



XML Binding APIs

XML Beans & JAXB

• • • Agenda

1. Einleitung

2. XMLBeans

- Einführung
- Quiz und Übung

3. Java Architecture for XML Binding (JAXB)

- Einführung
- Quiz und Übung

4. Reflexion

- XMLBeans vs. JAXB: Gemeinsamkeiten und Unterschiede
- sinnvolle Einsatzgebiete

• • • 1. Einführung

- XML-Binding APIs
 - Ansatz: Generierung von Java-Datenstrukturen aus XML Schemata
 - Ziel: effiziente Verarbeitung von XML-Daten
 - keine generische Low-Level-Verarbeitung von XML-Daten (SAX, DOM)
 - ...sondern: automatisierte Abbildung des durch das Schema definierten Modells in Java-Datenstrukturen
 - Typsicherheit
 - übernehmen Persistenz-Aufgaben (Marshalling, Unmarshalling)
- Einsatzgebiete
 - Verarbeitung von XML-Daten in Standardformaten (international-gültige, branchenspezifische...)



2. XML Beans

• • • 2. XMLBeans: Was ist/sind XMLBeans?

- Inhaltlich

- “XMLBeans is a technology for accessing XML by binding it to Java types.” -- *Apache Foundation*
- Mit XMLBeans ist die Generierung von Java-Datentypen aus einem Schema und ein einfacher, JavaBean-artiger Zugriff auf Daten eines XML-Instanz-Dokuments möglich

- Organisatorisch

- Ein Projekt der Apache Foundation bzw. Teil von deren XML Projekt (daher auch unter Apache Lizenz)
- XMLBeans wurde ursprünglich innerhalb von BEA's Weblogic Workshop Team entwickelt und 2003 an die Apache Foundation übergeben
 - Projektseite: <http://xmlbeans.apache.org>

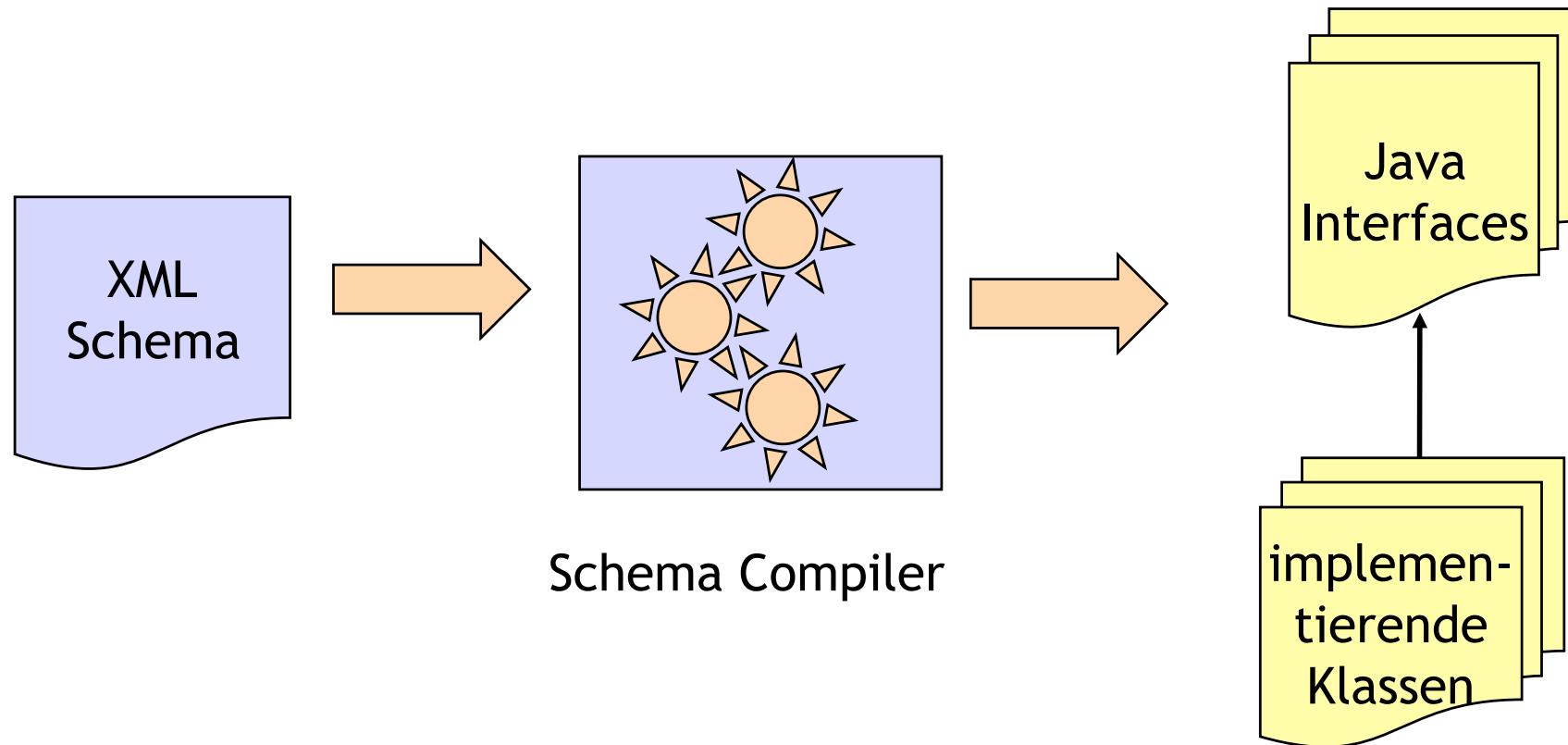
• • • 2. XMLBeans: Features

- Generierung von Java-Datentypen aus einem XML Schema
- unterstützt vollständige XML Schema Spezifikation
- objektbasierte Sicht auf XML-Daten ohne Verlust des Zugriffs auf die originäre, native XML-Struktur
 - Integrität eines XML Instanz Dokumentes bleibt gesichert (z.B. Reihenfolgen von Elementen, Leerzeichen, Tabs), da XML Daten als solche im Speicher gehalten werden
- performanter Zugriff auf XML Daten aus Java heraus

• • • 2. XMLBeans: Zugriffsarten

1. Generierung von Java Datentypen aus einem XML Schema
 - Kompilierung eines Schemas zu Java-Interfaces und implementierenden Klassen
2. Verwendung eines Cursor-Modells, durch das ein XML Infoset traversiert werden kann
 - Zerlegung eines XML-Instanz-Dokumentes in Tokens
 - Durchwandern und Manipulation mittels Cursors
 - ermöglicht Verarbeitung von XML Daten ohne Kenntnis eines Schemas
3. Unterstützung des W3C DOM

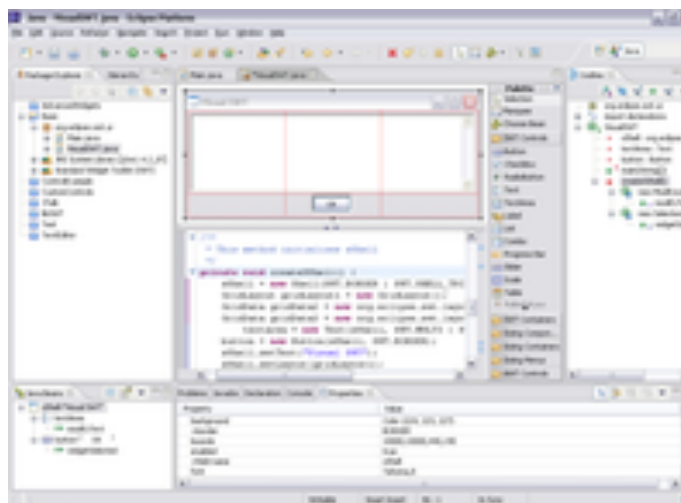
••• 2. XMLBeans: Java-Datentypen aus XML Schema



• • • 2. XMLBeans: Was geschieht beim Kompilieren?

