Hochschule **Rosenheim**
University of Applied Sciences

Verteilte Verarbeitung

Kapitel 4
Serialisierung in XML
(noch kein Video)

Lernziele

- Sie wissen …
  - Was Serialisierung bedeutet
  - Wie Sie ***manuell*** Objekte in Java serialisieren
  - Wie Sie die ***eingebaute Serialisierung*** nutzen
  - Wie Sie Objekte in ***XML und JSON***-Strukturen serialisieren
  - Was das Serializer Pattern ist

XML und Serialisierung (Java: JAXB)

XML

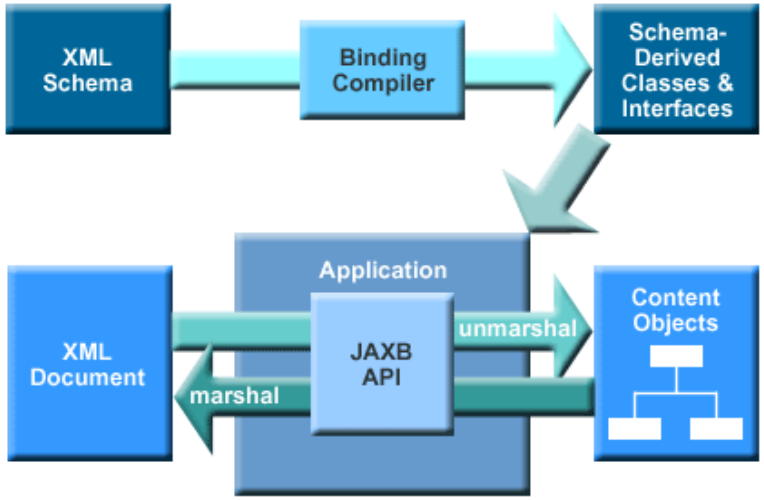
- XML = Extensible Markup Language
- idR. im UTF8-Format gespeichert (entspr. Ascii)
- = ***Selbstbeschreibende Baumstruktur***

```
<kunde>
  <nummer>4711</nummer>
  <name>Hugo Habicht</name>
  <adresse>
      <ort>Clausthal-Zellerfeld</ort>
      <!-- mehr -->
  </adresse>
</kunde>
```

- Elemente: `<element> … </element>`
- Attribute: `<element attribut1=„huhu“ …> </element>`

XML Serialisierung in Java: JAXB

- *Java Architecture for XML Binding (**JAXB**)*
  - Bidirektionale Abbildung von XML-Dokumenten auf Java-Objekte
  - Auf DTD/XML Schema-Basis generiert ein JAXB-Compiler Satz von Java-Klassen
  - XML-Dokumente lassen sich als Java-Objekt-Baum abbilden und manipulieren



- Notwendig zur Serialisierung (Marshalling) von Parametern und Rückgabewerten
- Bestandteil von ***Java SE 6*** und Java EE 5

Quelle der Abbildung:
http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

Annotationen an einer Klasse

```
@XmlRootElement(name="Kunde")
@XmlAccessorType(XmlAccessType.NONE)
public class Kunde {

    @XmlAttribute(name="number", required=true)
    private String nummer;

    @XmlElement(name="fullname", required=false, defaultValue="Hugo")
    private String name;

    @XmlElement(name="companyaddress")
    private Adresse firmenAdresse;

    @XmlTransient
    private Kunde geworbenerKunde;

    // … Getter und Setter, Konstruktor (usw)
    @XmlList  // Kompakt,
    // sonst @XmlElement, evtl. mit @XmlElementWrapper
    private List<String> hobbies;
}
```

Wurzelelement XML-Baum

nur gekennzeichnete Attribute serialisieren

Als XML Attribut

Als XML Element

Nicht serialisieren

Als XML Element (Liste)

