Softwareentwicklung 4

Namespaces und Validität

Dominik Dolezal

Höhere Lehranstalt für Informationstechnologie

20. Februar 2017

Inhalt

tgm
Die Schule der Technik

tgm
Die Schule der Technik

Beispiel

f. name	name	address	city	state
James	Smith	6649 N Blue Gum St	New Orleans	LA
Jenna	Darakjy	4 B Blue Ridge Blvd	Brighton	MI
Art	Venere	8 W Cerritos Ave 54	Bridgeport	NJ
Lenna	Paprocki	639 Main St	Anchorage	AK

```
f. name;name;address;city;state
James;Smith;6649 N Blue Gum St;New Orleans;LA
Jenna;Darakjy;4 B Blue Ridge Blvd;Brighton;MI
Art;Venere;8 W Cerritos Ave 54;Bridgeport;NJ
Lenna;Paprocki;639 Main St;Anchorage;AK
```

CSV

tgm
Die Schule der Technik

f. name	name	address	city	state
James	Smith	6649 N Blue Gum St	New Orleans	LA
Jenna	Darakjy	4 B Blue Ridge Blvd	Brighton	MI

- CSV-Dateien sind simple Textdateien mit speziellen Regeln
- Es gibt verschiedene Dialekte
- Eignen sich gut für tabellarische Informationen
- Eignen sich schlecht für
  - hierarchische Beziehungen
  - referentielle Beziehungen
  - Unterstützung von Layout
  - benutzerdefinierte Regeln (z.B. nur Zahlen, E-Mail-Adressen)

JSON

tgm
Die Schule der Technik


```
[{
  "first_name":"James",
  "last_name":"Smith",
  "address":"6649 N Blue Gum St",
  "city":"New Orleans",
  "state":"LA",
  "zip":70116,
  "female":false,
  "phones":["504-621-8927","504-845-1427"]
},{
  "first_name":"Jenna",
  "last_name":"Darakjy",
  "address":"4 B Blue Ridge Blvd",
  "city":"Brighton",
  "state":"MI",
  "zip":48116,
  "female":true,
  "phones":["810-292-9388","810-374-9840"]}]
```

JSON

tgm
Die Schule der Technik

- JSON kennt Datentypen
  - Strings, Boolean, Zahlen
  - null
  - Arrays (in eckigen Klammern `[]`)
  - Objekte (in geschwungenen Klammern {})
- Objekte beinhalten eine (ungeordnete) Liste von Eigenschaften
- Eine Eigenschaft besteht aus einem Schlüssel (Zeichenkette) und einem Wert
- Hierarchien können dargestellt werden
- Einfache Repräsentation (Serialisierung) von Objekten der objektorientierten Programmierung
- Es sind jedoch keine komplexen Zusammenhänge darstellbar oder fortgeschrittene Validierungen möglich

XML

tgm
Die Schule der Technik

```
<?xml version="1.0"?>
<menu>
  <food calories="650">
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
      Two of our famous Belgian Waffles with plenty of real
          maple syrup
    </description>
  </food>
  <food calories="900">
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>
      Light Belgian waffles covered with strawberries
    </description>
  </food>
</menu>
```

XML

tgm
Die Schule der Technik

- **E**xtensible **M**arkup **L**anguage
- Eine Auszeichnungssprache
- Ebenfalls Textdokumente, die „menschenlesbar“ sein sollen
- Strukturierte Darstellung von Informationen (ggf. auch inkl. benutzerdefinierten Regeln)
- Ebenfalls zur Serialisierung von Objekten geeignet
- Anwendung bei Webapplikationen, Webservices, Konfigurationen, ...
- Spezifikation von der W3C

DOM

tgm
Die Schule der Technik

- **D**ocument-**O**bject-**M**odel
- Ebenfalls W3C-Standard
- Für den programmgesteuerten Zugriff auf XML-Dateien
- Darstellung des Dokuments als Baumstruktur
- Gesamtes Dokument wird in den Speicher geladen
- Wahlfreier Zugriff möglich

