

## III. Dokumentsprachen

- Auszeichnungssprachen
- HTML
- Cascading Stylesheets CSS
- XML-Grundlagen
- XML-Schema
- Die XSL-Familie
- Parse-Paradigmen und APIs für XML

# Parse-Paradigmen und APIs für XML

## Einordnung

XML-Dokumente sind uns bislang in **serialisierter Form**, z.B. als Inhalte von Dateien, begegnet. Ihre Manipulation durch Programme erfordert:

1. geeignete Repräsentation im Hauptspeicher
2. Möglichkeit zum Zugriff und zur Manipulation: API

API-Technologien zum Zugriff und zur Manipulation von XML-Dokumenten:

- DOM, Document Object Model
- SAX, Simple API for XML
- StAX, Streaming API for XML [\[XML.com\]](http://XML.com)
- XPP, Common API for XML Pull Parsing
- XML Data Binding

## Bemerkungen (API, IDL, Language Binding) :

- “An application programming interface (API) [...] is a type of software interface, offering a service to other pieces of software. [...] A program or a programmer that uses one of these parts is said to call that portion of the API.”  
“One purpose of APIs is to hide the internal details of how a system works, exposing only those parts a programmer will find useful and keeping them consistent even if the internal details later change.” [\[Wikipedia\]](#)
- Mit Hilfe einer Schnittstellenbeschreibungssprache (Interface Definition Language, IDL) lassen sich Objekte und die auf sie anwendbaren Methoden einschließlich Parametern und Datentypen beschreiben, ohne dabei die Eigenschaften einer bestimmten Programmiersprache zu verwenden. Ein Compiler kann diese Definitionen in eine bestimmte Programmiersprache und Rechnerarchitektur umsetzen, das so genannte *Language Binding*.  
[\[Wikipedia\]](#) [IDL](#), [Language Binding](#)
- Mit einem Language Binding für JavaScript, Java, C++, etc. stehen in diesen Programmiersprachen die mittels der IDL spezifizierten Methoden zur Verfügung, um auf ein HTML-Dokument – genauer: die unterliegenden DOM-Objekte – im Browser zuzugreifen.

# Parse-Paradigmen und APIs für XML

## DOM: Historie

- 1998 DOM Level 1, Recommendation. What is the DOM? [W3C [REC intro](#)]
- 2000 DOM Level 2 Core, Recommendation. [W3C [REC intro](#), update 2020]
- 2004 DOM Level 3 Core, Recommendation. [W3C [REC intro](#), update 2021]
- 2015 DOM4, Recommendation. Snapshot der 2015 [WHATWG DOM](#)-Spezifikation.
- 2018 Shadow DOM. [W3C [NOTE](#)] [[github](#)] [[SELFHTML](#)]
- 2022 DOM. Living Standard. [[whatwg](#), [status](#)]

## Java Language Bindings:

- 2002 DOM-Implementierung in Java 1.4 (JDK4).
- 2022 DOM-Implementierung in Java 18. [Javadoc [org.w3c.dom](#), [node](#)]

## Bemerkungen:

- Das Document Object Model, DOM, entstand aus dem Wunsch, Java- und JavaScript-Programme zwischen Browsern austauschbar zu machen. Vorläufer waren herstellerspezifische Ideen und Implementierungen für das sogenannte [Dynamic HTML](#).
- “[...] It [the Document Object Model] is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents, both HTML and XML.” [W3C [faq](#)]
- Level 1 und 2 von DOM enthalten noch keine Spezifikation dafür, wie ein Dokument in eine DOM-Struktur geladen oder aus ihr heraus gespeichert werden kann: sie setzen das Vorhandensein der Dokumente in einer Browser-Umgebung voraus. Ab DOM Level 3 gehören auch Methoden zum Laden und Speichern zur Spezifikation.  
2015 wurde mit DOM4 (eigentlich: DOM Level 4) die levelweise Entwicklung (und damit einhergehende Kompatibilitätsstrategie) aufgegeben. Neue Snapshots werden vom W3C auf Basis des WHATWG-Standards herausgegeben. [\[Wikipedia\]](#)
- [caniuse.com](#) zeigt die Unterstützung einzelner (DOM-)Features für verschiedene Browser. Beispiel: [getElementsByClassName](#).
- Die Interfaces der meisten DOM-Objekte sind von dem generischen [node](#)-Interface abgeleitet. Das [node](#)-Interface behandelt die gemeinsamen Anteile der verschiedenen Knoten eines XML-Baums.

# Parse-Paradigmen und APIs für XML

## DOM: Konzepte

- DOM modelliert ein Dokument gemäß seiner Struktur als eine Hierarchie von Knoten. [W3C [DOM Level 1](#), [DOM4](#)]
- Für die Knoten definiert DOM keine Datentypen, sondern **Objekte**. Die Spezifikation beschreibt **Interfaces** mit den für die Knotenobjekte erlaubten Operationen. [W3C [DOM Level 1](#)]
- Die Semantik der Knotenobjekte orientiert sich am [XML Information Set](#).
- Die DOM-Spezifikation ist neutral in Bezug auf Betriebssysteme und Programmiersprachen: die Interfaces sind in der *Interface Definition Language* Web IDL verfasst. [W3C [REC](#), [status](#)]
- Die sprachspezifische Umsetzung von DOM erfolgt durch sogenannte **Language Bindings**.

## Bemerkungen:

- Oft wird mit dem Begriff “DOM” auch die Datenstruktur zur Repräsentation eines Dokuments bezeichnet – und nicht die Programmierschnittstelle (API) zum Zugriff auf die Datenstruktur. Diese Sicht motiviert sich aus dem Verständnis der objektorientierten Programmierung:  
“The name *Document Object Model* was chosen because it is an *object model* in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed.” [W3C [DOM Level 1](#)]
- Wiederholung. Das W3C hat mittlerweile drei Datenmodelle für XML-Dokumente definiert: XML Information Set, XPath, Document Object Model (DOM). Das XPath-Datenmodell basiert auf einer Baumstruktur, die bei der Abfrage eines XML-Dokuments durchlaufen wird und ist dem [XML Information Set](#) ähnlich; DOM ist der Vorläufer beider Datenmodelle. DOM und das XPath-Datenmodell können als Interpretationen des XML Information Sets betrachtet werden. [[MSDN](#)]
- Illustration von Markup, DOM und gerenderter HTML-Seite im Live-DOM-Viewer. [[hixie.ch](#)]

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [personen.xml]

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="personen.xsl" ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburstag>23. Juni 1912</geburstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

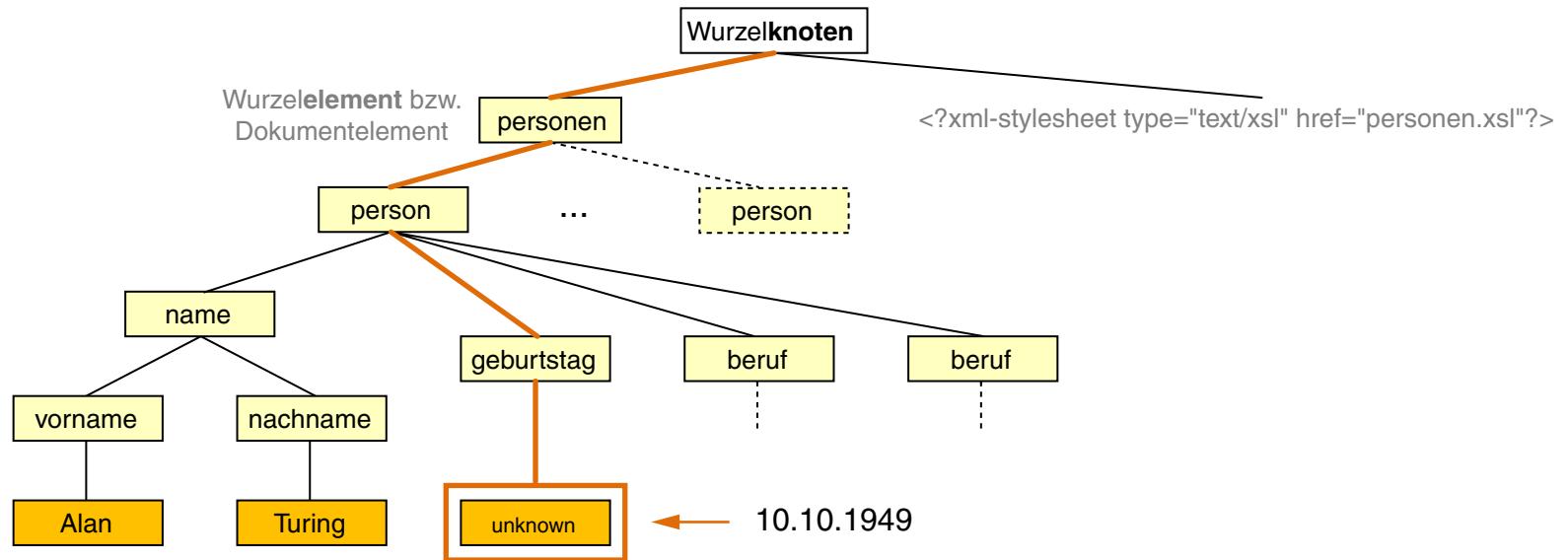
  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburstag>unknown</geburstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API