- Aus einem XML Schema werden Java Interfaces und deren Implementierung generiert
- Enthaltene Elemente und Attribute eines Schema-Elements werden zu Attributen einer Java-Klasse
 - Eingebaute Schema Datentypen werden auf einfachen Java -Datentypen sowie XMLBeans-Datentypen gemappt
 - Komplexe Typen werden zu Java Interfaces verarbeitet
- Zugriff auf enthaltene Elemente und Attribute über Methoden im JavaBean-Style, z.B.
 - `getXY()`, `setXY()`, `addXY()`, `removeXY()`, `isXY()`
 - Generierung der Methoden abhängig von der Kardinalität im XML Schema
- Detailliertere Erläuterung der Kompilierung später

- • • 2. XMLBeans: Live-Demo in Eclipse



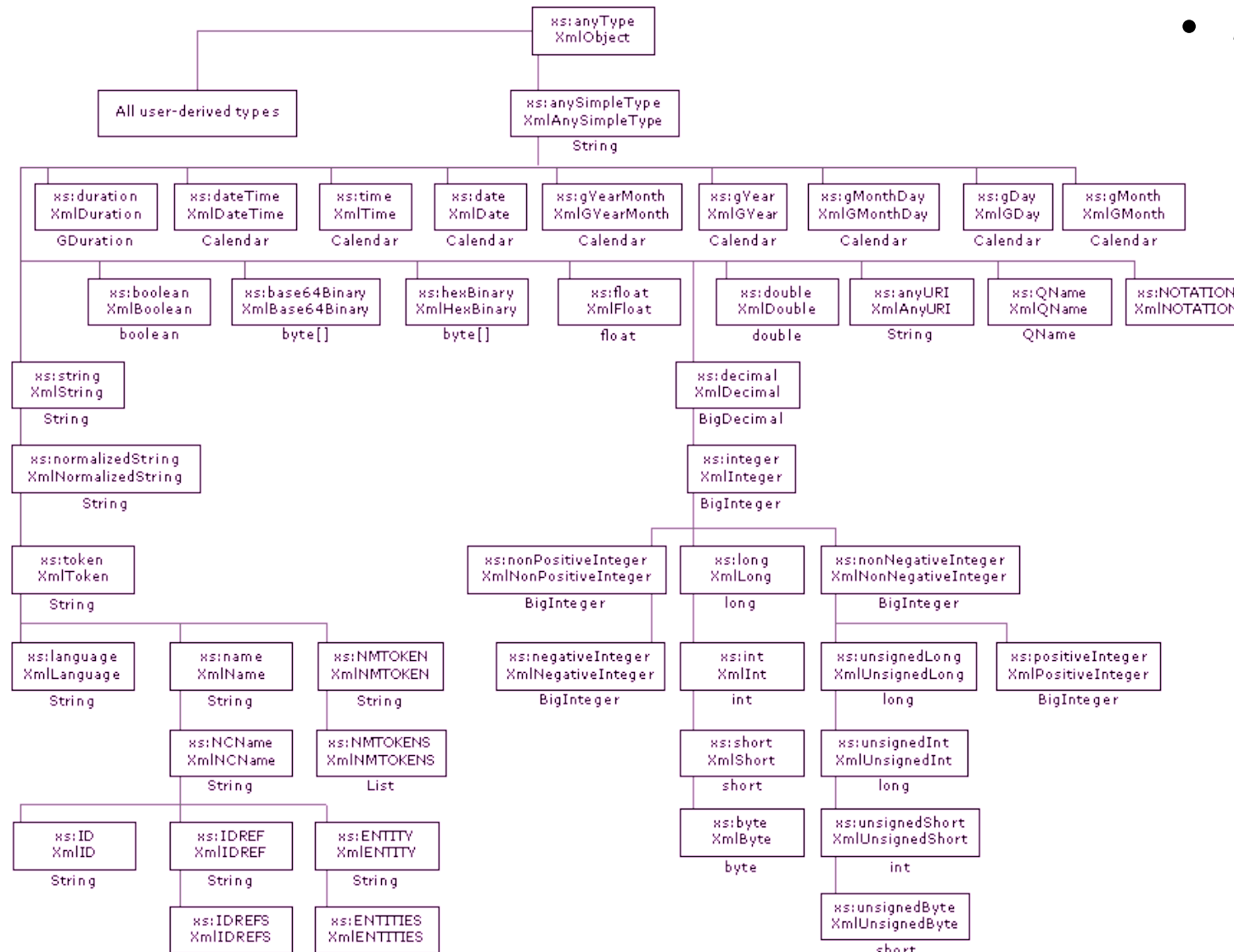
- Beispiel-Schema: Buchbestellungen (bestellung.xsd)
 - aus übersetztem Teil 0 der W3C Schema-Spezifikation
[<http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/#K2.1>]

• • • 2. XMLBeans: Typen-Generierung I

- Resultierendes API besteht generell aus zwei Arten von Daten
 - Eingebaute (built-in) Typen, die die Typen der Schema Spezifikation widerspiegeln
 - Z.B. `XmlString`, `XmlDecimal`, `XmlInt`
 - Zugriff daneben auch über native Java-Datentypen möglich
 - Typen, die aus selbst-abgeleiteten Schema-Typen generiert werden
 - neue Interfaces und deren Implementierung als Klassen
- Alle Typen erweitern `XmlObject`, das diverse Methoden und eine enthaltene Factory-Klasse vererbt

• • • 2. XMLBeans: Typen-Generierung II

- Mapping der 46 Datentypen aus XML Schema :



- Vererbungs-hierarchie in XML Schema durch eingebaute Typen (Xml<Typ>) abgebildet
- „bequemer“ Zugriff über native Java-Typen möglich

• • • 2. XMLBeans: Typen-Generierung III

- Package-Namen:
 - aus Namespace-URI generiert
 - z.B. `http://tempuri.org/xy` → `org.tempuri.xy`)
- Interfaces:
 - benannte Typen werden zu neuen, benannten Interfaces
 - lokal benannte Typen (benannte Typen innerhalb eines benannten Typs) werden zu inneren Interfaces des Typs, innerhalb dessen Typ sie definiert wurden
 - anonyme Schema-Typen werden zu inneren Interfaces der Typen, innerhalb derer sie definiert wurden

• • • 2. XMLBeans: Typen-Generierung IV

- Sonderfall: abgeleitete, simple Typen

= Typen, die aus den 46 Typen von XML Schema gebildet und modifiziert wurden durch...

- a) Einschränkungen
- b) Vereinigungen (Unions)
- c) kommagetrennte Listen
- hier werden die im Schema spezifizierten Modifikationen auch in der Implementierung der generierten Interfaces durchgesetzt
- Durchsetzung der im Schema spezifizierten Kardinalitäten etc.: später beim Thema Validierung

• • • 2. XMLBeans: Generierte Methoden I

- Methoden für Elemente und Attribute, die einzeln vorkommen können
 - Für XML-Schema-Datentypen* generiert werden:
 - `Type getX()`
 - `void setX(Type newValue)`
 - `XmlType getX()`
 - `void xsetX(XmlType newValue)`
 - `XmlType addNewX()`
 - Falls Element leer sein darf (`nillable="true"`)
 - `boolean isNilX()`
 - `void setNilX(boolean nillable)`
 - Falls Element oder Attribut optional
(Element mit `minOccurs="0"`; Attribut mit `use="optional"`)
 - `boolean isSetX()`
 - `void unsetX()`

Type: nativer Java-Typ

XmlType: XMLBeans-Typ

(nur falls XY simplem Schema-Typ entspricht)

*für `xs:any`-Bereiche werden
keine Zugriffsmethoden generiert!