DOM

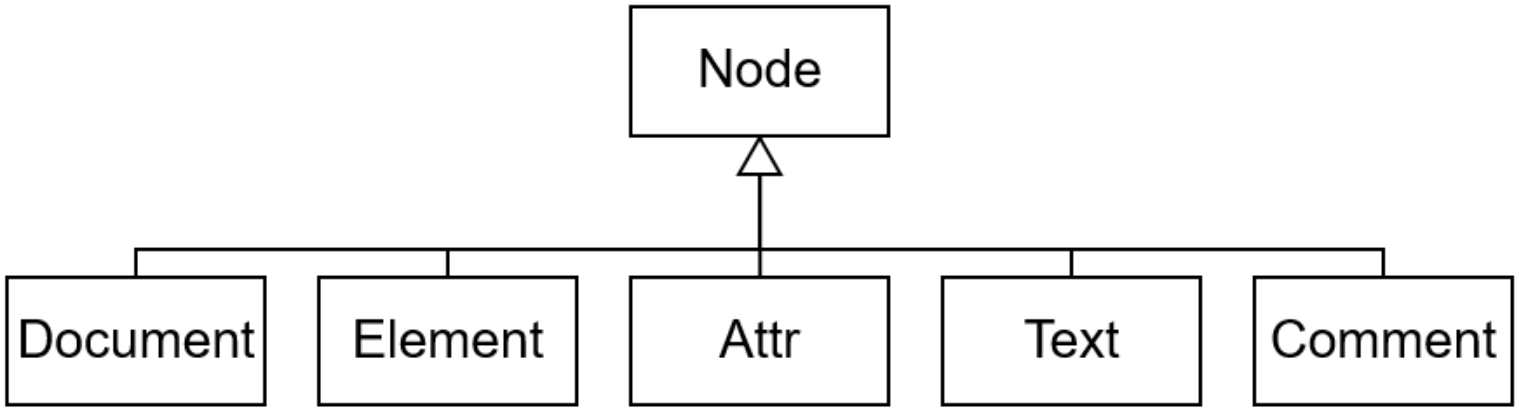
tgm
Die Schule der Technik

- Die Spezifikation von DOM unterscheidet prinzipiell zwischen
  - DOM Core: Wichtige gemeinsame Interfaces (Node, Element, ...) und Konzepte (Namespaces, ...)
  - DOM HTML: HTML-spezifische Definitionen (HTML-Elemente, Attribute, Methoden und Events)
  - DOM XML: Modell für Zugriff und Manipulation von XMl-Dateien
- Außerdem gibt es weitere Spezifikationen für weitere Konzepte (Validierung, XPath, ...)