Java-Klasse:

```
package documentlanguages.xmlparser.dom;

import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.*;

public class DomParserExample {

    ①  public Document load(String filename) ...

    ②  public Node findPerson(Node node, String firstName, String lastName) ...
        private boolean matchesPerson(Node n, String firstName, ...

    ③  public void setBirthday(Node personNode, String birthday) ...

    ④  public void save(Document docNode, String filename, String encoding) ...

    public static void main(String[] args) ...
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

main-Methode:

```
public static void main(String[] args) throws Exception {  
  
    DomParserExample dpe = new DomParserExample();  
    Document docNode =  
        dpe.load("./bin/documentlanguages/xmlparser/personen.xml");  
  
    Node personNode = dpe.findPerson(docNode, "Judea", "Pearl");  
    dpe.setBirthday(personNode, "10.10.1949");  
  
    dpe.save(docNode,  
        "./bin/documentlanguages/xmlparser/personen-neu.xml", "UTF-8");  
}
```

# Parse-Paradigmen und APIs für XML

DOM: Anwendung der Java-API (Fortsetzung) [DOM, SAX, Data Binding]

(1) DOM-Parser instantiiieren und Dokument in DOM-Objektmodell einlesen:

```
public Document load(String filename)
    throws ParserConfigurationException, IOException, SAXException {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = dbf.newDocumentBuilder();
    return docBuilder.parse(new File(filename));
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface [\[Javadoc\]](#):

```
public Node findPerson(Node node, String firstName, String lastName){  
  
    if(matchesPerson(node, firstName, lastName)) {  
        return node;  
    }  
  
    // Perform Depth First Search (DFS).  
    NodeList nodeList = node.getChildNodes();  
    for(int i=0; i< nodeList.getLength(); ++i){  
        Node person = findPerson(nodeList.item(i), firstName, lastName);  
        if(person != null) {  
            return person;  
        }  
    }  
  
    return null;  
}
```

Vergleiche mit [XPath-Variante](#).

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName){  
    if(!n.getNodeName().equals("person")) {return false;}  
    NodeList personChildren = n.getChildNodes();  
    for(int i=0; i< personChildren.getLength(); ++i){  
        Node personChild = personChildren.item(i);  
        if(personChild.getNodeName().equals("name")){  
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;  
            NodeList nameChildren = personChild.getChildNodes();  
            for(int j=0; j< nameChildren.getLength(); ++j){  
                Node nameChild = nameChildren.item(j);  
                if(nameChild.getNodeName().equals("vorname") &&  
                    nameChild.getTextContent().equals(firstName))  
                {FIRSTNAME_OK = true;}  
                else if(nameChild.getNodeName().equals("nachname") &&  
                    nameChild.getTextContent().equals(lastName))  
                {LASTNAME_OK = true;}  
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}  
            }  
        }  
    }  
    return false;  
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName){  
    if(!n.getNodeName().equals("person")) {return false;}  
    NodeList personChildren = n.getChildNodes();  
    for(int i=0; i< personChildren.getLength(); ++i){  
        Node personChild = personChildren.item(i);  
        if(personChild.getNodeName().equals("name")){  
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;  
            NodeList nameChildren = personChild.getChildNodes();  
            for(int j=0; j< nameChildren.getLength(); ++j){  
                Node nameChild = nameChildren.item(j);  
                if(nameChild.getNodeName().equals("vorname") &&  
                    nameChild.getTextContent().equals(firstName))  
                {FIRSTNAME_OK = true;}  
                else if(nameChild.getNodeName().equals("nachname") &&  
                    nameChild.getTextContent().equals(lastName))  
                {LASTNAME_OK = true;}  
                if(FIRSTNAME_OK && LASTNAME_OK) {return true;}  
            }  
        }  
    }  
    return false;  
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface:

```
private boolean matchesPerson(Node n, String firstName, String lastName) {  
    if (!n.getNodeName().equals("person")) {return false;}  
    NodeList personChildren = n.getChildNodes();  
    for (int i=0; i< personChildren.getLength(); ++i) {  
        Node personChild = personChildren.item(i);  
        if (personChild.getNodeName().equals("name")) {  
            boolean FIRSTNAME_OK = false, LASTNAME_OK = false;  
            NodeList nameChildren = personChild.getChildNodes();  
            for (int j=0; j< nameChildren.getLength(); ++j) {  
                Node nameChild = nameChildren.item(j);  
                if (nameChild.getNodeName().equals("vorname") &&  
                    nameChild.getTextContent().equals(firstName))  
                {FIRSTNAME_OK = true;}  
                else if (nameChild.getNodeName().equals("nachname") &&  
                    nameChild.getTextContent().equals(lastName))  
                {LASTNAME_OK = true;}  
                if (FIRSTNAME_OK && LASTNAME_OK) {return true;}  
            } } }  
    return false;  
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(3a) <geburstag>-Knoten unter <person>-Knoten im Dokumentbaum ändern.

Variante mit generischem node-Interface:

```
public void setBirthday(Node personNode, String birthday) {  
  
    NodeList personChildren = personNode.getChildNodes();  
  
    for(int i=0; i<personChildren.getLength(); ++i){  
        Node personChild = personChildren.item(i);  
  
        if(personChild.getNodeName().equals("geburstag")){  
            System.out.println("[DOM] Updating geburtstag: " +  
                personChild.getTextContent() + " -> " +birthday);  
            personChild.setTextContent(birthday);  
        }  
    }  
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(3b) <geburstag>-Knoten unter <person>-Knoten im Dokumentbaum ändern.

Variante mit Element-Interface [\[Javadoc\]](#):

```
public void setBirthday(Node personNode, String birthday) {  
  
    Element person = (Element) personNode;  
    NodeList birthdayElements = person.getElementsByTagName("geburstag");  
  
    if (birthdayElements.getLength() > 0) {  
        System.out.println("[DOM] Updating geburstag: " +  
            birthdayElements.item(0).getTextContent() + " -> " + birthday);  
        birthdayElements.item(0).setTextContent(birthday);  
    }  
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(4) Dokumentmodell serialisieren [\[Javadoc\]](#):

```
public void save(Document docNode, String filename, String encoding)
    throws IOException, TransformerException,
UnsupportedEncodingException {

TransformerFactory tf = TransformerFactory.newInstance();
Transformer serializer = tf.newTransformer();

serializer.setOutputProperty(OutputKeys.ENCODING, encoding);
serializer.transform(new DOMSource(docNode),
    new StreamResult(new FileOutputStream(filename)));
}
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(4) Dokumentmodell serialisieren [\[Javadoc\]](#):

```
public void save(Document docNode, String filename, String encoding)
    throws IOException, TransformerException,
    UnsupportedEncodingException {

    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer serializer = tf.newTransformer();

    serializer.setOutputProperty(OutputKeys.ENCODING, encoding);
    serializer.transform(new DOMSource(docNode),
        new StreamResult(new FileOutputStream(filename)));
}
```

Aufruf in der Shell:

```
[user@pc WORKINGDIR] $ java -cp CLASSPATH
    documentlanguages.xmlparser.dom.DomParserExample
```

**[DOM] Updating geburtstag: unknown -> 10.10.1949**

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2b) Navigation im Dokumentbaum. Zugriff mit **xpath-Interface** [Javadoc [javax.xml.xpath, evaluate](#)]:

```
public Node findPerson(Node node, String firstName, String lastName)
    throws XPathExpressionException{

    XPathFactory factory = XPathFactory.newInstance();
    XPath xpath = factory.newXPath();
    String path = "//person[name/vorname= '" + firstName + "' and " +
        "name/nachname= '" + lastName + "' ]";

    return (Node) xpath.evaluate(path, node, XPathConstants.NODE);
}
```

Vergleiche mit [DFS-Variante](#).