Erzeugte XML Datei (Beispiel)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Kunde number="4711">
   <fullname>Hugo Habicht</fullname>
   <companyaddress>
      <ort>Clausthal-Zellerfeld</ort>
      <plz>35678</plz>
      <strasse>Schütteweg 3</strasse>
   </companyaddress>
   <hobbies>Tontauben Staubsauger Fernseher</hobbies>
</Kunde>
```

JAXB Tags
Allgemeine Serialisierungsanweisungen

- **`@XmlRootElement`**
  - = Java Klasse kann XML Wurzelknoten sein
- **`@XmlType`**
  - Bindung der Java-Klasse an ein XML Schema
- **`@XmlAccessorOrder`**
  - = Reihenfolge der Serialisierung z.B. **`XmlAccessorOrder.ALPHABETICAL`**
- **`@XmlAccessorType`**
  - Bestimmt Zugriff auf die Attribute einer Klasse

`XmlAccessType.NONE:` nur gekennzeichnete Attribute serialisieren
`XmlAccessType.FIELD`: alle Attribute (auch private) werden serialisiert
`XmlAccessType.PROPERTY`: nur Attribute mit public get / set Methoden
`XmlAccessType.PUBLIC_MEMBER`: nur public Attribute

JAXB Tags /2
Serialisierung der Attribute

- **`@XmlAttribute`**
  - Kennzeichnet Attribut als serialisierbar (XML-Attribut)
  - Beispiel **`<AndererTag Attribut=Wert>…</AndererTag>`**
- **`@XmlElement`** (XML Element)
  - Kennzeichnet Attribut oder Get/Set-Methode als Serialisierbar
  - Beispiel: **`<Elementname>Wert</Elementname>`**
  - Beispiel für Liste:
    `<Elementname>Wert1</Elementname>`
    `<Elementname>Wert2</Elementname>`…
- **`@XmlList`**
  - Kennzeichnet Listen als serialisierbar List<String>, kompakte form
  - Beispiel: **`<Listenname>Wert1 Wert2 Wert3</Listenname>`**
- **`@XmlElementWrapper(name= …)`**
  - Eigener Name für **`@XmlElement`** - Listen
  - Beispiel:
    `<WrapperName>`
    `<Elementname>Wert1</Elementname>` …
    `</WrapperName>`

Steuerung der (de)Serialisierung

Beispiel:

```
@XmlElement(
```

- XML nicht unbedingt 1:1 auf Klassen- und Attributnamen abgebilden (Attribut “name”)

```
    name="fullname",
```

- XML Elemente (nicht) zwingend gefüllt

```
    required=false,
```

- Defaultwerte sind möglich

```
    defaultValue="Hugo")
```

Marshalling (= Serialisierung)

```
Kunde hugo = …;
FileOutputStream file = new FileOutputStream(
   new File("kunde.xml"));

JAXBContext ctx = JAXBContext.newInstance(Kunde.class);

Marshaller m = ctx.createMarshaller();
m.marshal(hugo, file);

file.close();
```

Unmarshalling (= Deserialisierung)

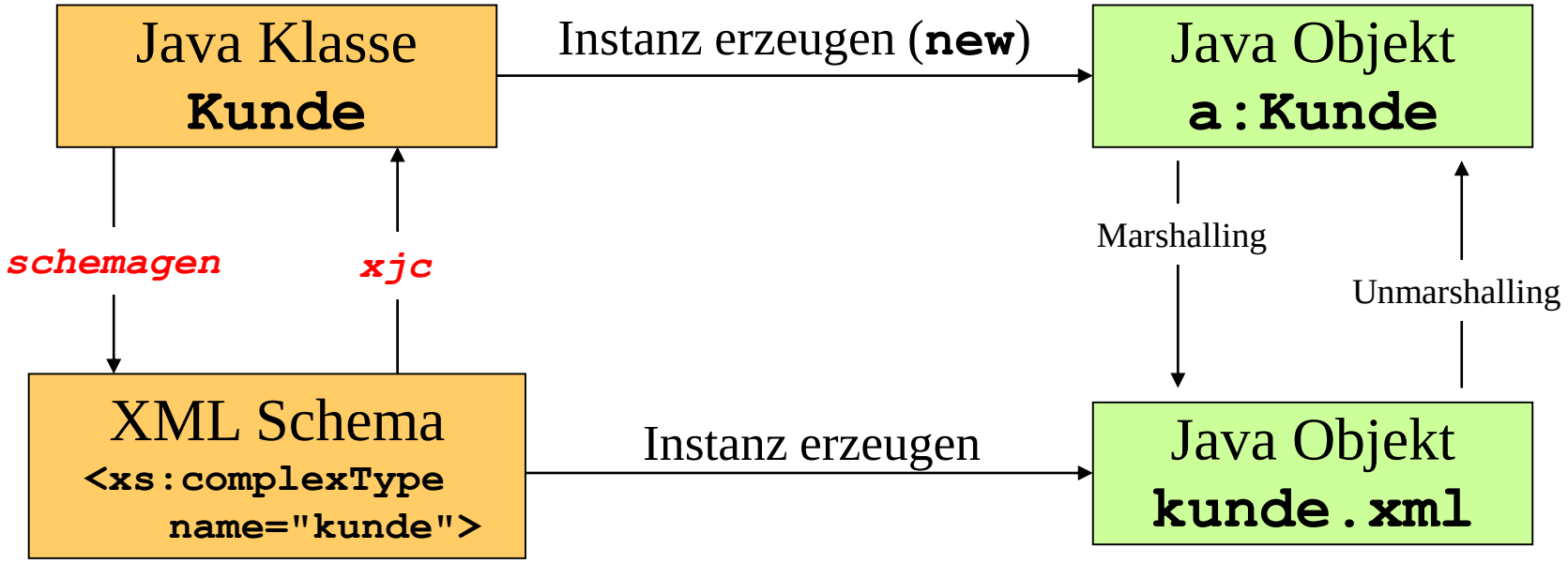
```
FileInputStream file = new FileInputStream(
    new File("kunde.xml"));