DOM Node

tgm
Die Schule der Technik

Wichtigstes Interface: `Node`

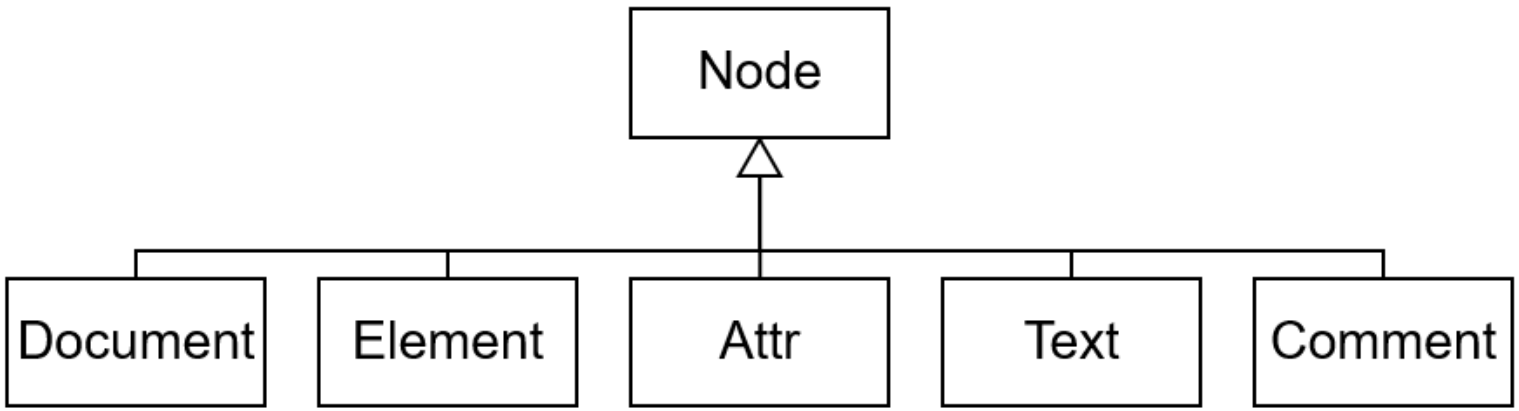


Alle Subtypen sind ebenfalls Interfaces (konkrete Implementierung ist egal)!

DOM Node

tgm
Die Schule der Technik

Je nach tatsächlichen Typ gibt es unterschiedliche mögliche Kindelemente und Rückgabewerte von `nodeValue` und `nodeName`!
Komplette Auflistung (**Lehrstoff!**):
http://www.w3schools.com/xml/dom_nodetype.asp



tgm
Die Schule der Technik

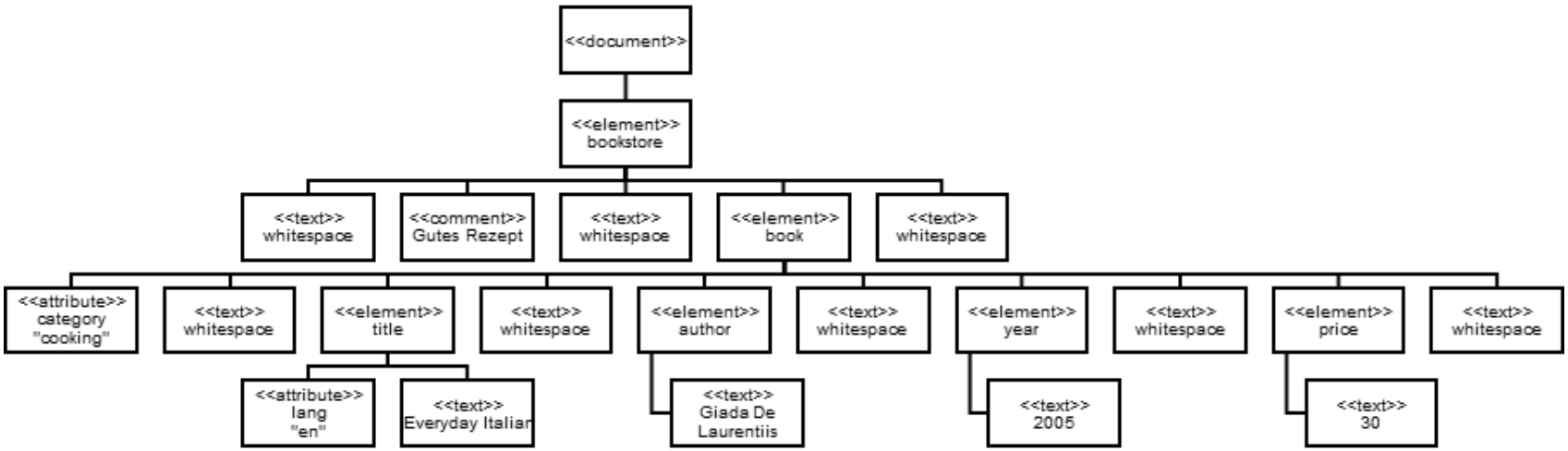
Wie viele Nodes besitzt der DOM-Baum?