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Python-API

main-Methode:

```
from urllib import etree
# Standard library provides same API, but has limited XPath support:
# from xml.etree import ElementTree as etree

def main():
    document = etree.parse('../personen.xml')
    person = find_person(document.getroot(), 'Judea', 'Pearl')
    if person is not None:
        set_birthday(person, '10.10.1949')

    document.write('../personen-neu.xml', encoding='utf-8',
                  xml_declaration=True, pretty_print=True)

if __name__ == '__main__':
    main()
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Python-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface:

```
def find_person(node, first_name, last_name):
    if matches_person(node, first_name, last_name):
        return node

    for child in node:
        person = find_person(child, first_name, last_name)
        if person is not None:
            return person

    return None
```

Vergleiche mit XPath-Variante.

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Java-API (Fortsetzung)

(2a) Navigation im Dokumentbaum. DFS mit generischem node-Interface:

```
def matches_person(node, first_name, last_name):
    if node.tag != 'person':
        return False

    # Iterate person node children
    first_name_ok = False
    last_name_ok = False

    for child_i in node:
        if child_i.tag != 'name':
            continue

        # Iterate person/name children
        for child_j in child_i:
            if child_j.tag == 'vorname' and child_j.text == first_name:
                first_name_ok = True
            if child_j.tag == 'nachname' and child_j.text == last_name:
                last_name_ok = True

    return first_name_ok and last_name_ok
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Python-API (Fortsetzung)

(3) <geburtstag>-Knoten unter <person>-Knoten im Dokumentbaum ändern.

```
def set_birthday(node, birthday):
    for child in node:
        if child.tag == 'geburtstag':
            print(f'[DOM] Updating geburtstag: {child.text} -> {birthday}')
            child.text = birthday
            break
```

# Parse-Paradigmen und APIs für XML

## DOM: Anwendung der Python-API (Fortsetzung)

(2b) Navigation im Dokumentbaum. Zugriff mit `xpath`-Interface:

```
# XPath implementation only supported with lxml.

def find_person_xpath(node, first_name, last_name):
    match = node.xpath(
        f'//person[name/vorname="{first_name}" and \
            name/nachname="{last_name}"]')
    if len(match) >= 1:
        return match[0]
    return None
```

Vergleiche mit DFS-Variante.

# Parse-Paradigmen und APIs für XML

SAX: Konzepte [\[Wikipedia\]](#)

Verwendung eines SAX-Parsers in folgenden Schritten:

1. Implementierung und Einbindung eines Content-Handlers.
2. Instantiierung einer spezifischen Parser-Ausprägung.
3. Aufruf des Parsers.

Konsequenzen:

- Das Dokument definiert die Ereignisse, auf die der Parser reagiert.
- Parse-Methoden werden nicht explizit vom Programmierer aufgerufen.
- Das Programm weist keinen erkennbaren Parse-Kontrollfluss auf.

Stichwort: Push-Prinzip

# Parse-Paradigmen und APIs für XML

## SAX: Konzepte (Fortsetzung)

Methoden (Callback-Funktionen) der Content-Handler-Schnittstelle:

Methoden	Beschreibung
startDocument()	einmaliger Aufruf bei Beginn eines Dokuments
startElement()	Aufruf bei Beginn (öffnender Tag) eines Elements
characters()	Aufruf bei der Verarbeitung von Zeichenkettendaten innerhalb eines Elements
ignorableWhitespace()	Aufruf beim Auftreten ignorierbarer Leerzeichen
endElement()	Aufruf bei Erreichen eines Elementendes
endDocument()	letztes Ereignis eines Parse-Vorgangs

## Bemerkungen:

- Die Simple API for XML, SAX, stellt einen „leichtgewichtigen“ Ansatz für die ereignisbasierte Verarbeitung von XML-Dokumenten dar. SAX ist kein Parser, sondern ein Gerüst in Form einer Schnittstellensammlung, um Parser zu implementieren.
- Das Push-Prinzip stellt nur minimale Speicheranforderungen: nur die Tiefe des Dokumentbaums und die Übergabeparameter der Callback-Funktionen sind verantwortlich für den variablen Teil des “Memory Footprint”.
- Aus dem Prinzip der SAX-Verarbeitung folgt, dass keine (in-Memory) Modifikationen am Eingabedokument möglich sind. Modifikationen werden durch eine veränderte Ausgabe des Eingabedokuments realisiert. Der mögliche Umfang dieser Transformationen hängt davon ab, wieviel von dem Eingabedokument während des Parse-Vorgangs zwischenspeichert wird.

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="personen.xsl" ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburstag>23. Juni 1912</geburstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburstag>unknown</geburstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xmlstylesheet type="text/xsl" href="personen.xsl" ?>  
  
<personen>  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
  <person>  
    <name>  
      <vorname>Judea</vorname>  
      <nachname>Pearl</nachname>  
    </name>  
    <geburstag>unknown</geburstag>  
    <beruf>Informatiker</beruf>  
  </person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
<personen>  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
  <person>  
    <name>  
      <vorname>Judea</vorname>  
      <nachname>Pearl</nachname>  
    </name>  
    <geburstag>unknown</geburstag>  
    <beruf>Informatiker</beruf>  
  </person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
<personen>                                         startElement ()  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
<person>  
  <name>  
    <vorname>Judea</vorname>  
    <nachname>Pearl</nachname>  
  </name>  
  <geburstag>unknown</geburstag>  
  <beruf>Informatiker</beruf>  
</person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
<personen>                                         startElement ()  
  <person>                                         startElement ()  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
<person>  
  <name>  
    <vorname>Judea</vorname>  
    <nachname>Pearl</nachname>  
  </name>  
  <geburstag>unknown</geburstag>  
  <beruf>Informatiker</beruf>  
</person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
<personen>                                         startElement ()  
  <person>                                         startElement ()  
    <name>                                         startElement ()  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
<person>  
  <name>  
    <vorname>Judea</vorname>  
    <nachname>Pearl</nachname>  
  </name>  
  <geburstag>unknown</geburstag>  
  <beruf>Informatiker</beruf>  
</person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
  
<personen>                                         startElement ()  
  <person>                                         startElement ()  
    <name>                                         startElement ()  
      <vorname>Alan</vorname>                     startElement ()  
      <nachname>Turing</nachname>  
    </name>                                         startElement ()  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
<person>  
  <name>  
    <vorname>Judea</vorname>  
    <nachname>Pearl</nachname>  
  </name>  
  <geburstag>unknown</geburstag>  
  <beruf>Informatiker</beruf>  
</person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
  
<personen>                                         startElement ()  
  <person>                                         startElement ()  
    <name>                                         startElement ()  
      <vorname>Alan</vorname>                     startElement () characters () ...  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
<person>  
  <name>  
    <vorname>Judea</vorname>  
    <nachname>Pearl</nachname>  
  </name>  
  <geburstag>unknown</geburstag>  
  <beruf>Informatiker</beruf>  
</person>  
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument ()  
<?xml-stylesheet type="text/xsl" href="perso|processingInstruction()  
  
<personen>  
  <person>  
    <name>  
      <vorname>Alan</vorname>  
      <nachname>Turing</nachname>  
    </name>  
    <geburstag>23. Juni 1912</geburstag>  
    <beruf>Mathematiker</beruf>  
    <beruf>Informatiker</beruf>  
  </person>  
  