• • • 2. XMLBeans: Generierte Methoden II

- Methoden für Elemente, die mehrfach vorkommen können (maxOccurs > 1 oder ="unbounded")

- `List<Type> getXYList()`
- `Type[] getXYArray()`
`XmlType[] xgetXYArray()`
- **void** `setXYArray(Type[] newValues),`
void `xsetXYArray(XmlType[] newValues)`
- **int** `sizeOfXYArray()`

- `Type` `getXYArray(int index)`
`XmlType` `xgetXYArray(int index)`
- **boolean** `isNilXYArray(int index)`
- `void` `setNilXYArray(int index)`
- **void** `setXYArray(Type newValue, int index)`
void `xsetXYArray(XmlType newValue, int index)`

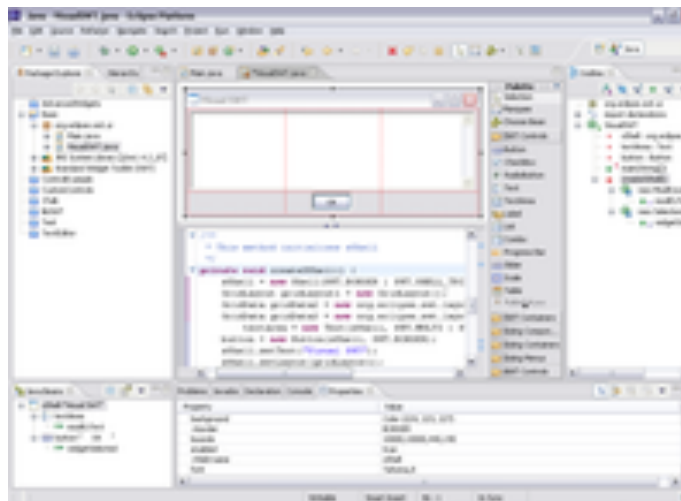
- **void** `addXY(Type newValue)`
- `XmlType` `addNewXY()`
- **void** `removeXY(int index)`
- **void** `insertXY(int index, Type newValue)`
- `XmlType` `insertNewXY(int index)`

Type: nativer Java-Typ
XmlType: XMLBeans-Typ
(nur falls XY simplem
Schema-Typ entspricht)

• • • 2. XMLBeans: XML Object

- Pendant zum Schema-Typ `xs:anyType`
- Ober-Interface aller eingebauten und generierten Typen
- vererbt wichtige Funktionalität
 - Factory-Klasse zur Erstellung neuer Instanzen
 - „leere Instanzen“: `Factory.newInstance()`
 - Instanzen aus Datei, Stream...:
`Factory.parse(File f)`, `Factory.parse (Stream s)`
 - Erstellung von `XmlCursors`: `newCursor()`
 - Rückgabe von DOM-Nodes: `getDomNode()`
 - Ausführung von XPath- und XQuery-Ausdrücken:
`selectPath(String expr)` bzw. `execQuery(String expr)`

- • • 2. XMLBeans: Live-Demo in Eclipse

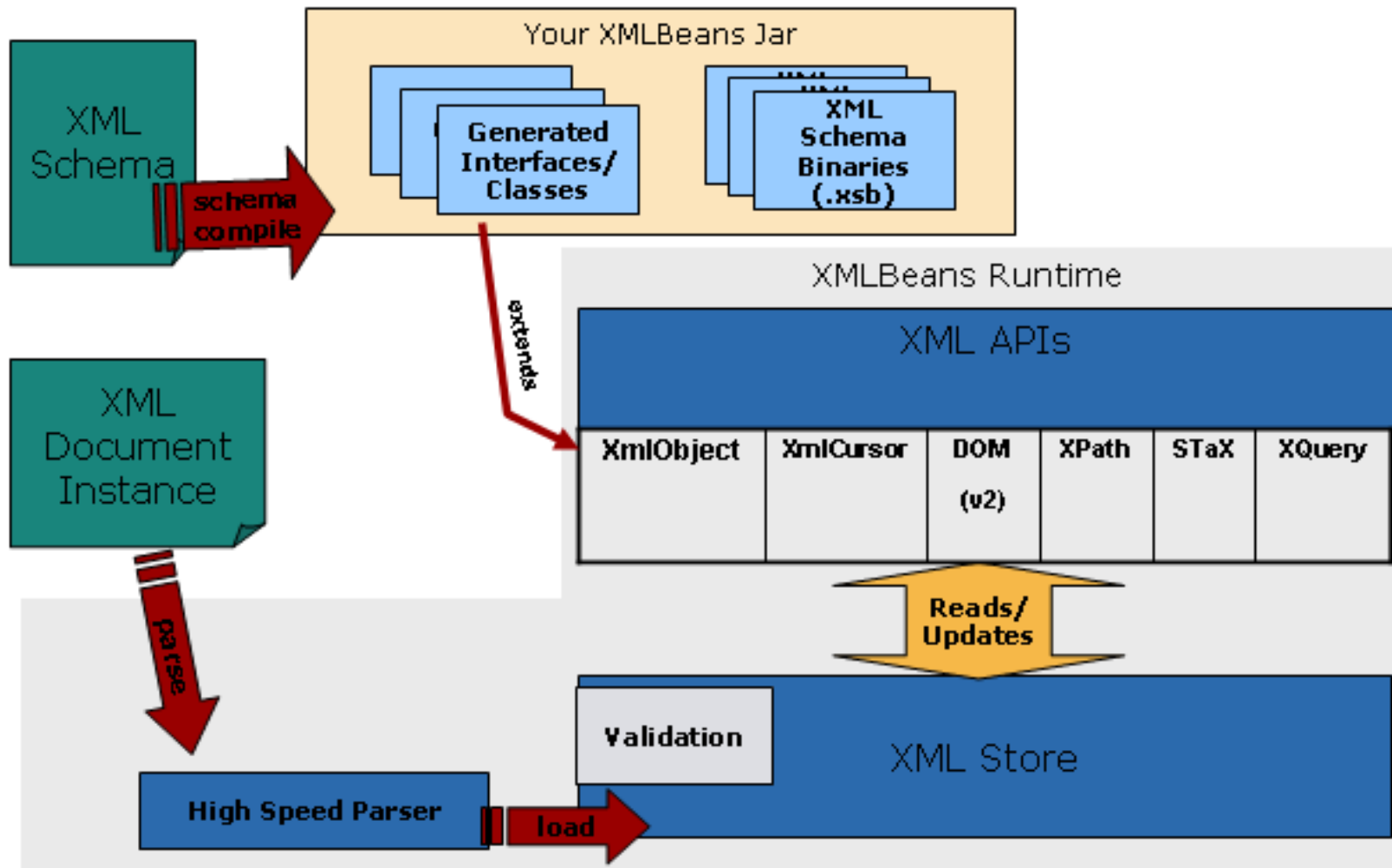


- Vergleich des Mappings der Datentypen

• • • 2. XMLBeans: Instanz-Dokumente an Klassen binden

- Aufruf einer der Methoden der Factory-Klasse:
 - z.B. `newInstance()`, `parse(File f)`
- Folge:
 1. Aus einem Instanz-Dokument wird im Arbeitsspeicher ein Graph von Java-Objekten abgelegt, die Instanzen der aus dem Schema „kompilierten“ Interfaces bzw. deren Implementierung sind.
 2. Zusätzlich wird das gesamte XML Infoset des XML-Instanz-Dokuments zum Durchwandern und zur Manipulation bereitgestellt.

- 2. XMLBeans: Architektur von XMLBeans



[http://www.apache.org/~kkrouse/apachecon/ac2004 XmlBeans_v2.ppt]

• • • 2. XMLBeans: Arbeiten mit XMLBeans

1. Java Datentypen aus Schema generieren

- **Commandline:** `scomp -out <zieldatei> <schema>`
- In Java mit der Klasse
`org.apache.xmlbeans.impl.tool.SchemaCompiler`
- mittels Ant-Task

2. Instanziierung der Java-Repräsentation eines XML-Dokuments mittels einer Factory-Methode

- z.B. `XYDocument.Factory.parse(File f)`
- Auslesen / Manipulation des XML-Dokuments mittels generierter Methoden (`get()`, `set()`, `add()` usw.)