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

<!-- Gute Rezepte -->
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>

</bookstore>
```

DOM

tgm
Die Schule der Technik

22 Nodes.



Auf Whitespaces aufpassen!

Python: XML-Dokument einlesen

tgm
Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")
print (et.tostring(doc.getroot()).decode('utf-8'))
```

Es gibt in Python mehrere XML-APIs – eine davon ist `ElementTree`

Python: find und text

tgm
Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")
print(doc.find("book/title").text)
```

`find` liefert das erste passende Element anhand des Namens bzw. Pfads zurück
`text` liefert den Textinhalt der Node

Python: findall

tgm
Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")

for element in doc.findall("book"):
    print(element.find("author").text + " " +
          element.find("year").text +
          ", EUR" + element.find("price").text)
```

`findall` liefert eine Liste an Elementen über den Namen oder Pfad

Python: attrib

tgm
Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")

for element in doc.findall("book"):
    print(element.attrib)
    print(element.attrib["category"])
```

attrib speichert als assoziatives Array alle Attribute des Elements

Python: Kindelemente und tag

tgm
Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")

for child in doc.getroot():
    print(child.tag, child.attrib)
```

Um alle Kindelemente zu erhalten, können `Element`-Objekte direkt iteriert werden

Python: write

tgm Die Schule der Technik

```
from xml.etree import ElementTree as et

doc = et.parse("bookstore.xml")

for element in doc.findall("book"):
    if "stock" not in element.attrib:
        element.attrib["stock"] = "10"

doc.write("bookstore2.xml")
```

Über **not in** wird geprüft, ob sich das Attribut im assoz. Array befindet
`write` schreibt den gesamten DOM-Baum in das angegebene File

Namespaces

Die Schule der Technik

Betrachten wir das folgende Beispiel:

```
<list>
  <table>
    <tr>
      <td>Name</td>
      <td>Preis</td>
    </tr>
  </table>
  <table>
    <name>Esstisch</name>
    <price currency="EUR">79</price>
  </table>
</list>
```

Namespaces

tgm
Die Schule der Technik

```
<list>
  <table>
    <tr>
      <td>Name</td>
      <td>Preis</td>
    </tr>
  </table>
  <table>
    <name>Esstisch</name>
    <price currency="EUR">79</price>
  </table>
</list>
```

Es sind zwei verschiedene <table> Elemente gemeint!

- HTML-Tabelle
- Tisch

Namespaces

Namespaces verhindern Namenskonflikte, wenn mehrere XML-Sprachen in einem Dokument gespeichert werden

```
<list>
  <h:table
    xmlns:h="http://www.w3.org/TR/html4">
    <h:tr>
      <h:td>Name</h:td>
      <h:td>Preis</h:td>
    </h:tr>
  </h:table>
  <f:table
    xmlns:f="http://www.w3schools.com/furniture">
    <f:name>Esstisch</f:name>
    <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

Namespaces

tgm
Die Schule der Technik

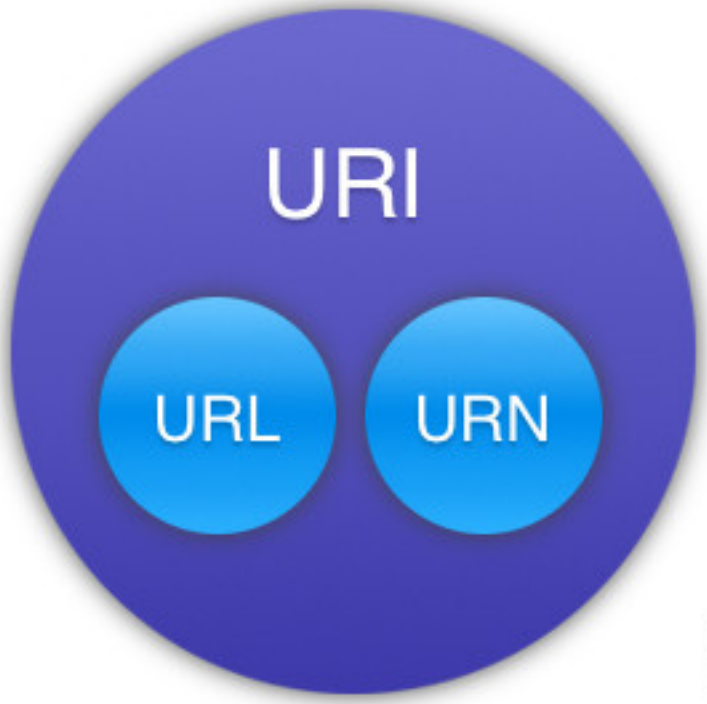
```
<h:table
   xmlns:h="http://www.w3.org/TR/html14">
   <h:tr>
   <h:td>Name</h:td>
   <h:td>Preis</h:td>
   </h:tr>
</h:table>
```

- Das `xmlns`-Attribut definiert einen neuen Namensraum
- Syntax: `xmlns:prefix="URI"`
- Zugehörigen Elementen wird der Präfix vorangestellt
- Gilt für: Aktuelles Element + alle Kindelemente mit dem Präfix

URI

