenthandler geschrieben werden, der aus einer XML-Datei mit Kontakten eine HTML-Datei mit tabellarischer Darstellung der Personen erzeugt.

Dieser Contenthandler muss folgende Eigenschaften haben:

- Beim Start oder Ende eines XML-Elementes muss das passende öffnende bzw. schließende HTML-Tag geschrieben werden.
- Text-Inhalt darf nur geschrieben werden, wenn er sich innerhalb einer der Elemente `vorname`, `nachname` oder `firma` befindet, während z. B. der Leerraum zwischen diesen Elementen ignoriert werden soll.

# Implementierung

Im Contenthandler existieren ein Writer, in den der HTML-Code ausgegeben wird, sowie ein Flag, anhand dessen entschieden wird, ob Textinhalt geschrieben wird.

```
public class HTMLWriter extends DefaultHandler {
    private String fname;
    private Writer out = null;
    private boolean writechars = false;

    public HTMLWriter(String fname) {
        this.fname = fname;
    }

    // ...
}
```

# Implementierung (Forts.)

```
@Override
public void startDocument() throws SAXException {
    try {
        out = new FileWriter(fname);
        out.write("<html><body>\n");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void endDocument() throws SAXException {
    try {
        out.write("</body></html>\n");
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

# Implementierung (Forts.)

```
@Override
public void startElement(String uri, String localName, String name
        Attributes attributes) throws SAXException {
    if (name.equals("kontakte")) {
        try {
            out.write("<table border=\"1\">\n");
        } catch (IOException e) { e.printStackTrace(); }
    } else if (name.equals("person")) {
        try {
            out.write("<tr>\n");
        } catch (IOException e) { e.printStackTrace(); }
    } else if (name.equals("vorname") || name.equals("nachname")
            || name.equals("firma")) {
        try {
            out.write("<td>");
        } catch (IOException e) { e.printStackTrace(); }
        writechars = true;
    }
}
```

# Implementierung (Forts.)

```
@Override
public void endElement(String uri, String localName, String name)
        throws SAXException {
    if (name.equals("kontakte")) {
        try {
            out.write("</table>\n");
        } catch (IOException e) { e.printStackTrace(); }
    } else if (name.equals("person")) {
        try {
            out.write("</tr>\n");
        } catch (IOException e) { e.printStackTrace(); }
    } else if (name.equals("vorname") || name.equals("nachname")
            || name.equals("firma")) {
        try {
            out.write("</td>");
        } catch (IOException e) { e.printStackTrace(); }
        writechars = false;
    }
}
```

# Implementierung (Forts.)

```java
@Override
public void characters(char[] ch, int start, int length)
        throws SAXException {
    if (writechars) {
        try {
            out.write(ch, start, length);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# DOM

Bei DOM, dem *Document Object Model*, wird das XML komplett gelesen und liegt als Baum im Speicher vor. Daher werden mehr Ressourcen benötigt, aber da die Verarbeitung nicht an die Reihenfolge im Dokument gebunden ist, ist dieser Weg mächtiger als SAX.

Der DOM-Baum besteht aus Knoten (Interface `Node`), die eine `NodeList` von Kindknoten enthalten können. Jeder Knoten hat dabei einen Typ, und je nach Typ eventuell Name, Wert bzw. Textinhalt.

Die Methode `getNodeType()` gibt den Knotentyp zurück, wobei für jeden Typ ein Subinterface existiert.

# Nodes und Nodetypen

| Interface | Node.XXX_NODE | Art |
|---|---|---|
| Attr | ATTRIBUTE_NODE | Attribute |
| Comment | COMMENT_NODE | Kommentar |
| Document | DOCUMENT_NODE | Das eigentliche Dokument |
| Element | ELEMENT_NODE | Element |
| Text | TEXT_NODE | Text |

`Node.XXX_NODE` ist hierbei eine statische Konstante vom Typ **`short`**, mit der das Ergebnis von `getNodeType()` verglichen werden kann.

# Beispiel

```
import java.io.IOException;
import javax.xml.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class MyDOMParser {
    public static void main(String[] args) {
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder(
            Document doc = builder.parse("personen.xml");
            Element root = doc.getDocumentElement();
```

# Beispiel (Forts.)

```
            NodeList personen = root.getElementsByTagName("person'
            for (int i=0; i<personen.getLength(); i++) {
                Element elem = (Element) personen.item(i);
                Node nachname = elem.
                    getElementsByTagName("nachname").item(0);
                System.out.println(nachname.getTextContent());
                if (elem.getAttribute("vip").equals("true")) {
                    System.out.println("VIP");
                }
            }
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```