• • • 2. XMLBeans: XmlCursors

- Zerlegung eines Instanz-Dokuments in sog. Token
 - für Element-Beginn/-Ende, Attribute, Kommentare usw.
- Navigiert werden kann mittels einer Instanz von `XmlCursor`
 - Instanziierung: Aufruf der Methode `newCursor()` auf einem beliebigen Objekt, das von `XmlObject` ableitet (auch auf Document-Objekten)
 - N.B.:Cursor gelten Dokument-weit, egal, wo (=in welcher Schachtelungstiefe) sie erstellt werden
- Cursors ermöglichen...
 - Durchwandern eines Instanz-Dokuments auf Basis der Tokens
 - Manipulation der Instanz-Dokuments (Einfügen von Elementen, Attributen, Element-Inhalten usw.)
 - Ausführung von XQuery und XPath-Ausdrücke

... 2. XMLBeans: XmlCursors - Token-Typen

9 Token Typen

- STARTDOC
- ENDDOC
- START
- END
- TEXT
- ATTR
- NAMESPACE
- COMMENT
- PROCINST

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!-- Order summary -->  
<?xml-stylesheet type="text/xsl" href="order.xslt"?>  
<WidgetOrder batchId="3DJ" xmlns="org.shipping">  
  <Order id="12">  
    <item><name>Some Widget</name></item>  
  </Order>  
</WidgetOrder>
```

Verarbeitungsanweisung

Abfrage des Token-Typs mit folgenden Methoden auf

einer Cursor-Instanz: `currentTokenType()` oder `is<Token-Typ>()`

adaptiert von: http://www.apache.org/~kkrouse/apachecon/ac2004_XmlBeans_v2.ppt

• • • 2. XMLBeans: XmlCursors - Navigation

- Typische Methoden zum Navigieren

- `toNextToken()`, `toFirstChild()`,
`toChild(int index)`, `toNextSibling()`
- **ACHTUNG:** auch wenn die Methoden-Namen nach DOM-Verarbeitung klingen, liegt hier ein etwas anderes Datenmodell zugrunde!
- die jeweilige Methode versucht nur, ein der Anforderung (z.B. Kind, Geschwister) entsprechendes Token zu finden
 - falls gefunden, wird Cursor vor dieses bewegt und die Methode gibt `true` zurück
 - falls nicht gefunden, gibt Methode `false` zurück

• • • 2. XMLBeans: XmlCursors - Manipulation

- Methoden zur Manipulation von Instanz-Dokumenten
 - Einfügen von Elementen:
 - `beginElement(String elementtyp, String uri)`
 - Cursor anschließend zwischen Start- und Ende-Tag
 - Hinzufügen von Attributen:
 - `insertAttribute(String attributname, String uri)`
`insertAttributeWithValue(String attributname, String wert)...`
 - Hinzufügen von Text-Inhalt:
 - `insertChars(String inhalt)`
 - Cursor steht anschließend hinter eingefügtem Inhalt

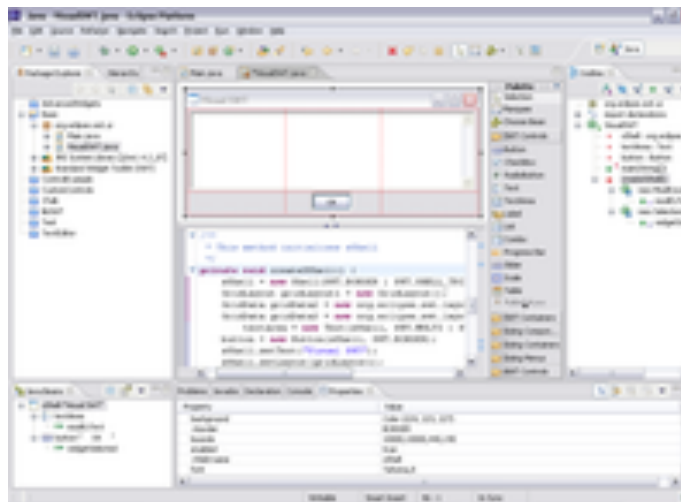
• • • 2. XMLBeans: XmlCursors - Reflektion I

- Vorteile:
 - ein Instanz-Dokument kann ohne Kenntnis eines Schemas interpretiert und manipuliert werden
 - geringerer Ressourcen-Verbrauch als bei Verarbeitung mittels DOM
- Nachteile:
 - man interpretiert ein Instanz-Dokument auf sehr niedriger Ebene (prinzipiell lediglich String-Zerlegung und Zuordnung von Token -Typen)
 - Validierung findet bei Manipulation des Instanz-Dokuments NIE statt (auch nicht falls Schema vorhanden)

• • • 2. XMLBeans: XmlCursors - Reflektion II

- Mögliche Anwendungsgebiete:
 - Schema-unabhängige Verarbeitung
 - ein eigenes Schema befindet sich noch in der Entwicklung
 - Verarbeitung Resultate von XPath- und XQuery-Abfragen
 - Handling von Bereichen, die im Schema mit `xs:any` definiert sind
 - diese Bereiche können beliebige, wohgeformte XML-Instanz-Dokumente enthalten
 - es werden dafür keine Zugriffsmethoden (Getter/Setter etc.) generiert

• • • 2. XMLBeans: Live-Demo in Eclipse



- kurze Vorführung der XMLCursor
- Lesen und modifizieren

• • • 2. XMLBeans: XQuery und XPath-Ausdrücke ausführen I

- XmlBeans-API bietet 2 Methoden und 2 Klassen, auf deren Instanzen sie ausgeführt werden können:
 1. `XmlObject[] XmlObject.selectPath()`
 - gibt Array von `XmlObject`- bzw. Unterklassen-Instanzen zurück
 2. `void XmlCursor.selectPath()`
 - speichert eine Liste von Selektionen des Instanz-Dokuments in einem Attribut der `XmlCursor`-Instanz
 - Selektionen können durchlaufen und abgefragt werden mit Methoden wie `toNextSelection()`, `toSelection(int index)`
 - N.B.: nach Verarbeitung der Selektionen die Methode `clearSelections()` aufrufen (sonst „trackt“ Cursor die Selektionen permanent live im Instanz-Dokument)
- die `selectPath()`-Methoden arbeiten generell mit *Referenzen* auf das aktuelle Instanz-Dokument

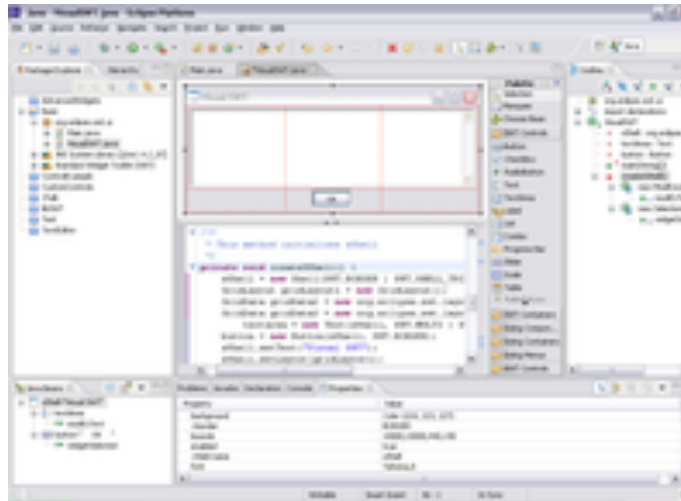
• • • 2. XMLBeans: XQuery und XPath-Ausdrücke ausführen II

- XmlBeans-API bietet 2 Methoden und 2 Klassen, auf deren Instanzen sie ausgeführt werden können:
 1. `XmlObject[] XmlObject.execQuery()`
 - gibt ein Array von Instanzen des Typs `XmlObject` bzw. einer Unterklasse zurück
 2. `XmlCursor XmlCursor.execQuery()`
 - gibt eine Instanz von `XmlCursor` zurück, die am Beginn eines neu erstellten Instanz-Dokumentes steht, das die Abfrage-Ergebnisse enthält (Navigation wie üblich mit `XmlCursors`)
- Resultate der `execQuery()`-Methoden sind aus dem Instanz-Dokument *kopierte* XML-Fragmente (und keine Referenzen)
- XPath- und XQuery-Abfragen setzen APIs wie Saxon im Classpath voraus

• • • 2. XMLBeans: Validierung

- Wann wird validiert?
 - bei Generierung des Schemas zu Java-Datentypen: *immer*
 - bei der Verarbeitung von XML Instanz Dokumenten:
 - implizit: *nie!*
 - explizit möglich:
 - „on request“:
Aufruf der Methode `validate()`, die einen Boolean zurückgibt