JAXBContext ctx = JAXBContext.newInstance(Kunde.class);

Unmarshaller u = ctx.createUnmarshaller();
Kunde k = (Kunde) u.unmarshal(fileIn);

file.close();
```


Java und XML


Java Klasse
Kunde
Instanz erzeugen (new)
Java Objekt
a:Kunde
schemagen
xjc
Marshalling
Unmarshalling
XML Schema
<xs:complexType
name="kunde">
Instanz erzeugen
Java Objekt
kunde.xml

Was sind „erlaubte“ xml-Dokumente

- ***Wohlgeformte*** XML-Dokumente
  - Halten XML Konventionen ein z.B.
    `<bla>` … **`</bla>`** nicht **`<bla>`** … **`</blub>`**
  - Konventionen: Siehe oben
- ***Gültige*** XML-Dokumente
  - Haben einen lokalen oder zentral festgelegten Dokumenttyp
  - Halten die Einschränkungen des Dokumenttyps ein
  - Dokumenttyp wird festgelegt entweder als DTD oder als XSD
- Eigenschaften werden durch XML-Parser geprüft

Inhalte des XML-Schemas

- Ein XML-Schema ist ein XML-Dokument

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 … <!-- hier die Infos -->
<xs:schema/>
```

- Deklaration eines Elements

```
<xs:element name="firmenAdresse" type="adresse"/>
```

- Elementare Datentypen sind vorhanden:
  xs:integer, xs:string, xs:date, …

- Deklaration eines eigenen nicht-strukturierten Datentyps auf der Basis eines vorgegebenen Datentyps wie xs:string oder xs:integer

```
<xs:simpleType name="weinFarbe">
   <xs:restriction base="xs:string">
       <xs:enumeration value="WEISS"/>
       <xs:enumeration value="ROT"/>
       <xs:enumeration value="ROSE"/>
                  <!- auch schön: <xs:pattern value="[0-9]{5}"/>
   </xs:restriction>
</xs:simpleType>
```

Inhalte des XML-Schemas

- Deklaration eines eigenen strukturierten Datentyps
  - Teilelemente <xs:sequence><xs:element … > …
  - Attribute <xs:attribute>
  - Elemente Pflicht **`minOccurs= "1"`** oder optional **`minOccurs="0"`**

```
<xs:complexType name="adresse">
  <xs:sequence>
     <xs:element name="ort" type="xs:string" minOccurs="0"/>
     <xs:element name="plz" type="xs:string" minOccurs="0"/>
     <xs:element name="strasse" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

XML-Schema einmal hin und zurück

- **`xjc`**: Erzeugt aus XML Schema Java Klassen

  `xjc -d src -p de.fhr.vv.xml.gen` ***`order.xsd`***

  -d = Verzeichnis, wo die Quelltexte liegen sollen

  -p = Package der generierten Klassen

- **`schemagen`**: Erzeugt aus Java Klassen XML Schema

