11 XML Programmierung

- Programmatische Interfaces (API) zur Manipulation von XML (DOM und JDOM)
- Data Binding: Abbildung von DTD/Schema auf isomorphe Datenstrukturen
- Strombasierte Verarbeitung
- Integration von XML in Programmiersprachen

DSLs für XML

- DSL = domain specific language
- XPath: Spezialsprache zur Navigation in XML Dokumenten
- XSLT: Spezialsprache zur Transformation von XML Dokumenten
- (XQuery: datenbankmäßige Verarbeitung von XML)
- Helfen nur bedingt bei
  - Anwendungen, die XML-Daten lesen und schreiben müssen
  - Anwendungen, die XML-Daten visuell darstellen und editieren
  - Implementierung von XSLT, XQuery, etc

Ansätze zur XML Verarbeitung

- Strings lesen und schreiben
  nicht zufriedenstellend
  komplizierte Aufgabe:
  - Unicode, verschiedene Zeichenkodierungen
  - Wohlgeformtheit
  - Gültigkeit
- Spezialisierte APIs fürs
  - Parsen von XML $\rightarrow$ Repräsentation
  - Navigieren durch XML Repräsentation
  - Manipulieren von XML Repräsentation
  - Serialisierung in XML Text

11.1 Die DOM API

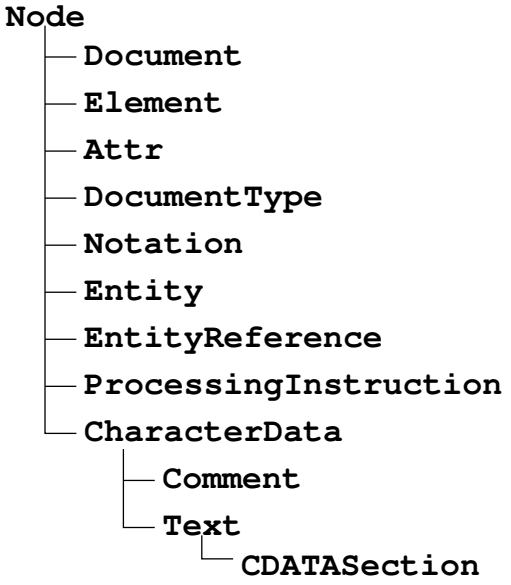
- DOM = Document Object Model
- W3C Empfehlung "DOM Level 3" `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`
- Selbstdarstellung:

  > ... a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of [XML] documents.

- Sprachunabhängigkeit durch Verwendung von IDL (OMG)
- Implementierungen für viele Sprachen: Java, JavaScript, Python, Perl, C#, Fortran, Ada, ...
- Implementiert in vielen Anwendungen (Webbrowser, OpenOffice, XMetaL, ...)

11.1.1 Das DOM Datenmodell

- Low-level Graphdarstellung für XML Dokumente
- Knotentypen (Interfaces, Ausschnitt)



Invarianten des Datenmodells

- Wichtigste Invariante: Graph muss immer eine Menge von Bäumen sein (d.h., höchstens ein Vaterknoten und keine Zyklen)
- Jeder Knoten gehört zu einem `Document` Knoten
- Nicht jeder Knoten darf als Kind eines anderen Knoten auftreten
  - `Document` darf nie als Kind auftreten (muss Wurzel sein)
  - `DocumentType` (höchstens einmal) nur Kind von `Document`
  - `Element`: `Document` (höchstens einmal), `Element`, `Entity`, `EntityReference`
  - `Attr` gilt nicht als Kind des zugehörigen `Element`
  - `Text`: `Element`, `Attr`, `Entity`, `EntityReference`
  - ...
- Invarianten werden nur informell in der API angegeben

11.1.2 XML Verarbeitung mit DOM

- Java Binding

```
Document dom = ...;
Element result = dom.createElement("span");
Attr at = dom.createAttribute("id");
at.nodeValue = "draw" + nr;
result.setAttributeNode (at);
```

Weitere Aspekte

- Navigation: über `Node`-Attribute `parentNode`, `childNodes`, `previousSibling`, `nextSibling`, `attributes`
- Manipulation: über `Node`-Methoden `insertBefore()`, `replaceChild()`, `appendChild()`, `removeChild()`
- Parsen und Serialisieren (Load and Save)
- Validieren gegen DTD und XML Schema
- Auswertung von XPath (1.0)
- Insgesamt etwa 200 Methoden