 - wird von `XmlObject` vererbt
 - echte Validierung findet statt
 - „on the fly“:
beim Erzeugen einer Dokument-Instanz im Speicher mittels Factory-Methode
Parameter vom Typ `XmlOptions` übergeben
 - Methodenaufruf `setValidateOnSet()` auf `XmlOptions`-Instanz aktiviert
Validierung on the fly
 - aber: keine echte Validierung, eher „ein Debugging-Tool“ (Apache Foundation)

- • • 2. XMLBeans: Live-Demo in Eclipse



- Kurzbeispiel: Validierung

• • • 2. XMLBeans: Tools-Übersicht

| Aufruf | Name | Funktion |
|------------------|-------------------------------|--|
| dumpxsb | XSB File Dumper | Stellt die Inhalte einer xsb-Dateien in menschenlesbarer Form dar |
| inst2xsd | Instance to schema tool | Generiert XML Schema aus XML Instanz Dateien |
| scomp | Schema compiler | Kompiliert ein Schema zu XMLBeans-Class-Files und Metadaten |
| scopy | Schema copier | Kopiert ein Schema aus einer angegebenen URL in eine angegebene Datei |
| sdownload | Schema downloader | Verwaltet einen Datei namens xsdownload.xml, die als Index aus dem Web heruntergeladener und lokal gecachter XSD-Files fungiert. |
| sfactor | Schema factoring tool | Führt refactoring durch, indem es in einem Set von Schemata vorkommende redundante Definitionen herausfiltert und durch Imports ersetzt. |
| svalidate | Streaming Instance validator | Validiert eine Schema-Definition und Instanzen innerhalb eines Schemas |
| validate | Instance validator | Validiert eine Instanz gegen ein Schema |
| xpretty | XML pretty printer | Gibt spezifiziertes XML „hübsch“ auf der Konsole aus |
| xsd2inst | Schema to instance tool | Gibt eine XML Instanz vom globalen Element aus - unter Verwendung eines übergebenen Schemas |
| xsdtree | Schema type hierarchy printer | Gibt eine Vererbungshierarchie der in einem Schema definierten Typen aus |
| xmlbean ant task | | Ant Task, der ein set von XSD- und/oder WSDL-Dateien zu XMLBean Typen kompiliert (Ant-Pendant zu scomp) |

- Aufruf via Commandline oder aus programmatisch aus Java heraus

• • • 2. XMLBeans: weitere Funktionalitäten

- Typen-Generierung anpassen
 - Kompilier-Vorgang kann durch eine Konfigurationsdatei gesteuert werden
 - z.B. lassen sich Package-Namen, Prä- und Suffixe der generierten Interfaces und implementierenden Klassen definieren
- Reflektion des Schemas
 - „Schema Type System API“ ermöglicht Reflection der generierten Typen und des zugrundeliegenden Schemas
 - Einstieg: Interface `org.apache.xmlbeans.SchemaTypeSystem`
- Annotationen mit Bookmarks
 - Lesezeichen (=selbst implementierte Klassen mit beliebigen Attributen) lassen sich mit einem `XmlCursor` an Positionen innerhalb eines Instanz-Dokumentes heften

... 2. XMLBeans: Literatur

- Quellen

- XMLBeans Documentation

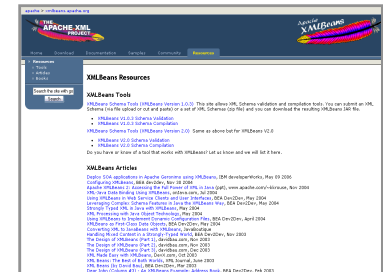
- Online: <http://xmlbeans.apache.org/docs/2.0.0/guide/>
- im XMLBeans-Paket enthalten unter ...\\docs\\guide

- XMLBeans API

- Online: <http://xmlbeans.apache.org/docs/2.2.0/reference/index.html>
- im XMLBeans-Paket enthalten unter ...\\docs\\reference

- Weiterführende Quellen, Tutorials, Interna etc.

- Online: <http://xmlbeans.apache.org/resources/index.html>



• • • 2. XMLBeans

Soweit so gut ... Fragen?



Quiztime



3. JAXB

• • • 3. JAXB: Was ist JAXB?

- Inhaltlich:
 - Java Architecture for XML-Binding
 - „The Java Architecture for XML Binding (JAXB) provides a fast and convenient way to bind between XML schemas and Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications.“ (Sun, Java Web Services Tutorial)
- Organisatorisch
 - wurde im Rahmen des Java Community Processes (JCP) entworfen
 - JAXB 1.0 als JSR (Java Specification Request) 31
 - JAXB 2.0 als JSR 222
 - Referenzimplementierung von SUN seit JSE 1.6 im JDK enthalten
 - Ursprung: Java Web Services (Referenzimplementierung in Glassfish / SUN Application Server)

• • • 3. JAXB: Features:

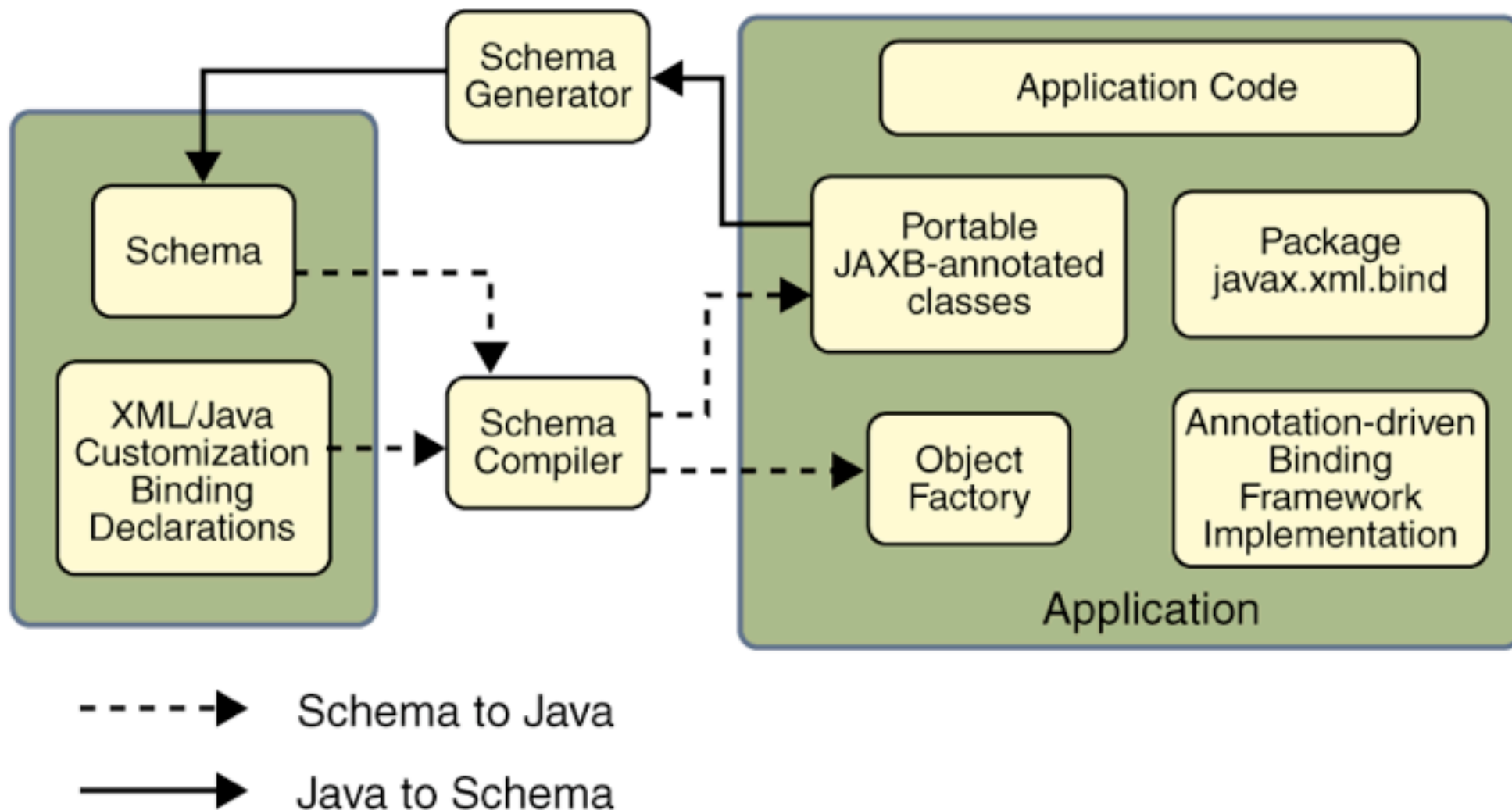
- Generierung von Java-Datentypen aus einem XML Schema
- unterstützt vollständige XML Schema Spezifikation
- objektbasierte Sicht auf XML-Daten
- bequemer & performanter Zugriff auf XML Daten aus Java heraus
 - Diverse XML-parsing APIs verwendbar: DOM, StAX, SAX

• • • 3. JAXB: Aufbau

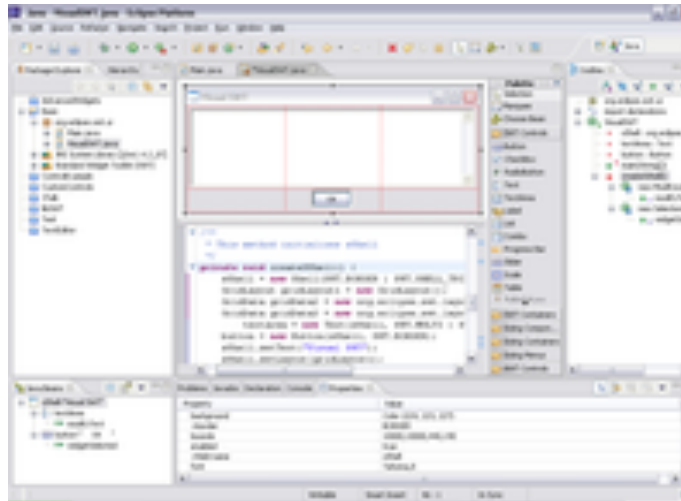
JAXB besteht aus:

- schema compiler:
 - generiert aus einem XML Schema eine Menge abgeleiteter Java Datenstrukturen
 - diese Abbildung kann in einer XML-basierten Binding-Sprache angepasst werden
 - Datentypen-Mapping, Package, ...
 - Im Schema selbst oder getrennte Datei
- schema generator:
 - bildet eine Menge vorhandener Java-Datenstrukturen auf ein abgeleitetes XML Schema ab
 - dieses Mapping wird durch Java-Annotationen beschrieben
- binding runtime framework:
 - ermöglicht lesende (unmarshalling) und schreibende (marshalling) Zugriffe auf XML-Instanzdokumente sowie Validierung

... 3. JAXB: Aufbau



••• 3. JAXB: Live-Demo



Schema Compile mit xjc

• • • 3. JAXB: Schema to JAVA

Packages und enthaltene Klassen

- Java Klassen: Name abgeleitet aus Name des XML-Elements/Typs oder aus Binding-Deklaration

`<xs:type name="BestellungTyp" />` wird zu `BestellungTyp.java`

- eine `ObjectFactory`-Klasse, die Instanzen der an das XML-Schema gebundenen Java-Klassen zurückliefern kann