  <person>  
    <name>  
      <vorname>Judea</vorname>  
      <nachname>Pearl</nachname>  
    </name>  
    <geburstag>unknown</geburstag>  
    <beruf>Informatiker</beruf>  
  </person>  
</personen>
```

startElement ()  
startElement ()  
startElement ()  
startElement () characters () ...  
startElement () characters () ...  
endElement ()  
startElement () characters () ...  
startElement () characters () ...  
startElement () characters () ...  
endElement ()

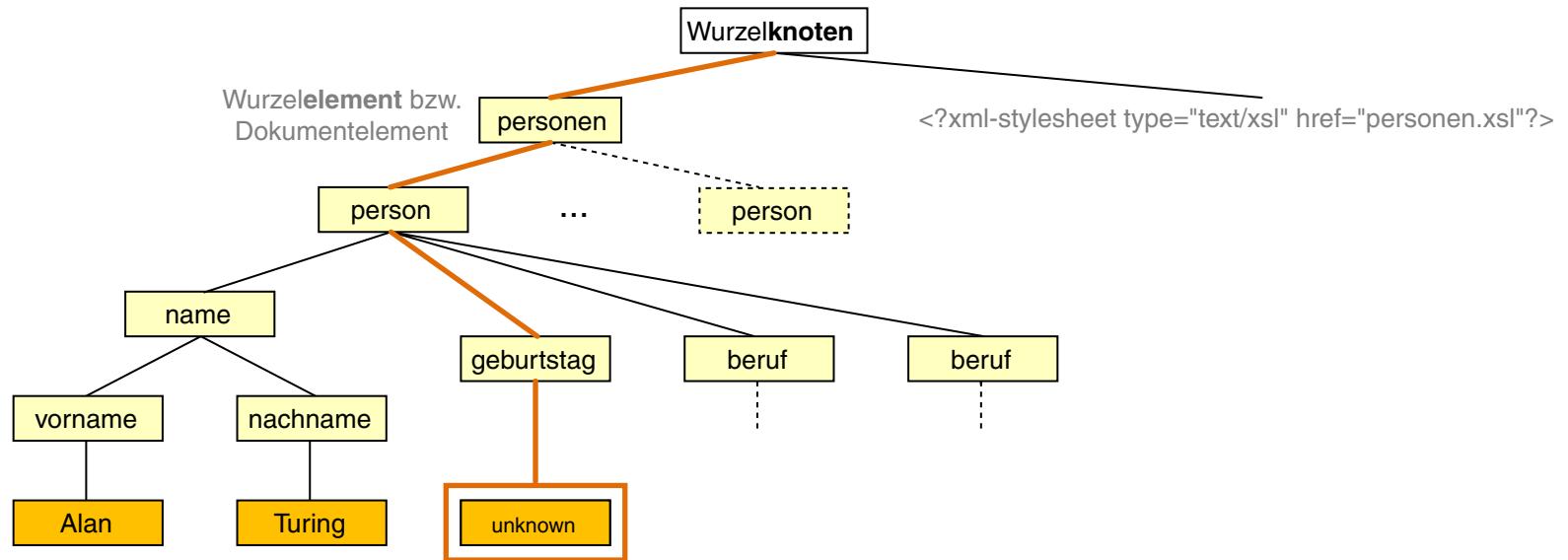
...

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag ausgeben. [vgl. [DOM-Aufgabe](#)]



# Parse-Paradigmen und APIs für XML

SAX: Anwendung der Java-API [DOM, SAX, Data Binding]

SAX-Parser instantiiieren und XML-Ereignisstrom öffnen [Javadoc]:

```
package documentlanguages.xmlparser.sax;

import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;

public class SAXParserExample {

    public void load(String filename, DefaultHandler handler)
        throws SAXException, IOException, ParserConfigurationException {
        SAXParser parser = SAXParserFactory.newInstance().newSAXParser();
        parser.parse(filename, handler);
    }

    public static void main(String[] args)
        throws SAXException, IOException, ParserConfigurationException {
        SAXParserExample spe = new SAXParserExample();
        DefaultHandler handler = new SAXParserExampleHandler("Judea", "Pearl");
        spe.load("./bin/documentlanguages/xmlparser/personen.xml", handler);
    }
}
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Java-API (Fortsetzung) [Callback-Funktionen]

Schnittstellenklasse mit eigenen Callback-Funktionen [\[Javadoc\]](#):

```
package documentlanguages.xmlparser.sax;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SAXParserExampleHandler extends DefaultHandler{

    // Boolean state variables for modeling the states of the parser.
    boolean parseVorname, parseNachname, parseGeburtstag;
    String vorname, nachname, geburtstag;
    String targetFirstName, targetLastName;

    public SAXParserExampleHandler(String firstName, String lastName) {
        targetFirstName = firstName;
        targetLastName = lastName;
    }

    public void startElement(String uri, String localName, String qName, ...
    public void characters(char[] ch, int start, int length) ...
    public void endElement(String uri, String localName, String qName) ...
}
```

## Bemerkungen:

- Die Klasse SAXParserExampleHandler ist von der Klasse DefaultHandler abgeleitet, die für jede Callback-Funktion eine Default-Implementierung enthält, die nichts tut. Somit müssen nur diejenigen Methoden überschrieben werden, die genau die Ereignisse verarbeiten, an denen man interessiert ist.

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Java-API (Fortsetzung) [Callback-Funktionen]

Verarbeitung von Elementstart-Ereignissen (= Zustandscodierung) :

```
public void startElement(String uri, String localName, String qName,
    Attributes attributes) {

    // Encode parse situation with the respective Boolean state variable.
    if(qName.equals("vorname")) {parseVorname = true;}
    if(qName.equals("nachname")) {parseNachname = true;}
    if(qName.equals("geburtstag")) {parseGeburtstag = true;}

}
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Java-API (Fortsetzung) [Callback-Funktionen]

Verarbeitung von Character-Data-Ereignissen (= Einlesen):

```
public void characters(char[] ch, int start, int length) {  
  
    if(parseVorname) {  
        vorname = new String(ch, start, length);  
        parseVorname = false;  
    }  
    if(parseNachname) {  
        nachname = new String(ch, start, length);  
        parseNachname = false;  
    }  
    if(parseGeburtstag) {  
        geburtstag = new String(ch, start, length);  
        parseGeburtstag = false;  
    }  
}
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Java-API (Fortsetzung) [Callback-Funktionen]

Verarbeitung von Elementende-Ereignissen:

```
public void endElement(String uri, String localName, String qName) {  
  
    // When leaving a <person>-element...  
    if (qName.equals("person")) {  
  
        // If the names are correct, print the birthday.  
        if (vorname.equals(targetFirstName) &&  
            nachname.equals(targetLastName)) {  
            System.out.println("[SAX] " + targetFirstName + " "  
                + targetLastName + "'s Geburtstag: " + geburtstag);  
        }  
  
        // Reset person data.  
        vorname = null; nachname = null; geburtstag = null;  
    }  
}
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Java-API (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR] $ java -cp CLASSPATH  
documentlanguages.xmlparser.sax.SAXParserExample
```

```
[SAX] Judea Pearl's Geburtstag: unknown
```

## Bemerkungen:

- Um sinnvoll auf Ereignisse reagieren zu können, muss sich der Zustand gemerkt werden, in dem sich der Parser befindet.  
Im Beispiel: Tritt ein Character-Data-Ereignis ein, so soll sich die Zeichenkette nur dann gemerkt werden, falls vorher der Start-Tag eines <vorname>-, <nachname>- oder <geburtstag>-Elements geparsed wurde. Stichwort: endlicher Automat
- Im Beispiel sind die Zustände des endlichen Automaten durch Variablen wie `parseVorname`, `parseNachname` oder `parseGeburtstag` codiert.