```
xmlns:prefix="URI"
```

- Uniform Resource Identifier
- Muss (im aktuellen Dokument) eindeutig sein
- Präfix dient als Abkürzung
- Meistens wird eine URL verwendet



URI = Uniform Resource Indentifier
URL = Uniform Resource Locator
URN = Uniform Resource Name

Namespaces

tgm
Die Schule der Technik

```
<list
  xmlns:h="http://www.w3.org/TR/html4"
  xmlns:f="http://www.w3schools.com/furniture">
  <h:table>
    <h:tr>
      <h:td>Name</h:td>
      <h:td>Preis</h:td>
    </h:tr>
  </h:table>
  <f:table>
    <f:name>Esstisch</f:name>
    <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

Die Namespaces können auch direkt im Wurzelelement definiert werden

Default Namespaces

- Namespace ohne Präfix
- Syntax: `xmlns="URI"`
- Gilt für: Aktuelles Element + alle Kindelemente ohne Präfix

Beispiel: Zwei Default Namespaces

```
<list>
  <table xmlns="http://www.w3.org/TR/html4">
    <tr>
      <td>Name</td>
      <td>Preis</td>
    </tr>
  </table>
  <table
     xmlns="http://www.w3schools.com/furniture">
    <name>Esstisch</name>
    <price currency="EUR">79</price>
  </table>
</list>
```

Default Namespaces

Alternativ: Einen Default Namespace verwenden und jene Elemente kennzeichnen, die nicht zum Default Namespace gehören

```
<list
xmlns="http://www.w3.org/TR/html4"
xmlns:f="http://www.w3schools.com/furniture">
  <table>
    <tr>
      <td>Name</td>
      <td>Preis</td>
    </tr>
  </table>
  <f:table>
    <f:name>Esstisch</f:name>
    <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

Namespaces und Attribute

tgm Die Schule der Technik

- Attribute haben keinen Default Namespace
- Attribute erben nicht den Namespace ihres Elements
- Damit Attribute explizit einem Namespace haben, muss man sie ebenfalls mit dem Präfix auszeichnen (macht man aber kaum)
- Allerdings geht man meist implizit davon aus, dass ein Attribut zur selben Sprache des Elements gehört
- Bitte den folgenden Artikel lesen (**Lehrstoff!**): http://www.xmlplease.com/attributexmlns

```
<list
    xmlns:f="http://www.w3schools.com/furniture">
  <f:table>
  <f:name>Esstisch</f:name>
  <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

Validität

tgm Die Schule der Technik

Korrektheit eines XML-Dokuments ist über zwei Eigenschaften definiert:

- **Wohlgeformtheit (well-formedness)**
  Einhaltung der allgemeinen XML-**Syntax**, z.B.
  - Genau 1 Wurzelelement
  - Keine Überlappungen der Tags
  - Korrekte Attribute
- **Gültigkeit / Validität (validity)**
  Dokument ist wohlgeformt und entspricht einer bestimmten **Grammatik**, definiert durch entweder
  - einer DTD oder
  - einem XML Schema