```
ObjectFactory objFactory = new ObjectFactory();
```

```
BestellungTyp bt= objFactory.createBestellungTyp()
```

- Ausnahme: Für Top-Level-Elemente (z.B. `Bestellung`) werden keine Klassen erzeugt, sie sind jedoch instanziiierbar über `create`-Methoden der `Object-Factory`

```
JAXBElement<BestellungTyp> b = objFactory.createBestellung(BestellungTyp bt);
```

• • • 3. JAXB: Schema to JAVA

Mapping eigener Elemente und Typen

- Pro benanntem XML-Schema Typ eine Klasse
- Innere anonyme Schema-Typen werden zu inneren Klassen der Typen, innerhalb derer sie definiert wurden (Name der inneren Klasse == Name des Elementes, das vom anonymen Typ ist)

```
<xsd:complexType name="WarenTyp">  
  <xsd:sequence>  
    <xsd:element name="Buch" minOccurs="0" maxOccurs="unbounded">  
      <xsd:complexType>  
        ...
```

```
public class WarenTyp{  
  ...  
  public static class Buch{  
    ...  
  }  
}
```

••• 3. JAXB: Mapping von Schema-Typen nach Java

Standard-Mappings

- alle i.F. beschriebenen Standard-Mappings können durch eine Binding-Deklaration überschrieben/verändert werden

Simple Schema-Typen

- typischerweise als Properties einer Java-Klasse abgebildet

Typen:

- Basis-Typen aus Java-Packages
- Collections

• • • 3. JAXB: Mapping von Schema-Typen nach Java

| XML Schema Type | Java Data Type |
|-----------------|---------------------------|
| xsd:string | java.lang.String |
| xsd:integer | java.math.BigInteger |
| xsd:int | int |
| xsd.long | long |
| xsd:short | short |
| xsd:decimal | java.math.BigDecimal |
| xsd:float | float |
| xsd:double | double |
| xsd:boolean | boolean |
| xsd:byte | byte |
| xsd:QName | javax.xml.namespace.QName |

• • • 3. JAXB: Mapping von Schema-Typen nach Java

| XML Schema Type | Java Data Type |
|-------------------|---|
| xsd:dateTime | javax.xml.datatype.XMLGregorianCalendar |
| xsd:base64Binary | byte[] |
| xsd:hexBinary | byte[] |
| xsd:unsignedInt | long |
| xsd:unsignedShort | int |
| xsd:unsignedByte | short |
| xsd:time | javax.xml.datatype.XMLGregorianCalendar |
| xsd:date | javax.xml.datatype.XMLGregorianCalendar |
| xsd:g* | javax.xml.datatype.XMLGregorianCalendar |
| xsd:anySimpleType | java.lang.Object |

• • • 3. JAXB: Mapping von Schema-Typen nach Java

| XML Schema Type | Java Data Type |
|-----------------|-----------------------------|
| xsd:duration | javax.xml.datatype.Duration |
| xsd:NOTATION | javax.xml.namespace.QName |

falls kein sinnvolles Mapping von XML Schema Datentypen auf Java-Datentypen gefunden werden kann, wird die Klasse `JAXBElement` verwendet, die Methoden zum Setzen des Namens und Werts des Objektes bereitstellt

• • • 3. JAXB: Schema nach JAVA - User defined bindings

- Inline Bindungskonfiguration

- Namespace und Version im Root-Element