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Python-API

SAX-Parser instantiiieren und XML-Ereignisstrom öffnen:

```
from xml.sax import make_parser, handler

def main():
    person_handler = PersonHandler('Judea', 'Pearl')
    parser = make_parser()
    parser.setContentHandler(person_handler)
    parser.parse('../personen.xml')

if __name__ == '__main__':
    main()
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Python-API (Fortsetzung) [Callback-Funktionen]

Schnittstellenklasse mit eigenen Callback-Funktionen:

```
class PersonHandler(handler.ContentHandler):  
    def __init__(self, first_name, last_name):  
        super().__init__()  
  
        self.parse_vorname = False  
        self.parse_nachname = False  
        self.parse_geburtstag = False  
  
        self.vorname = None  
        self.nachname = None  
        self.geburtstag = None  
  
        self.target_first_name = first_name  
        self.target_last_name = last_name  
  
    def startElement(self, name, attrs) ...  
    def characters(self, content) ...  
    def endElement(self, name) ...
```

# Parse-Paradigmen und APIs für XML

SAX: Anwendung der Python-API (Fortsetzung) [\[Callback-Funktionen\]](#)

Verarbeitung von Elementstart-Ereignissen (= Zustandscodierung):

```
def startElement(self, name, attrs):
    # Encode parse situation with the respective Boolean state variable.
    if name == 'vorname':
        self.parse_vorname = True
    elif name == 'nachname':
        self.parse_nachname = True
    elif name == 'geburtstag':
        self.parse_geburtstag = True
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Python-API (Fortsetzung) [Callback-Funktionen]

Verarbeitung von Character-Data-Ereignissen (= Einlesen):

```
def characters(self, content):
    if self.parse_vorname:
        self.vorname = content
        self.parse_vorname = False
    elif self.parse_nachname:
        self.nachname = content
        self.parse_nachname = False
    elif self.parse_geburtstag:
        self.geburtstag = content
        self.parse_geburtstag = False
```

# Parse-Paradigmen und APIs für XML

## SAX: Anwendung der Python-API (Fortsetzung) [Callback-Funktionen]

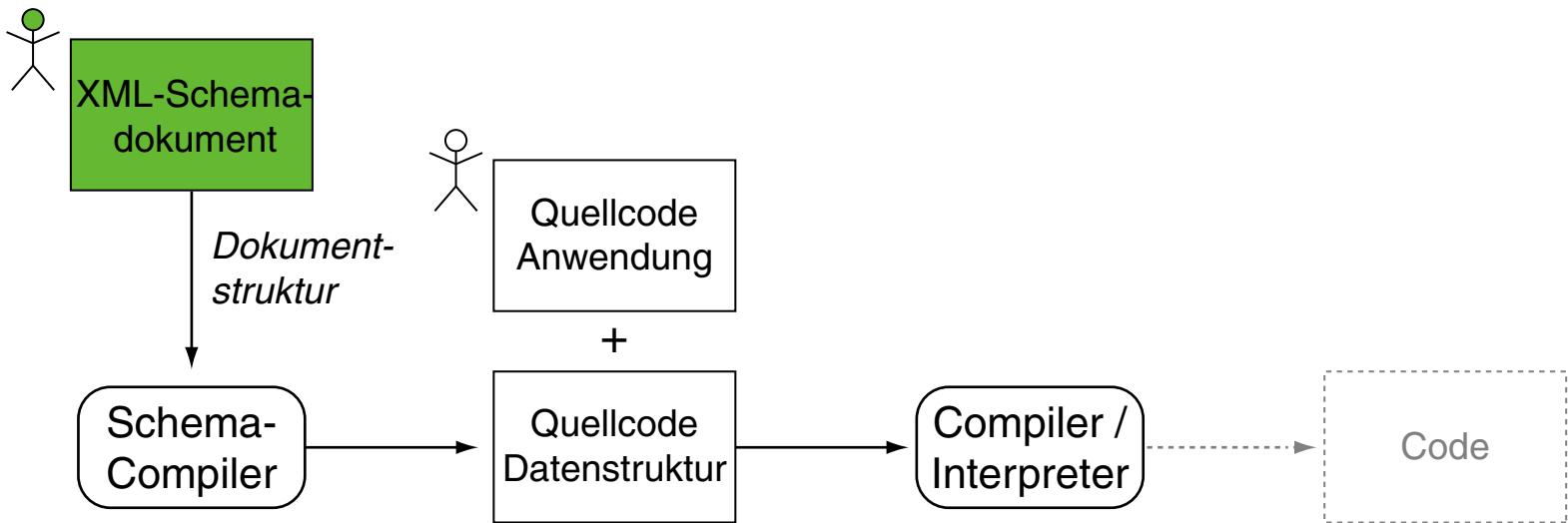
Verarbeitung von Elementende-Ereignissen:

```
def endElement(self, name):
    # When leaving a <person> element.
    if name == 'person':
        # If the names are correct, print the birthday.
        if self.vorname == self.target_first_name and \
           self.nachname == self.target_last_name:
            print(f' [SAX] {self.vorname} {self.nachname}\''s Geburtstag:',
                  self.geburtstag)

    # Reset person data.
    self.vorname, self.nachname, self.geburtstag = None, None, None
```

# Parse-Paradigmen und APIs für XML

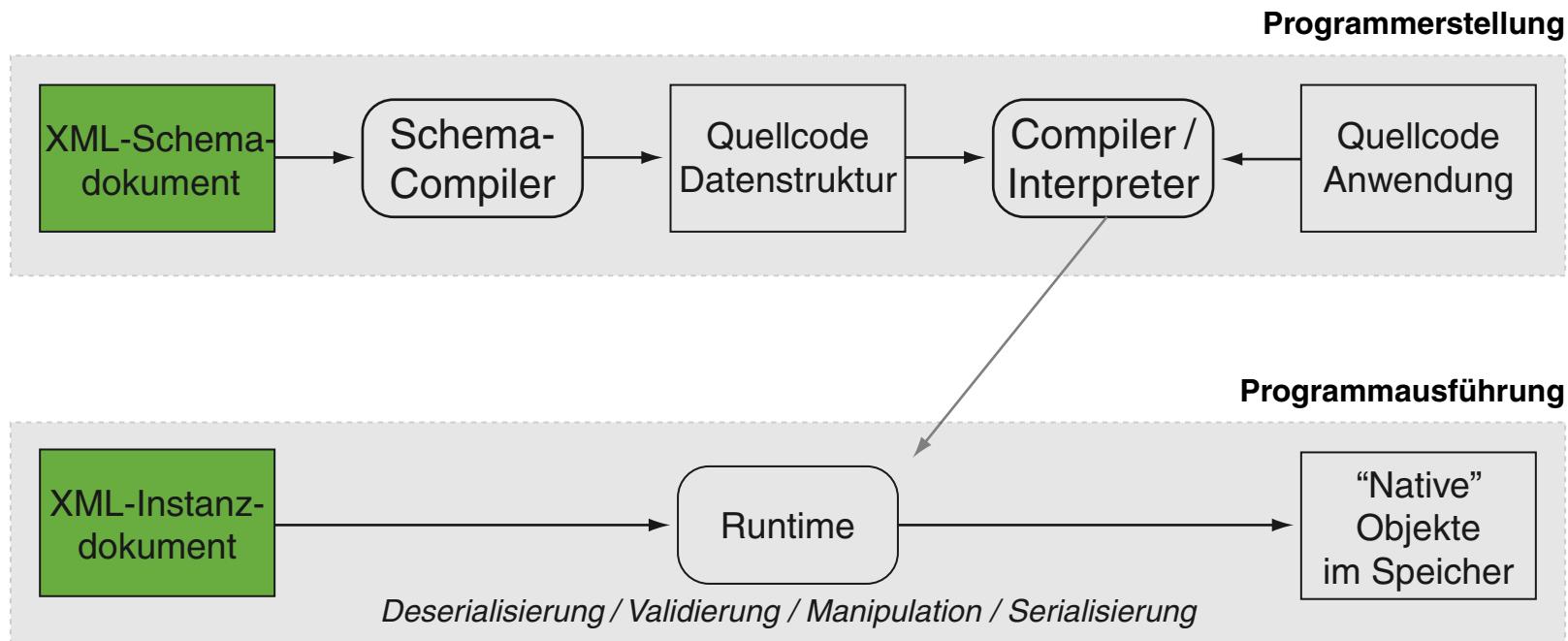
## XML Data Binding: Konzepte I [Konzepte II]



- Ein Schema-Compiler erzeugt für die Datentypen im XML-Schema entsprechende Klassen der Zielsprache, mit denen sich die Elemente von schemavaliden XML-Dokumenten erzeugen und manipulieren lassen.
- Die Anwendung setzt direkt auf den erzeugten Klassen auf.

# Parse-Paradigmen und APIs für XML

XML Data Binding: Konzepte I (Fortsetzung) [\[Konzepte II\]](#)



- XML-Schema für Programmerstellung.
- Instanzdokument für Programmausführung.

