

## III. Dokumentsprachen

- ❑ Auszeichnungssprachen
- ❑ HTML
- ❑ Cascading Stylesheets CSS
- ❑ XML-Grundlagen
- ❑ XML-Schema
- ❑ Die XSL-Familie
- ❑ APIs für XML-Dokumente

# APIs für XML-Dokumente

## 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

*“An application programming interface (API) is a set of routines, protocols, and tools for building software applications. [...] An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other.”* [\[Wikipedia\]](#)

# APIs für XML-Dokumente

## 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

*“An application programming interface (API) is a set of routines, protocols, and tools for building software applications. [...] An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other.”* [\[Wikipedia\]](#)

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

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

# APIs für XML-Dokumente

## DOM: Historie

- 1998 DOM Level 1, Recommendation. What is the DOM? [W3C [REC intro](#)]
- 2004 DOM Level 3 Core, Recommendation. [W3C [REC intro](#)]
- 2015 DOM4, Recommendation. [W3C [REC intro](#), [status](#)]
- 2018 Shadow DOM. [W3C [NOTE](#)] [[github](#)] [[SELFHTML](#)]
- 2020 DOM. Living Standard. [[whatwg](#)]

## Language Bindings:

- 2002 DOM-Implementierung in Java 1.4 (JDK4).
- 2020 DOM-Implementierung in Java 14. [[Javadoc](#)]

## 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 Level 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. Seit DOM Level 3 gehören auch Methoden zum Laden und Speichern zur Spezifikation.
- ❑ [caniuse.com](http://caniuse.com) zeigt die Unterstützung einzelner (DOM-)Features für verschiedene Browser. Beispiel: [getElementsByClassName](#).

# APIs für XML-Dokumente

## 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](#).

# APIs für XML-Dokumente

## 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](#)]



## Bemerkungen (Fortsetzung) :

- ❑ 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.
- ❑ 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.

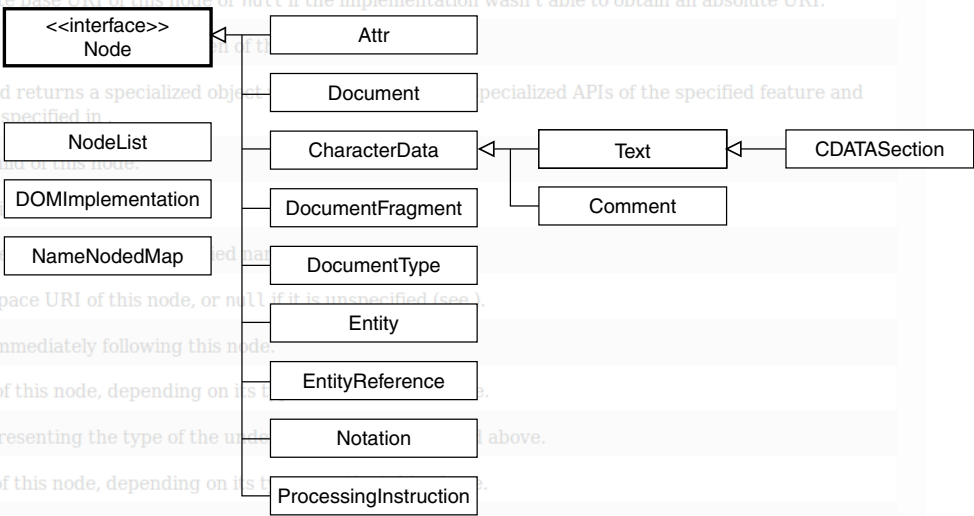
# APIs für XML-Dokumente

## DOM: Funktionen der API

Methoden des `node`-Interfaces im `org.w3c.dom`-Package [Javadoc [node](#), [org.w3c.dom](#)] :

### Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method	Description
Node	<code>appendChild(Node newChild)</code>	Adds the node <code>newChild</code> to the end of the list of children of this node.
Node	<code>cloneNode(boolean deep)</code>	Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
short	<code>compareDocumentPosition(Node other)</code>	Compares the reference node, i.e. the node on which this method is being called, with a node, i.e. the one passed as a parameter, with regard to their position in the document and according to the document order.
NamedNodeMap	<code>getAttributes()</code>	A <code>NamedNodeMap</code> containing the attributes of this node (if it is an <code>Element</code> ) or null otherwise.
String	<code>getBaseURI()</code>	The absolute base URI of this node or null if the implementation wasn't able to obtain an absolute URI.
NodeList	<code>getChildNodes()</code>	A <code>NodeList</code> of the children of this node.
Object	<code>getFeature(String feature, String version)</code>	This method returns a specialized object implementing the specified feature and version, as specified in the DOM specification.
Node	<code>getFirstChild()</code>	The first child of this node.
Node	<code>getLastChild()</code>	The last child of this node.
String	<code>getLocalName()</code>	Returns the local name of this node.
String	<code>getNamespaceURI()</code>	The namespace URI of this node, or null if it is unspecified (e.g., for HTML elements).
Node	<code>getNextSibling()</code>	The node immediately following this node in the document.
String	<code>getNodeName()</code>	The name of this node, depending on its type.
short	<code>getNodeType()</code>	A code representing the type of the underlying node.
String	<code>getNodeValue()</code>	The value of this node, depending on its type.