```
<xsd:schema jaxb:version="2.0" xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

- Bindungskonfiguration in Schema-Element

```
<xsd:annotation>  
  <xsd:appinfo>  
    <jaxb:... />  
  </xsd:appinfo>  
</xsd:annotation>
```

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Beispiel für Inline-Bindungskonfiguration

```
<xsd:schema jaxb:version="2.0"
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ...
  <xsd:complexType name="WarenTyp">
    <xsd:annotation>
      <xsd:appinfo>
        <jaxb:class name="Waren" />
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:sequence>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Externe Bindungskonfiguration

- Namespace und Version im root-Element

```
<jaxb:bindings schemaLocation="best.xsd"  
  jaxb:version="2.0"  
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

- Bindungskonfiguration für Schema-Element

```
<jaxb:bindings node="XPath-Ausdruck">  
  <jaxb:... />  
</jaxb:bindings>
```

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Beispiel für externe Bindungskonfiguration

```
<jaxb:bindings schemaLocation="best.xsd"jaxb:version="2.0"  
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  ...  
    <jaxb:bindings node="/xsd:complexType[@name='WarenTyp']">  
      <jaxb:class name="Waren" />  
    </jaxb:bindings>  
  ...
```

Aufruf mittels `xjc -d <Dir> schema.xsd -b <ext.B.File>`

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Globale Bindungskonfiguration (<jaxb:globalBindings ...>)

- Bereiche: globale Einstellungen für das Schema und alle referenzierten Schemas
- Verwendung: ermöglicht z.B. Aufzählungen anzupassen, isSet-Methoden zu generieren
- Einstellungen:
 - collectionType= “...” (z.B. java.util.Vector)
 - generatelsSetMethod= “true“/“false“

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Schema Bindungskonfiguration (<jaxb:schemaBindings>)

- Bereiche: Schema-Ebene - alle Elemente in diesem Schema
- Verwendung: ermöglicht z.B. die Anpassung von Package- oder Typ-Namen
- Einstellung:
 - <jaxb:package name="neuer.Name" />
 - <jaxb:nameXmlTransform>
 <typeName suffix="..." prefix="..." />
</jaxb:nameXmlTransform>

• • • 3. JAXB: Schema nach JAVA - User defined bindings

Bindings für Typen und Elemente (<jaxb:class>)

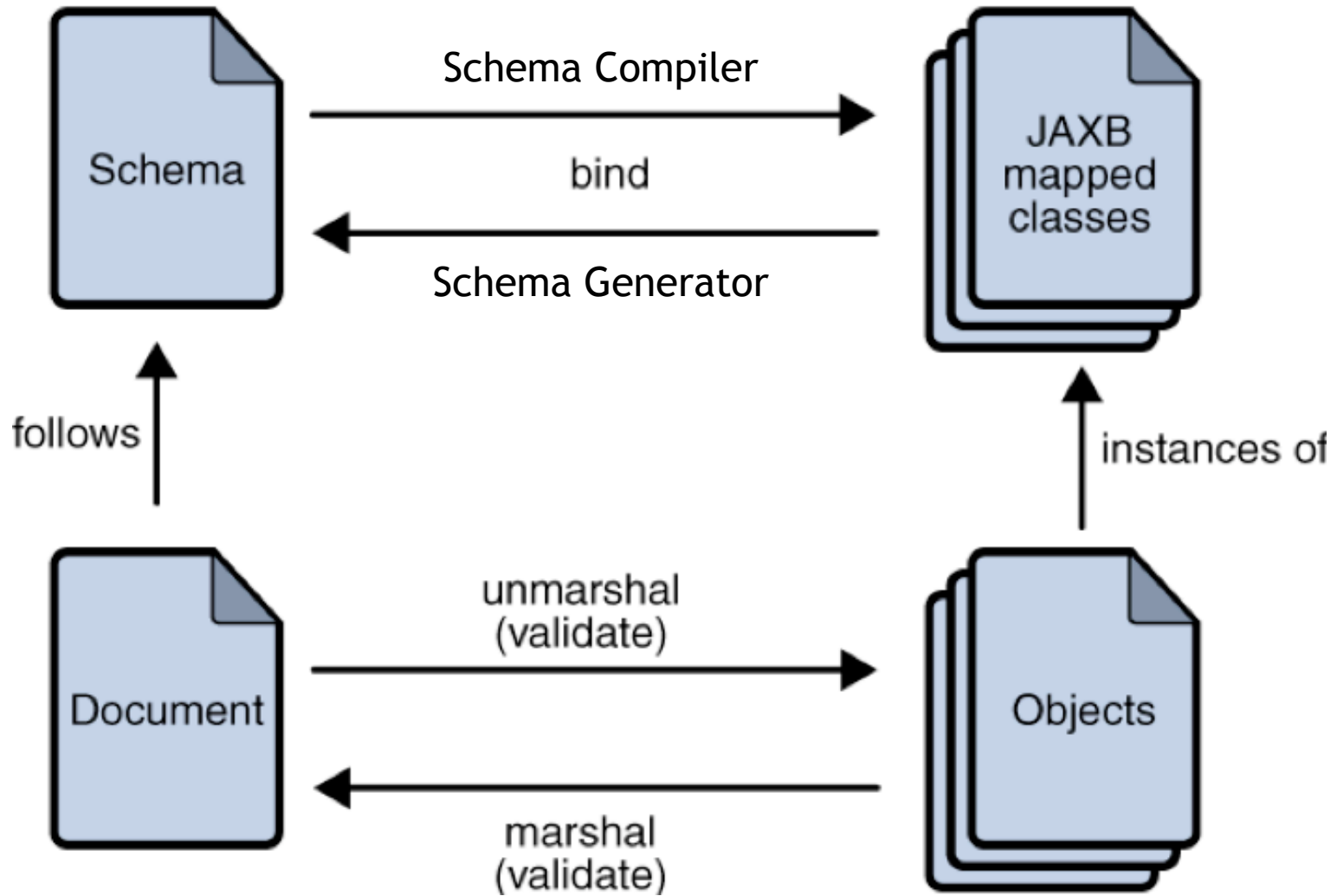
- Bereiche: Typen und Elemente
- Verwendung: ermöglicht z.B. das Anpassen des Namens der generierten Klasse
- Einstellungen:
 - name="NeuerName"

• • • 3. JAXB: Schema nach JAVA - User defined bindings

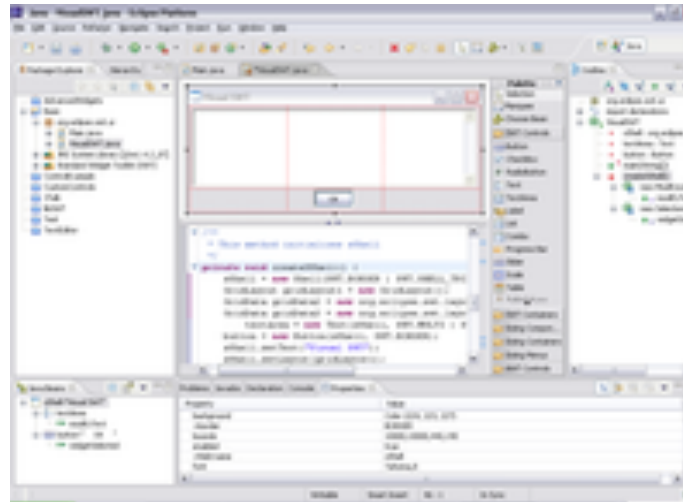
Property Bindings (<jaxb:property>)