## Bemerkungen:

- Unter Data Binding versteht man die Abbildung eines Datenmodells (hier: XML-Schema) auf die Konzepte einer Zielsprache. Data Binding macht die Daten auf Basis der Datenstrukturen der gewählten Zielsprache verfügbar.
- Data Binding ist attraktiv, wenn Mechanismen für dessen Automatisierung existieren: die Konzepte der Zielsprache (Typ-Constraints, Datenstruktur-Mapping, Setter / Getter-Methoden, etc.) werden aus Sicht des Programmierers transparent gehandelt.
- DOM versus XML Data Binding. “In the DOM approach, the parser creates a tree of objects that represents the content and organization of data in the document. The application can then navigate through the tree in memory to access the data it needs. DOM data, however, is contained in objects of a single type, linked according to the XML document’s structure, with individual node objects containing an element, an attribute, a CDATA section, etc. *Values are invariably provided as strings.*”

“Deserializing [Java: unmarshalling] an XML document with the appropriate method [Java: JAXB] also results in a tree of objects, with the significant difference being that the nodes in this tree correspond to XML elements, which contain attributes and the content as instance variables and refer to child elements by object references.” [Java: [Oracle](#), [GlassFish](#)]

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument [personen.xml]

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="personen.xsl" ?>

<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburstag>23. Juni 1912</geburstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

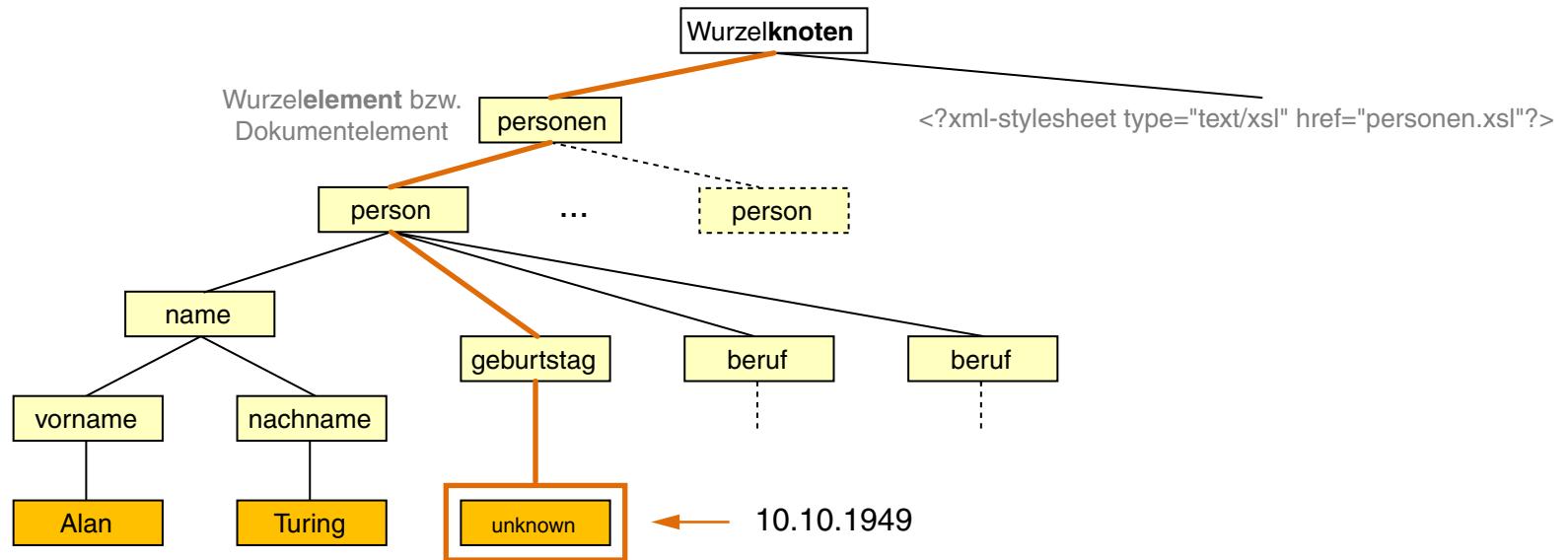
  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburstag>unknown</geburstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

# Parse-Paradigmen und APIs für XML

## XML-Beispieldokument (Fortsetzung)

Aufgabe:

1. Die Person „Judea Pearl“ finden.
2. Seinen Geburtstag auf einen bestimmten Wert setzen.



# Parse-Paradigmen und APIs für XML

## XML-Schema für Beispieldokument [personen.xsd]

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

  <xs:complexType name="NameType">
    <xs:sequence>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="nachname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PersonType">
    <xs:sequence>
      <xs:element name="name" type="NameType"/>
      <xs:element name="geburtstag" type="xs:string"/>
      <xs:element name="beruf" type="xs:string" minOccurs="0" .../>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PersonenType">
    <xs:sequence>
      <xs:element name="person" type="PersonType" minOccurs="0" .../>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="personen" type="PersonenType"/>
</xs:schema>
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API [Konzepte II]

Generierung der Klassen für die komplexen Datentypen inklusive Getter- und Setter-Methoden (= Data Binding) mit dem Java-Schema-Compiler `xjc`:

1. [user@pc *SOURCEDIR*] \$ `jaxb-ri/bin/xjc.sh`  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd

2. Entstandene Package-Struktur:

`documentlanguages.xmlparser.jaxb.generated.personen.`  
`NameType.java`  
`PersonenType.java`  
`PersonType.java`

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung) [Konzepte II]

Java-Klasse:

```
package documentlanguages.xmlparser.jaxb;

import java.io.*;
import java.util.List;
import javax.xml.bind.*;
import documentlanguages.xmlparser.jaxb.generated.personen.*;

public class JAXBParserExample1 {

    public PersonenType load(String filename) ...
    public void setBirthday(PersonenType personen, ...
    public void save(PersonenType personen, String filename) ...

    public static void main(String[] args) throws JAXBException, IOException {
        JAXBParserExample1 pe = new JAXBParserExample1();
        PersonenType personen =
            pe.load("./bin/documentlanguages/xmlparser/personen.xml");
        pe.setBirthday(personen, "Judea", "Pearl", "10.10.1949");
        pe.save(personen, "./bin/documentlanguages/xmlparser/personen-neu.xml");
    }
}
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung) [DOM, SAX, Data Binding]

<personen>-Parser instantiiieren, Dokument parsen und <personen>-Element im Speicher als Java-Objekt anlegen:

```
public PersonenType load(String filename)
    throws FileNotFoundException, JAXBException{

    // Create JAXBContext.
    JAXBContext jc = JAXBContext.newInstance(PersonenType.class);

    // Create unmarshaller.
    Unmarshaller u = jc.createUnmarshaller();

    // Unmarshal an instance document into a tree of Java content objects
    // composed of classes from the xmlparser.generated.personen package.
    return u.unmarshal
        (new StreamSource(filename), PersonenType.class).getValue();
}
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung)

<geburstag>-Element suchen und neu setzen:

```
public void setBirthday(PersonenType personen, String targetFirstName,
    String targetLastName, String birthday) {

    List<PersonType> personList = personen.getPerson();

    for (PersonType person : personList) {
        NameType name = person.getName();
        if (name.getNachname().equals(targetLastName) &&
            name.getVorname().equals(targetFirstName)) {
            System.out.println("[JAXB] Updating \"geburstag\": "
                + person.getGeburtstag() + " -> " + birthday);
            person.setGeburtstag(birthday);
            break;
        }
    }
}
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung)

Geändertes <personen>-Element speichern:

```
public void save(PersonenType personen, String filename)
throws IOException, JAXBException {
    // Create JAXBContext.
    JAXBContext jc = JAXBContext.newInstance(PersonenType.class);

    // Create marshaller.
    Marshaller m = jc.createMarshaller();

    // Produce formatted output.
    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

    // Write to file.
    m.marshal(new ObjectFactory().createPersonen(personen), new File(filename));
}
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR] $ jaxb-ri/bin/xjc.sh  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd
```

```
Java major version: 18  
parsing a schema...  
compiling a schema...  
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java  
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonenType.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung)