```
schemagen -cp .\target\classes
    .\src\main\java\de\fhr\inf\vv\exp2\xml\Kunde.java
```

-cp = (Classpath) wo sind die anderen Dateien

Validierung gegen ein XML Schema

- XML Schema wird im Wurzelknoten mit angegeben
- Validierung (Syntax Prüfung) der Datei damit möglich
- Gute Unterstützung durch Frameworks / Werkzeuge
- Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<Kunde
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="kunde.xsd">
…
</Kunde>
```

Unmarshalling (= Deserialisierung)
Mit Prüfung gegen Schema

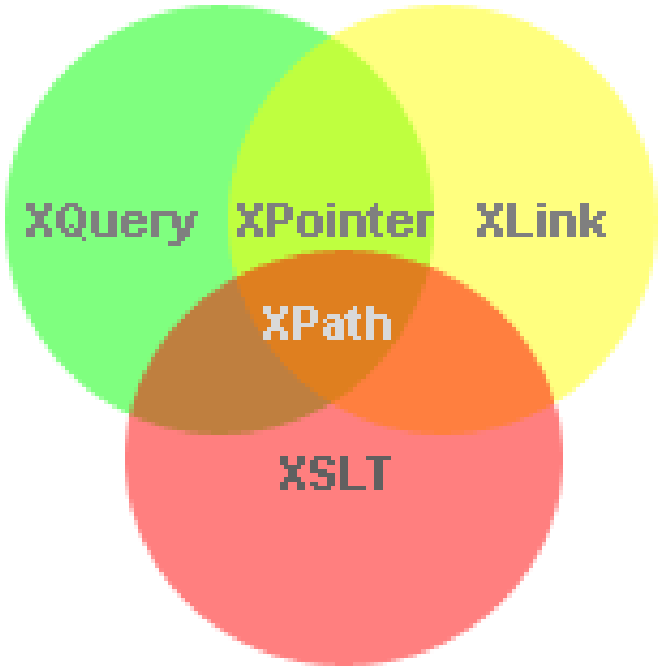
```
FileInputStream file = new FileInputStream(
    new File("kunde.xml"));

JAXBContext ctx = JAXBContext.newInstance(Kunde.class);
SchemaFactory schemaFactory = SchemaFactory.newInstance(
    XMLConstants.W3C_XML_SCHEMA_NS_URI );
Schema schema = schemaFactory.newSchema(
    new File( "kunde.xsd" ));

Unmarshaller u = ctx.createUnmarshaller();
u.setSchema(schema);
Kunde k = (Kunde) u.unmarshal(fileIn);
```

Suchen in XML: Xpath, Xquery, Xpointer, Xlink, …

- XPath
  - = Navigation in XML-Dokumenten
  - = Adressierung von Dokument-Teilen
- XQuery
  - = Anfragesprache für XML-Dokumente
- XLink / XPointer
  - = Verweise in XML Dokumenten
- XSLT
  - = Stylesheet Sprache zur Transformation von XML in andere Formate (z.B. XHTML)

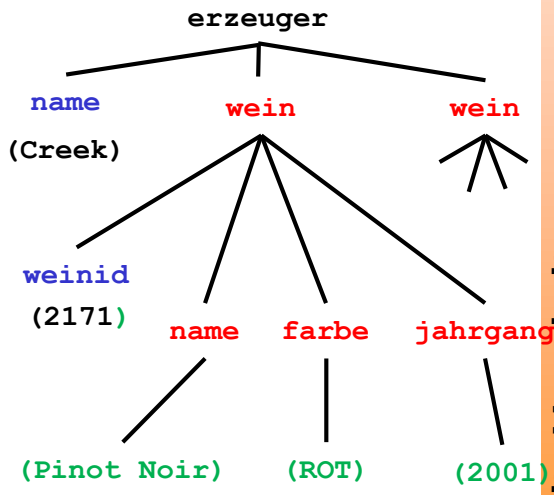


Quelle: W3C

DOM - Baum

- XML Dokument wird in einen ***DOM-Baum*** übersetzt
- DOM = Document Object Model
  - XML-Elemente sind die Knoten (Node)
  - Beziehungen zwischen Elementen sind die Kanten
  - Expliziter Wurzelknoten
  - Spezielle Knotentypen: **Element-**, **Attribut-**, **Text-** und Kommentarknoten
  - Attribut- und Textknoten haben einen (Wert)
- = „Schnittstelle“ zum Zugriff auf XML-Dokumente
- DOM haben Sie ggf. schon in Webtechnologie angewendet
  - Navigation und Manipulation in (X)HTML-Seiten

```
<erzeuger name="Creek">
  <wein weinid="2168">
     <name>Creek Shiraz</name>
     <farbe>ROT</farbe>
     <jahrgang>2003</jahrgang>
  </wein>
  <wein weinid="2171">
     <name>Pinot Noir</name>
     <farbe>ROT</farbe>
     <jahrgang>2001</jahrgang>
  </wein>
</erzeuger>
```