# APIs für XML-Dokumente

## DOM: Anwendung

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

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

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

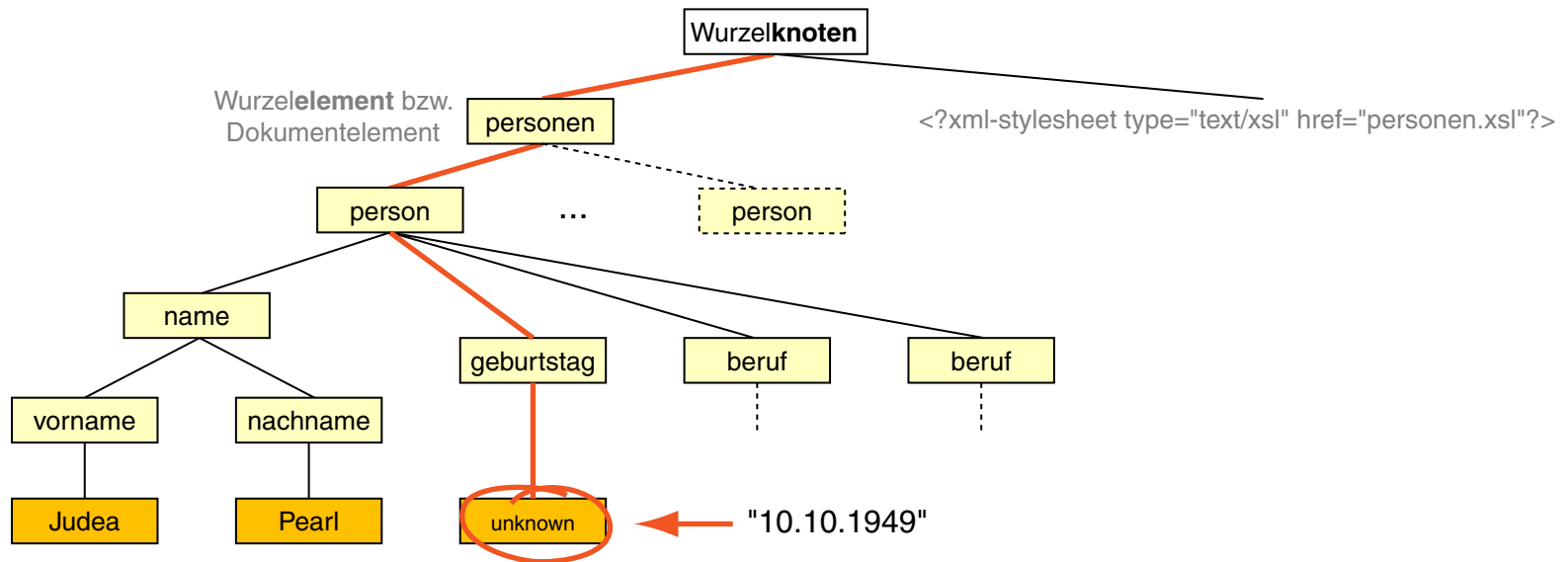
[\[personen.xml\]](#)

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

### Aufgabe:

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



# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

Java-Klasse:

```
package documentlanguages.xmlparser.dom;
```

```
import java.io.*;
```

```
import org.w3c.dom.*;
```

```
import org.xml.sax.*;
```

```
import javax.xml.*;
```

```
public class DomParserExample {
```

```
    public Document load(String filename)...
```

```
    public Node findPerson(Node node, String firstName, String lastName)...
```

```
    private boolean matchesPerson(Node n, String firstName, ...
```

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

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

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

```
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (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");  
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

DOM-Parser instantiieren und Dokument in DOM-Objektmodell einlesen [[SAX](#), [JAXB](#), [Javadoc](#)] :

```
public Document load(String filename)
    throws ParserConfigurationException, IOException, SAXException {

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder docBuilder = dbf.newDocumentBuilder();
    return docBuilder.parse(new File(filename));
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

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 [XPath-Variante](#).



# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

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;
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

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;
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

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;
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

<geburtstag>-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("geburtstag"))
        {
            System.out.println(" [DOM] Updating geburtstag: " +
                personChild.getTextContent()+" -> "+birthday);
            personChild.setTextContent(birthday);
        }
    }
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

<geburtstag>-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("geburtstag");  
    if (birthdayElements.getLength() > 0)  
    {  
        System.out.println("[DOM] Updating geburtstag: " +  
            birthdayElements.item(0).getTextContent() + " -> " + birthday);  
        birthdayElements.item(0).setTextContent(birthday);  
    }  
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

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)));
}
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

Aufruf in der Shell:

```
[user@pc WORKINGDIR]$ java -cp CLASSPATH  
documentlanguages.xmlparser.dom.DomParserExample  
  
[DOM] Updating geburtstag: unknown -> 10.10.1949
```

# APIs für XML-Dokumente

## DOM: Anwendung (Fortsetzung)

Navigation im [Dokumentbaum](#). Direkter Zugriff mit `xpath`-Interface [Javadoc [Package](#), [Interface](#)] :

```
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 [DFS-Variante](#).



# APIs für XML-Dokumente

## SAX: Historie

- 1997 Unter der Koordination von [David Megginson](#) entwickeln Teilnehmer der XML-DEV Mailing List einen einfachen, effizienten Parser.
- 2004 SAX 2.0.2. [\[saxproject.org\]](http://saxproject.org)
- 2020 SAX2-Implementierung in Java 14. [\[Javadoc\]](#)

## Bemerkungen:

- ❑ Die Simple API for XML, SAX, entstand aus dem Wunsch, die Entwicklung von Programmen zur Verarbeitung von XML-Dokumenten (XML-Prozessoren) zu vereinfachen.
- ❑ 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.
- ❑ Ursprünglich entstand die SAX-API aus einer Sammlung generischer Java-Schnittstellen für XML-Parser. Inzwischen hat sie sich als eigenständige Möglichkeit zur Verarbeitung von XML-Dokumenten in verschiedenen Hochsprachen entwickelt. Neben den Umsetzungen für Java existieren auch Implementierungen für C++, Python, Perl und Eiffel.

# APIs für XML-Dokumente

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

Verwendung eines SAX-Parsers in folgenden Schritten:

1. Implementierung und Einbindung eines **Content-Handlers**.
2. Instantiierung einer spezifischen Parser-Ausprägung (via Factory-Pattern).
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**

## Bemerkungen:

- ❑ Die SAX-API impliziert keine spezielle Datenstruktur. Deshalb ist der Ansatz mit nur geringen Modifikationen auf viele Programmiersprachen übertragbar.
- ❑ 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.

# APIs für XML-Dokumente

## SAX: Funktionen der API

Methoden (Callback-Funktionen) der **Content-Handler**-Schnittstelle [\[Javadoc\]](#) :

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

# APIs für XML-Dokumente

## SAX: Anwendung [\[Callback-Funktionen\]](#)

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

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

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

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

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

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument()
<?xml-stylesheet type="text/xsl" href="perso.xml" ?>    → processingInstruction()

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

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



# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument()
<?xml-stylesheet type="text/xsl" href="perso.xml" ?>    processingInstruction()

<personen>                                              startElement()
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument()
<?xml-stylesheet type="text/xsl" href="perso.xml"?>    processingInstruction()

<personen>                                              startElement()
  <person>                                              startElement()
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

```
<?xml version="1.0" ?>                                → startDocument()
<?xml-stylesheet type="text/xsl" href="perso.xml" ?>    processingInstruction()

<personen>                                              startElement()
  <person>                                              startElement()
    <name>                                              startElement()
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