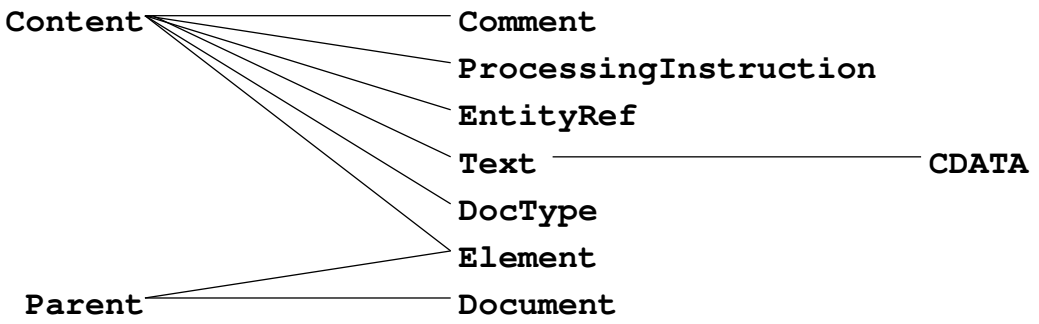
11.2 JDOM

- DOM oft unelegant, da
  - sprachunabhängig
  - universell einsetzbar
  - erfordert Erfahrung
- JDOM
  - spezialisiert für Java
  - 80/20 Prinzip:
    - einfache und häufige Aufgaben unterstützt
    - speziellere Aufgaben schwieriger

11.2.1 JDOM Eigenschaften

- Kollektionen von Elementen und Attributen dargestellt durch `java.util.List` mit `Iterator`
- Datenmodell besteht aus (abstrakten) Klassen und Interfaces (Java)
- Implementierung kann für Java optimiert werden
- API einfacher zu lernen, da die Indirektion über IDL wegfällt

11.2.2 JDOM Datenmodell



- ähnliche Operationen wie DOM
- Bauminvariante wird zur Laufzeit geprüft

Beispiel: Höhe eines XML Baums

```
import java.util.*;
import org.jdom.*;
public class XmlHeight {
    int xmlHeight(Element e) {
        Iterator i = e.getContent.iterator();
        int max = 0;
        while (i.hasNext()) {
            Object c = i.next();
            int h = (c instanceof Element
                    ? xmlHeight((Element)c) : 1);
            if (h>max) max = h;
        }
        return max+1;
    }
}
```

Beispiel: Modifikation eines Dokuments

```
import org.jdom.filter.*;

  static void doubleSugar(Document d)
    throws DataConversionException {
    Namespace rcp = Namespace.getNamespace("http://www.brics.dk/ixwt/re
    Filter f = new ElementFilter("ingredient", rcp);
    Iterator i = d.getDescendants(f);
    while (i.hasNext()) {
      Element e = (Element)i.next();
      if (e.getAttributeValue("name").equals("sugar")) {
        double amount = e.getAttribute("amount").getDoubleValue();
        e.setAttribute("amount", new Double(2*amount).toString());
      }
    }
  }
```

11.2.3 Parsen, Validieren, Serialisieren

- JDOM enthält keinen Parser
- `import org.jdom.input.*;`
  Kann Dokumente aufbauen aus
  - externem SAX-Parser (Voreinstellung: JAXP) oder
  - existierendem DOM Dokument
- Validierung nur beim Parsen, nicht im Speicher
- `import org.jdom.output.*;`
  Ausgabe als
  - DOM Dokument
  - SAX-Events
  - XML-Strom

Beispiel: XML Lesen, Ändern, Schreiben

```
static void doit(String[] args)
    throws Exception {
  SAXBuilder b = new SAXBuilder();
  Document d = b.build(new File("recipes.xml"));
  Namespace rcp =
    Namespace.getNamespace("http://www.brics.dk/ixwt/recipes");
  d.getRootElement()
   .getChild("description", rcp)
   .setText("Cool␣recipes!");
  XMLOutputter outputter = new XMLOutputter();
  outputter.output(d, System.out);
}
```

Beispiel: . . . mit Validierung

```
SAXBuilder b = new SAXBuilder();
b.setValidation(true);
String msg = "No errors!";
try {
  Document d = b.build(new File(args[0]));
} catch (JDOMParseException e) {
  msg = e.getMessage();
}
```

11.2.4 JDOM mit XPath und XSLT

- Hooks für
  - XPath 1.0 Auswertung
    - Package `org.jdom.xpath`
    - Implementierung kann zur Laufzeit gesetzt werden
  - XSLT Transformation
    - Package `org.jdom.transform`
    - Implementierung kann zur Laufzeit gesetzt

Beispiel: JDOM mit XPath

```
import org.jdom.xpath.*;

  static void doubleSugar(Document d)
    throws JDOMException {
    XPath p = XPath.newInstance ("//rcp:ingredient[@name='sugar']");
    p.addNamespace("http://www.brics.dk/ixwt/recipes");
    Iterator i = p.selectNodes(d).iterator();
    while (i.hasNext()) {
      Element e = (Element)i.next();
      if (e.getAttributeValue("name").equals("sugar")) {
        double amount = e.getAttribute("amount").getDoubleValue();
        e.setAttribute("amount", new Double(2*amount).toString());
      }
    }
  }
```

Beispiel: JDOM mit XSLT/Xalan

```
import org.jdom.transform.*;

static void applyXSLT(String file, String sheet)
    throws Exception {
  System.setProperty("javax.xml.transform.TransformerFactory",
          "org.apache.xalan.processor.TransformerFactoryImpl");
  SAXBuilder b = new SAXBuilder();
  Document d = b.build(new File(file));
  XSLTransformer t = new XSLTransformer(sheet);
  Document h = t.transform(d);
  XMLOutputter outputter = new XMLOutputter();
  outputter.output(h, System.out);
}
```