- Bereiche: Java-Bean Eigenschaften
- Verwendung: ermöglicht, z.B. den Namen anzupassen, JAXBElement zu erzeugen
- Eigenschaften
 - name= "neuerName"
 - generateElementProperty= "true"/"false"

... 3. JAXB: Binding Prozess



••• 3. JAXB: Live-Demo



Mashalling / Unmarshalling

• • • 3. JAXB: Marhallung

Unmarshall

```
JAXBContext context = JAXBContext.newInstance("packagename");
```

```
Unmarshaller unmarshaller = context.createUnmarshaller();
```

```
JAXBElement<typ> element = (JAXBElement<typ>)unmarshaller.unmarshall(File f);
```

```
Typ t = element.getValue();
```

• • • 3. JAXB: Unmarshalling

Marshall

```
JAXBContext context = JAXBContext.newInstance("packagename");
```

```
Marshaller marshaller = context.createMarshaller();
```

```
Marshaller.marshall(JAXBElement<typ> element, File f);
```

• • • 3. JAXB: Java to Schema

Schemagen-Tool

- Command-Line-Tool
`{JAVA_HOME}\bin\schemagen.exe / .sh`
 - Aufruf
`schemagen src/`
- Programmatisch
`context.generateSchema(FileOutputResolver format);`

• • • 3. JAXB: Java to Schema

| Java Data Type | XML Schema Type |
|---|------------------|
| java.lang.String | xs:string |
| java.math.BigInteger | xs:integer |
| java.math.BigDecimal | xs:decimal |
| java.util.Calendar | xs:dateTime |
| java.util.Date | xs:dateTime |
| javax.xml.namespace.QName | xs:QName |
| java.net.URI | xs:string |
| javax.xml.datatype.XMLGregorianCalendar | xs:anySimpleType |

••• 3. JAXB: Java to Schema

| Java Class | XML Data Type |
|---|------------------------------|
| <code>javax.xml.datatype.Duration</code> | <code>xs:duration</code> |
| <code>java.lang.Object</code> | <code>xs:anyType</code> |
| <code>java.awt.Image</code> | <code>xs:base64Binary</code> |
| <code>javax.activation.DataHandler</code> | <code>xs:base64Binary</code> |
| <code>javax.xml.transform.Source</code> | <code>xs:base64Binary</code> |
| <code>java.util.UUID</code> | <code>xs:string</code> |

• • • 3. JAXB: Java to Schema

Mapping eigener Klassen

- Bindung anpassbar über JAXB-Annotations
- Annotierbare Java-Programm-Elemente:
 - Felder, Properties, Klassen, Enums, Packages
- Dokumentation:
<http://java.sun.com/javase/6/docs/api/index.html>
package javax.xml.bind.annotation

• • • 3. JAXB: Java to Schema

- @XMLElement
 - Java-Elemente: Felder, Properties
 - Verwendung: bindet ein Feld oder ein Property an eine XML-Element-Deklaration
 - Eigenschaften:
 - name: String
 - namespace: String
 - required: boolean(false → minOccurs=0)
 - type: Class
 - defaultValue: String(→ default=“...“)

• • • 3. JAXB: Java to Schema

- @XMLType
 - Java-Elemente: Klassen, Enums
 - Verwendung:
 - bindet eine Klasse oder Enum an einen XML-Typ
 - Reihenfolge der Unterelemente bestimmbar
 - Eigenschaften
 - name: String
 - namespace: String
 - propOrder: String[] (→<xsd:sequence>; leeres Array →<xsd:any>)

• • • 3. JAXB: Java to Schema

- `@XMLAccessType`
 - Java-Elemente: Klassen, Packages
 - Verwendung: bestimmt, welche Felder und Properties(JavaBean-konforme Felder) an XML-Deklarationen gebunden werden
 - Eigenschaften:
 - `XmlAccessType.FIELD`: bindet alle Java- Elemente ungeachtet ihrer Sichtbarkeit
 - `XmlAccessType.NONE`: bindet keines der Java-Elemente
 - `XmlAccessType.PROPERTY`: bindet alle Properties
 - `XmlAccessType.PUBLIC_MEMBER`: Default, bindet alle public Felder und Properties

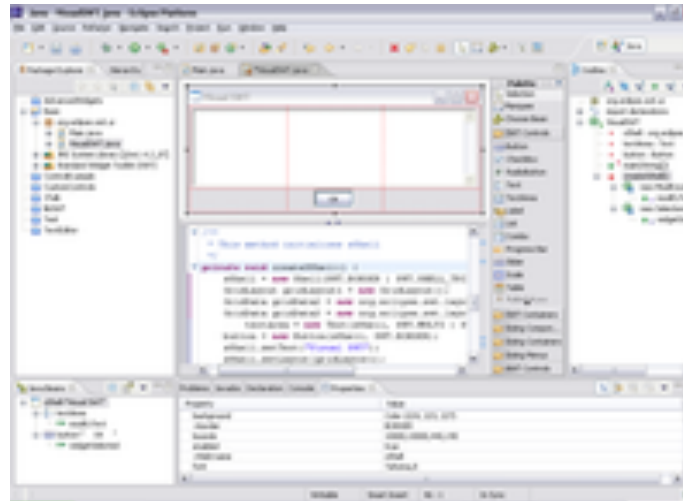
• • • 3. JAXB: Java to Schema

- @XMLTransient
 - Java-Elemente: Felder, Properties
 - Verwendung: schließt ein Feld oder eine Property von der Bindung aus (z. B. um Namenskonflikte aufzulösen)

• • • 3. JAXB: Java to Schema

- @XMLAttribute
 - Java-Elemente: Felder, Properties
 - Verwendung: bindet ein Feld oder eine Property an eine XML-Attribut-Deklaration
 - Eigenschaften
 - name: String
 - namespace: String
 - required: boolean(true → use="required")

• • • 3. JAXB: Live-Demo



Schema-Generierung aus Java Datenstrukturen

••• 3. JAXB: 1.0 vs. 2.0

| JAXB 1.0 | JAXB 2.0 |
|---|--|
| Schema nicht vollständig unterstützt | Schema vollständig unterstützt |
| - | JAXB Annotations |
| Schema to Java - | Schema to Java Java to Schema |
| Schema to Java generiert Interfaces und Implementierungen dieser | Schema to Java generiert einfache Java-Bean Klassen mit JavaDoc und JAXB-Annotations |
| Objekte der generierten Interfaces werden mit Hilfe der Klasse ObjectFactory erzeugt | Objekte können direkt von den generierten Klassen instanziiert werden |
| Validierung wird beim Unmarshalling unterstützt sowie Aufruf von „validate()“ möglich | Validierung Marshalling & Unmarshalling |

• • • 3. JAXB: Quellen

- Sun Microsystems Inc.: The Java Web Services Tutorial
<http://java.sun.com/webservices/docs/2.0/tutorial>
- Java Architecture for XML Binding (JAXB)
<http://java.sun.com/developer/technicalArticles/WebServices/jaxb/>
- Unofficial JAXB Guide
<https://jaxb.dev.java.net/guide/>
- JAVA SE 1.6 API
<http://java.sun.com/javase/6/docs/api>
packages: javax.xml.bind.*



4. Reflexion

• • • 4. Reflexion: XMLBeans vs. JAXB

Gemeinsamkeiten

- Erzeugung von Klassen (und Interfaces) aus XML-Schema

Unterschiede

- JAXB erlaubt Schemagenerierung aus Java Klassen
- JAXB verwendet (fast) ausschließlich Java-Datentypen

• • • 4. Reflexion: Sinnvolle Einsatzgebiete

- Typisiertes Datenmodell durch xml-Schema spezifiziert (z.B. als Industriestandard oder Branchenstandard)
- Als Persistenzschicht
- ...



Vielen Dank für die Aufmerksamkeit!