XPath

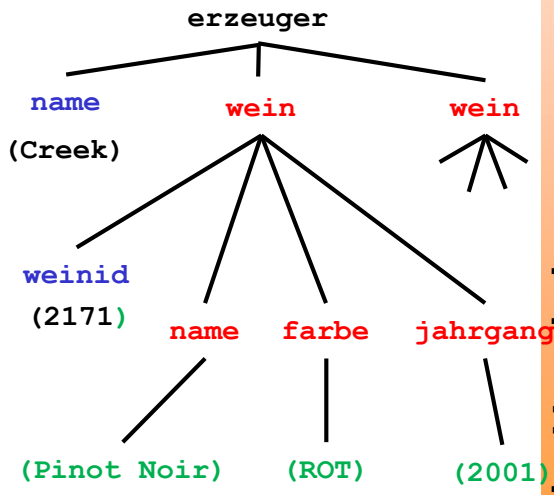
- = Pfad-Ausdruck im DOM-Baum
- Beispiele:
  `/child::erzeuger/child::wein`
  (alle Wein-Knoten)

  `/child::erzeuger/child::wein`
  `/child::name`
  (alle Name-Knoten aller Wein-Knoten)

  `/child::erzeuger/child::wein`
  `/child::name/child::text()`
  (die tatsächlichen Namen der Weine)

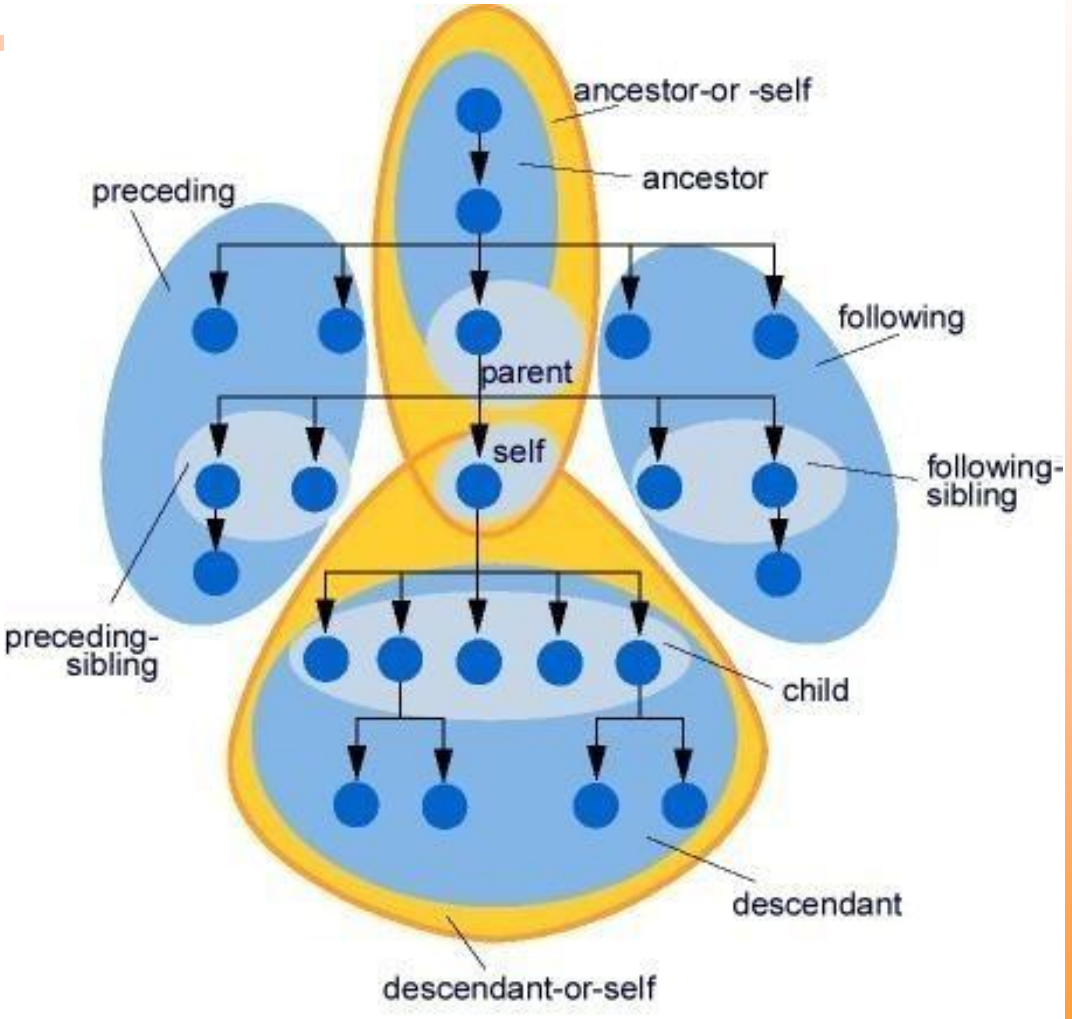
  `/child::erzeuger/child::wein`
  `/attribute::weinid`
  (Wert des Attributs weinid)

```
<erzeuger name="Creek">
  <wein weinid="2168">
     <name>Creek Shiraz</name>
     <farbe>ROT</farbe>
     <jahrgang>2003</jahrgang>
  </wein>
  <wein weinid="2171">
     <name>Pinot Noir</name>
     <farbe>ROT</farbe>
     <jahrgang>2001</jahrgang>
  </wein>
</erzeuger>
```