Aufruf in der Shell:

```
[user@pc SOURCEDIR] $ jaxb-ri/bin/xjc.sh  
-p documentlanguages.xmlparser.jaxb.generated.personen  
documentlanguages/xmlparser/personen.xsd  
  
Java major version: 18  
parsing a schema...  
compiling a schema...  
documentlanguages/xmlparser/jaxb/generated/personen/NameType.java  
documentlanguages/xmlparser/jaxb/generated/personen/ObjectFactory.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonenType.java  
documentlanguages/xmlparser/jaxb/generated/personen/PersonType.java  
  
[user@pc SOURCEDIR] $ javac -d ..../bin -cp CLASSPATH  
documentlanguages/xmlparser/jaxb/JAXBParserExample1.java  
  
[user@pc WORKINGDIR] $ java -cp CLASSPATH  
documentlanguages.xmlparser.jaxb.JAXBParserExample1  
[JAXB] Updating "geburtstag": unknown -> 10.10.1949
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Python-API

Generierung der Klassen für die komplexen Datentypen inklusive Getter- und Setter-Methoden (= Data Binding) mit dem Python-Schema-Compiler `xsdata`:

1. [user@pc *SOURCEDIR*] \$ `xsdata generate`  
`documentlanguages/xmlparser/personen.xsd`

2. Im Dateisystem:

```
xml_parser/data_binding/generated/  
    __init__.py  
    personen.py
```

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Python-API (Fortsetzung)

main-Methode:

```
from xsdata.formats.dataclass.parsers import XmlParser
from xsdata.formats.dataclass.serializers import XmlSerializer
from xsdata.formats.dataclass.serializers.config import SerializerConfig

# Generate with: xsdata generate xml/personen.xsd.
from generated.personen import Personen

def main():
    parser = XmlParser()
    personen = parser.parse('../personen.xml', Personen)

    set_birthday(personen, 'Judea', 'Pearl', '10.10.1949')

    config = SerializerConfig( pretty_print=True,
        no_namespace_schema_location='../personen.xsd')
    serializer = XmlSerializer(config=config)
    with open('../personen-neu.xml', 'w') as f:
        serializer.write(f, personen)

if __name__ == '__main__':
    main()
```

# Parse-Paradigmen und APIs für XML

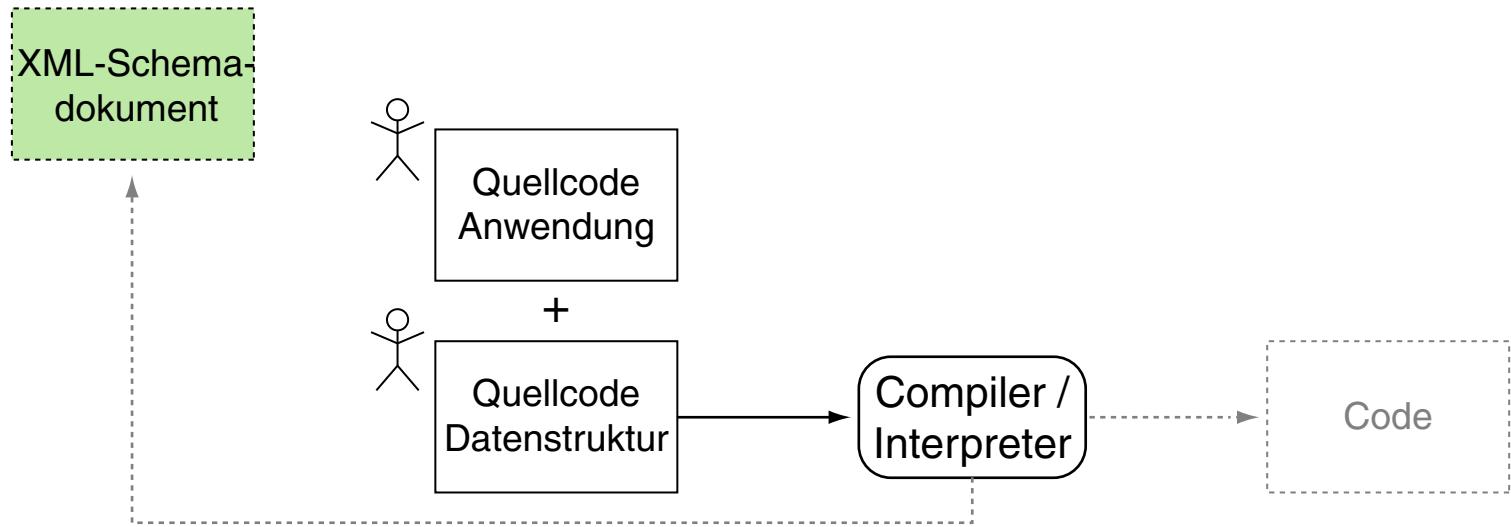
## XML Data Binding: Anwendung der Python-API (Fortsetzung)

<geburstag>-Element suchen und neu setzen:

```
def set_birthday(personen, target_first_name, target_last_name, birthday):
    for person in personen.person:
        name = person.name
        if name.vorname == target_first_name and
           name.nachname == target_last_name:
            print(f'[xsData] Updating "geburstag": \
                  {person.geburstag} -> {birthday}')
            person.geburstag = birthday
    return
```

# Parse-Paradigmen und APIs für XML

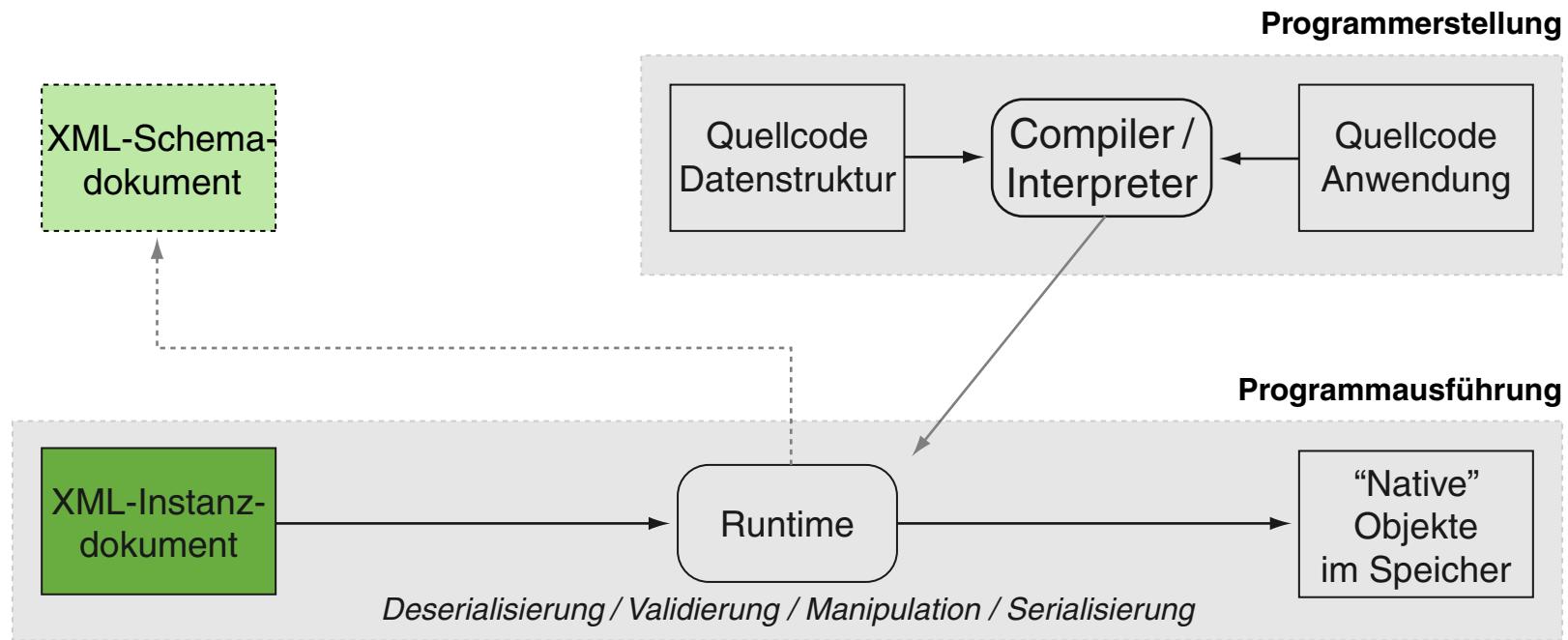
## XML Data Binding: Konzepte II [Konzepte I]



- Manuelle Erstellung von Elementtypklassen.
- An die Stelle des Schema-Compilers tritt der Anwender, der annotierten Quellcode der Datenstrukturen spezifiziert.

# Parse-Paradigmen und APIs für XML

XML Data Binding: Konzepte II (Fortsetzung) [\[Konzepte I\]](#)



- Ein XML-Schema *kann* aus dem Quellcode generiert werden.

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API [Konzepte I]

Java-Klasse für das <personen>-Element:

```
package documentlanguages.xmlparser.jaxb.manual.personen;

import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "personen")
public class PersonenType {

    @XmlElement(name = "person")
    private List<PersonType> personList;

    public List<PersonType> getPersonList() {
        if (personList == null) {personList = new ArrayList<PersonType>(); }
        return this.personList;
    }
}
```

## Bemerkungen (Annotationen) :

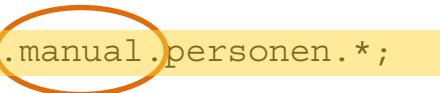
- Die Annotationen dienen dazu, dem Java-Compiler mitzuteilen, wie XML-Elemente vom Typ `<person>` auf den Java-Datentyp `personList` gemappt werden. Sie bilden das gewünschte Inhaltsmodell (hier: mit Kindelementen) durch die Schachtelung nach.
- Java Annotations. “Annotations, a form of metadata, provide data about a program that is not part of the program itself.” [\[Oracle\]](#)
- Vergleiche die manuell erstellte Personenklasse mit der via xjc automatisch generierten Personenklasse `PersonenType.java`.

# Parse-Paradigmen und APIs für XML

## XML Data Binding: Anwendung der Java-API (Fortsetzung) [Konzepte I]

Java-Klasse:

```
package documentlanguages.xmlparser.jaxb;

import java.io.*;
import java.util.List;
import javax.xml.bind.*;
import documentlanguages.xmlparser.jaxb.manual.personen.*;  // This line is highlighted with an orange oval

public class JAXBParserExample1 {

    public PersonenType load(String filename) ...
    public void setBirthday(PersonenType personen, ...
    public void save(PersonenType personen, String filename) ...

    public static void main(String[] args) throws JAXBException, IOException {
        JAXBParserExample1 pe = new JAXBParserExample1();
        PersonenType personen =
            pe.load("./bin/documentlanguages/xmlparser/personen.xml");
        pe.setBirthday(personen, "Judea", "Pearl", "10.10.1949");
        pe.save(personen, "./bin/documentlanguages/xmlparser/personen-neu.xml");
    }
}
```

# Parse-Paradigmen und APIs für XML

## Diskussion der API-Technologien

DOM repräsentiert ein XML-Dokument als einen Objektbaum im Hauptspeicher.

- Das ganze Dokument befindet sich (scheinbar) im Speicher.
- Random-Access und einfache Manipulation von Dokumentbestandteilen.
- Laden und Speichern ist in der DOM-API (ab Level 3) realisiert.
- DOM-Objekte eignen sich nur bedingt als Datenstrukturen einer Applikation.

SAX repräsentiert ein XML-Dokument als einen Strom von Ereignissen.

- Jedes Dokument-Token begegnet einem genau einmal.
- Kein Random-Access auf die Bestandteile eines Dokuments.
- Programmierende sind selbst verantwortlich für die Speicherung.
- Es kann flexibel bei der Speicherung von Inhalten entschieden werden.

# Parse-Paradigmen und APIs für XML

## Diskussion der API-Technologien

DOM repräsentiert ein XML-Dokument als einen Objektbaum im Hauptspeicher.

- Das ganze Dokument befindet sich (scheinbar) im Speicher.
- Random-Access und einfache Manipulation von Dokumentbestandteilen.
- Laden und Speichern ist in der DOM-API (ab Level 3) realisiert.
- DOM-Objekte eignen sich nur bedingt als Datenstrukturen einer Applikation.

SAX repräsentiert ein XML-Dokument als einen Strom von Ereignissen.

- Jedes Dokument-Token begegnet einem genau einmal.
- Kein Random-Access auf die Bestandteile eines Dokuments.
- Programmierende sind selbst verantwortlich für die Speicherung.
- Es kann flexibel bei der Speicherung von Inhalten entschieden werden.

# Parse-Paradigmen und APIs für XML

## Diskussion der API-Technologien (Fortsetzung)

XML Data Binding repräsentiert ein XML-Dokument als entsprechende Klassen.

- Hinsichtlich der Speicherung ist es mit DOM vergleichbar, eher besser.
- Generierung von Datentypen (Klassen) und Zugriffsmethoden für XML-Elemente; die Navigation durch die Baumstruktur entfällt.
- Die generierten Klassen stehen in der Anwendung zur Verfügung.
- Hinsichtlich Speicherplatz und Laufzeit ist der Ansatz optimal.
- Änderung des XML-Schemas erfordert die Neugenerierung der Klassen.

## Bemerkungen (API-Technologien) :

- DOM ist vorzuziehen, falls viele Manipulationen zu machen sind und falls (fremder) Scripting-Code einfachen Zugriff haben soll.
- SAX ist vorzuziehen, falls Effizienz eine Rolle spielt, sehr große Dokumente zu verarbeiten sind, oder falls die Verarbeitung hauptsächlich datenstromorientiert ist.
- DOM und SAX lassen sich in *einer* Applikation kombinieren: ein SAX-Ereignisstrom dient als Eingabe für DOM-Aufrufe.

## Bemerkungen (Java) :

- Die Java API for XML Processing, `javax.xml` (JAXP), stellt die Technologien für DOM, SAX und StAX bereit. [\[Javadoc\]](#)
- Die Java Architecture for XML Binding, `javax.xml.bind` (JAXB), stellt die Data-Binding-Technologien bereit. In den Java-Versionen 7 bis 10 ist JAXB bei Aufrufen von `javac` und `java` mittels `--add-modules java.xml.bind` zu aktivieren.

Seit Java 11 ist JAXB aus dem JDK ausgegliedert. Die JAXB-Bibliotheken sind zusätzlich bereitzustellen (als jar) und dem Classpath hinzu zu fügen [\[Eclipse\]](#), [\[Referenzimp.\]](#) :

```
-cp ".:jaxb-ri/mod/*"
```

- Die aktuelle JAXB-Entwicklung geschieht unter dem Namen “Jakarta XML Binding” im Rahmen von [Jakarta EE](#). [\[Wikipedia\]](#), [\[JEP 320\]](#)

# Parse-Paradigmen und APIs für XML

Quellen zum Nachlernen und Nachschlagen im Web: Konzepte

- W3C. *What is the Document Object Model?*  
[www.w3.org/TR/DOM-Level-3-Core](http://www.w3.org/TR/DOM-Level-3-Core)
- W3C. *DOM FAQ.*  
[www.w3.org/DOM/faq.html](http://www.w3.org/DOM/faq.html)
- W3C. *DOM Living Standard* (Recommendation).  
[dom.spec.whatwg.org](http://dom.spec.whatwg.org)
- W3 Schools. *XML DOM Tutorial.*  
[www.w3schools.com/xml/dom\\_intro.asp](http://www.w3schools.com/xml/dom_intro.asp)

# Parse-Paradigmen und APIs für XML

Quellen zum Nachlernen und Nachschlagen im Web: Programmierung

- Oracle. *Java API for the Document Object Model (DOM)*.  
[docs.oracle.com/en/java/javase/18/docs/api](https://docs.oracle.com/en/java/javase/18/docs/api)
- Oracle. *Java API for XML Processing (JAXP)*.  
[docs.oracle.com/en/java/javase/18/docs/api/java.xml](https://docs.oracle.com/en/java/javase/18/docs/api/java.xml)
- Eclipse. *Jakarta XML Binding (JAXB)*.  
[projects.eclipse.org/projects/ee4j.jaxb-impl](https://projects.eclipse.org/projects/ee4j.jaxb-impl)  
[repo1.maven.org/maven2/com/sun/xml/bind/jaxb-ri](https://repo1.maven.org/maven2/com/sun/xml/bind/jaxb-ri)
- Oracle. *JAXP Tutorial*.  
[docs.oracle.com/javase/tutorial/jaxp](https://docs.oracle.com/javase/tutorial/jaxp)
- Oracle. *Java Generics Tutorial*.  
[docs.oracle.com/javase/tutorial/extras/generics](https://docs.oracle.com/javase/tutorial/extras/generics)