Für den Datenaustausch ist es sinnvoll, eine DTD bzw. ein Schema zu verwenden, um die Gültigkeit eines XML-Dokuments überprüfen zu können

DTD

tgm
Die Schule der Technik

- Document Type Definition (Dokumenttypdefinition)
- Verwendet eine andere Syntax als XML selbst
- Definiert Elemente, Attribute, Struktur, Entitäten
- Befindet sich direkt nach der XML-Deklaration
- Kann *intern* oder *extern* sein
  - Externe DTDs werden über eine URI referenziert und befinden sich in einer eigenen Datei
    `<!DOCTYPE HandyKat SYSTEM "HandyKatalog.dtd">`
    `SYSTEM` steht für „private“ DTDs, `PUBLIC` für öffentliche
  - Eine interne DTD definiert die Grammatik direkt im Dokument
  - Es gibt auch Mischformen (externe DTD wird durch interne erweitert)

DTD Beispiel

tgm
Die Schule der Technik

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tracklist [
<!ELEMENT tracklist (track*, interpret*)>

<!ELEMENT track (name, length, review*)>
<!ATTLIST track id   ID  #REQUIRED
        style  (Rock|Funk|HipHop) "Rock">
<!ELEMENT name          (#PCDATA)>
<!ELEMENT length        (#PCDATA)>
<!ELEMENT review        (#PCDATA | link)*>
<!ELEMENT link       (#PCDATA)>
<!ATTLIST link  ref IDREF #IMPLIED>

<!ELEMENT interpret   (band | soloartist)>
<!ATTLIST interpret id  ID  #REQUIRED>
<!ELEMENT band       (name, person+)>
<!ELEMENT soloartist     (person)>

<!ELEMENT person       (firstname?, name, instrument*)>
<!ELEMENT firstname    (#PCDATA)>
<!ELEMENT instrument     (#PCDATA)>
<!ATTLIST person gender   (male|female) "female">
]>
<tracklist>...</tracklist>
```

DTD

tgm
Die Schule der Technik

- DTDs werden oft als veraltet bezeichnet
- Sie sind aber nach wie vor weit verbreitet (z.B. HTML)
- DTDs verwenden leider eine andere Syntax als XML
- DTDs sind nicht so mächtig wie XML-Schemas
  - Keine Datentypen
  - Kein Namespace-Support
  - Anzahl an Kindelementen kaum beschränkbar
  - Nicht über DOM manipulierbar
  - Text innerhalb von Elementen kann nicht eingeschränkt werden

XSD

tgm
Die Schule der Technik

- XML Schema Definition
- Basiert auf XML
- Unterstützung von Namespaces
- Unterstützung von Datentypen
- Unterstützung von Kardinalitäten
- u.v.m

XSD Beispiel

tgm Die Schule der Technik

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading"
          type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XSD Beispiel

tgm
Die Schule der Technik

```
<?xml version="1.0"?>

<note
  xmlns="http://www.w3schools.com"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

xs:schema

tgm
Die Schule der Technik

- Wurzelelement einer Schema-Datei
- Attribute:
  - XMLSchema Namespace
  - Namespace der XML-Datei (Instanz)
  - Default-Namespace (eher zu vermeiden)
  - Kann Namespaces der Instanz vorschreiben

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xs:schema>
```

simpleType

tgm
Die Schule der Technik

- Wir unterscheiden zwischen complexType und simpleType
- simpleType sind einfache Elemente
- Sie enthalten keine Attribute oder Kindelemente
- Somit kann nur Text zwischen den Tags stehen
- Beispiel aus Schema:

```
<xs:element name="to" type="xs:string"/>
```

- XML-Instanz:

```
<to>Tove</to>
```

simpleType: type

- Es gibt eingebaute primitive **Datentypen**