Pfadausdrücke im DOM-Baum

- **`self`** = aktueller Kontext- oder Referenzknoten
- **`child`** bezeichnet alle direkten Subelemente des Kontextknotens
- **`descendant`** umfasst alle direkt und indirekten Subelemente
- **`descendent-or-self`** kombiniert beides
- **`parent`** steht für den Elternknoten
- **`ancestor`** bezeichnet alle Knoten auf dem Pfad zur Wurzel
- **`preceding`** bezeichnet alle in der Dokumentreihenfolge vorangehenden Knoten
- **`following`** = alle nachfolgenden Knoten
- **`preceding-sibling`** = Geschwisterknoten



Quelle: http://www.tecchannel.de

Aufbau der Pfadausdrücke

- Ausführliche Schreibweise
  - / Wurzelknoten
  - descendant::* alle untergeordneten Knoten
  - child::wein alle direkten Unterknoten vom Typ Wein
  - attribute::weinid Attribut weinid
  - child::name/child::text() Textknoten der Unterknoten vom Typ Name
- Verkürzte Schreibweisen
  - . für self::node()
  - .. für parent::node()
  - // für /descendant-or-self::node()/
  - @ für attribute::
  - * für alle Elementknoten

Einfache einschränkende Prädikate

- Einschränkungen immer in […]
- Einschränkungen auf Werten von Textknoten, Beispiele
  - `//erzeuger[@name='Creek']/wein/name/text()`
  - `//erzeuger/wein[jahrgang<2000]/name/text()`
  - `//erzeuger/wein[jahrgang>=2000 and jahrgang <= 3003]`
- Einschränkungen auf den Positionen der Kind-Konten
  - `//erzeuger/wein[2]/name/text()` Zweiter Wein eines Erzeugers
- Hilfsfunktionen (unter java nicht getestet)
  - `//erzeuger/wein[fn:position() = ( 1, 2, 3)]`

Diskussion XML als Nachrichtenformat

- ***Derzeit Standard in Middleware (WebService, Rest, Messaging)***
  - Neutrales, plattformunabhängiges Format
  - Wichtig: Syntax und Semantikprüfung von XML-Dateien über XML-Schema möglich -> vgl. Schnittstellen Vereinbarung
- Java: Einfach zu verwenden (Annotations), mehr Probleme in anderen Programmiersprachen
- Aufwendig zu Parsen, da sehr viele Sonderfälle möglich
- Selbstbeschreibendes Format
  - Dadurch eher „geschwätzig“
- Kaum geeignet für embedded Devices:
  - Grund: Speicherverbrauch und umfangreichem Parser