```
<xs:element name="lastname"
    type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
<xs:element name="start" type="xs:time"/>
<xs:element name="account"
    type="xs:decimal"/>
<xs:element name="discount"
    type="xs:boolean"/>
```

- XML-Instanz:

```
<lastname>Renoir</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
<start>10:03:30</start>
<account>-234.50</account>
<discount>true</discount>
```

simpleType: default & Wertebereiche

tgm Die Schule der Technik

- Es gibt die Möglichkeit, **Standardwerte** für leere Elemente zu definieren

```
<xs:element name="color" type="xs:string"
    default="red"/>
```

- Es können **Wertebereiche** festgelegt werden

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

simpleType: Eigene Datentypen

tgm
Die Schule der Technik

- Alternativ: **Eigenen Datentyp** erstellen und mehrfach verwenden

```
<xs:element name="age" type="ageType" />

<xs:simpleType name = "ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
```

simpleType: enumeration

tgm
Die Schule der Technik

- **Aufzählungen** (Wertemengen) können definiert werden

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

simpleType: pattern & length

tgm Die Schule der Technik

- Reguläre Ausdrücke (**Regular Expressions**) möglich

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Länge** des Inhalts kann eingeschränkt werden (auch min-max)

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

complexType: attribute

tgm
Die Schule der Technik

- Komplexe Typen erlauben das Definieren von Attributen und Kindelementen sowie gemischten Inhalten (Text + Kindelemente)
- Beispiel eines Attributes im Schema:

```
<xs:attribute name="lang" type="xs:string"/>
```

- XML-Instanz:

```
<lastname lang="EN">Smith</lastname>
```

- Attribute können optional (`use="optional"`) oder verpflichtend sein (`use="required"`)
- Attribute können Standardwerte haben (`default="123"`) oder fixiert sein (`fixed="10"`)

complexType: sequence

tgm Die Schule der Technik

- Kindelemente in bestimmter **Reihenfolge**

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

complexType: all

- Jedes Element muss **einmal** vorkommen, Reihenfolge egal

```
<xs:element name="employee">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <lastname>Smith</lastname>
  <firstname>John</firstname>
</employee>
```

complexType: choice

tgm Die Schule der Technik

- **Eines** der Elemente muss vorkommen

```
<xs:element name="employee">
  <xs:complexType>
    <xs:choice>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
</employee>
```

complexType: maxOccurs & minOccurs

- **Mehrmaliges** Vorkommen

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname"
          type="xs:string" maxOccurs="3"
          minOccurs="0"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
  <firstname>James</firstname>
  <lastname>Smith</lastname>
</employee>
```

complexType: Beispiele

tgm
Die Schule der Technik

- **Attribut** (ohne Kindelemente)

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid"
        type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<product prodid="1345" />
```

complexType: Beispiele

tgm Die Schule der Technik

- Element mit **Textinhalt und Attribut** (ohne Kindelemente)

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country"
            type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<shoesize country="france">35</shoesize>
```

complexType: Beispiele

- Element mit **gemischten** Inhalt

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name"
          type="xs:string"/>
      <xs:element name="orderid"
          type="xs:positiveInteger"/>
      <xs:element name="shipdate"
          type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<letter>Dear Mr.<name>John Smith</name>.
    Your order <orderid>1032</orderid> will
    be shipped on <shipdate> 2001-07-13
    </shipdate>.</letter>
```

Zusammenfassung

tgm
Die Schule der Technik

Namespaces, Wohlgeformtheit und Validität

- Namespaces erlauben mehrere XML-Sprachen im selben Dokument
- Ein XML-Dokument ist wohlgeformt, wenn es der allgemeinen XML-Syntax entspricht
- Ein XML-Dokument ist valide, wenn es seiner DTD oder seinem XML-Schema entspricht (Grammatik)
- DTDs verwenden eine andere Syntax und sind nicht so mächtig wie XSD
- XSD ist sehr komplex und erlaubt die Definition einer komplexen Grammatik inkl. Datentypen, Kardinalitäten etc.