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

# APIs für XML-Dokumente

## SAX: Anwendung [\[Callback-Funktionen\]](#)

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

# APIs für XML-Dokumente

## SAX: Anwendung [Callback-Funktionen]

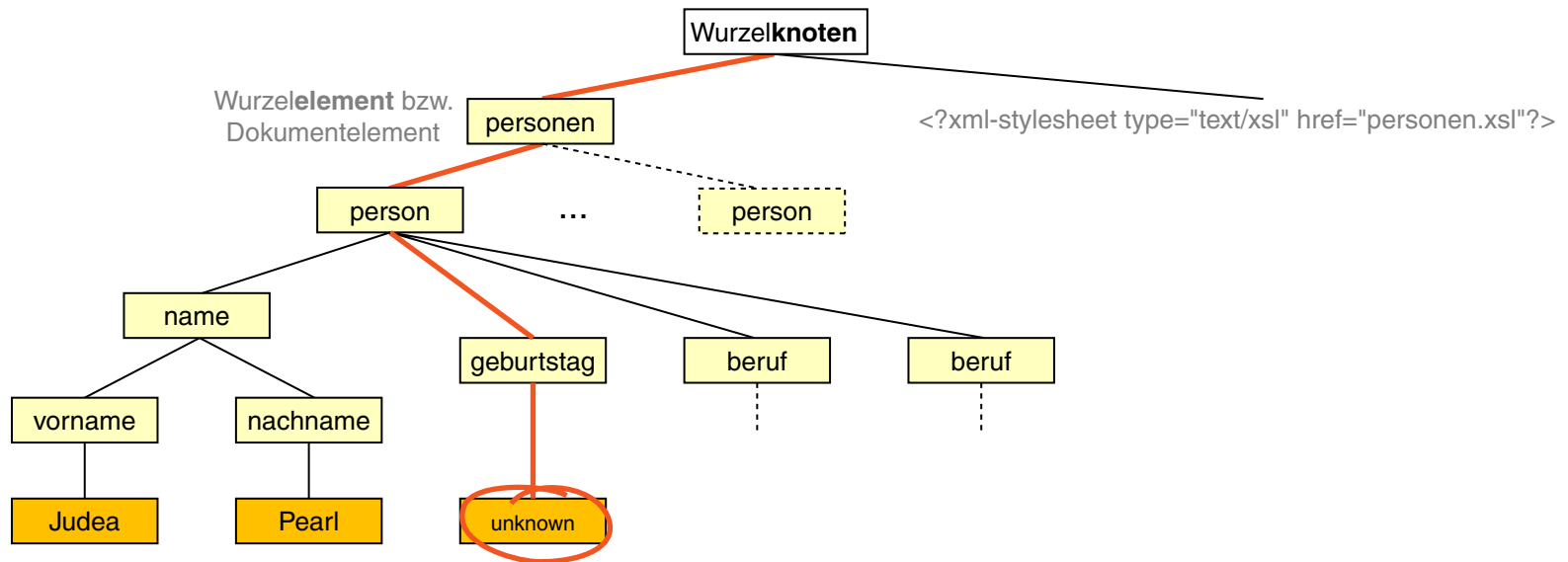
<code>&lt;?xml version="1.0" ?&gt;</code>	<code>→ startDocument()</code>
<code>&lt;?xml-stylesheet type="text/xsl" href="perso</code>	<code>processingInstruction()</code>
<code>&lt;personen&gt;</code>	<code>startElement()</code>
<code>&lt;person&gt;</code>	<code>startElement()</code>
<code>&lt;name&gt;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

# APIs für XML-Dokumente

## SAX: Anwendung (Fortsetzung)

### Aufgabe:

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



# APIs für XML-Dokumente

## SAX: Anwendung (Fortsetzung)

SAX-Parser instantiieren und XML-Ereignisstrom öffnen [[DOM](#), [JAXB](#), [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);
    }
}
```



# APIs für XML-Dokumente

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

Schnittstellenklasse mit eigenen Callback-Funktionen:

```
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.

# APIs für XML-Dokumente

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

Verarbeitung von Elementstart-Ereignissen (= Zustandsmarkierung):

```
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;}

}
```

# APIs für XML-Dokumente

## SAX: Anwendung (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;  
    }  
}
```

# APIs für XML-Dokumente

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

Verarbeitung von Elementende-Ereignissen:

```
public void endElement(String uri, String localName, String qName){

    if (qName.equals("person"))
    // When leaving a <person>-element...
    {
        // 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;
    }
}
```

# APIs für XML-Dokumente

## SAX: Anwendung (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.

# APIs für XML-Dokumente

## XML Data Binding: Historie

- 2006 JSR 222. Java Specification Request für JAXB 2.0, der Java Architecture for XML Binding. [[jcp.org](http://jcp.org)]
- 2013 JAXB-Referenzimplementierung. [[Oracle](http://oracle.com)]
- 2013 Castor. Open Source Data Binding Framework in Java. [[Castor](http://castor.xml-cast.org)]
- 2013 EclipseLink MOXy. XML Binding with JAXB and [SDO](#). [Eclipse [MOXy](#), [SDO](#)]
- 2018 JAXB-Implementierung in Java 10. [[JavaDoc](http://java-docs.oracle.com)]
- 2018 JAXB-Referenzimplementierung unter [EE4J](#). [[Github](https://github.com)]



## Bemerkungen:

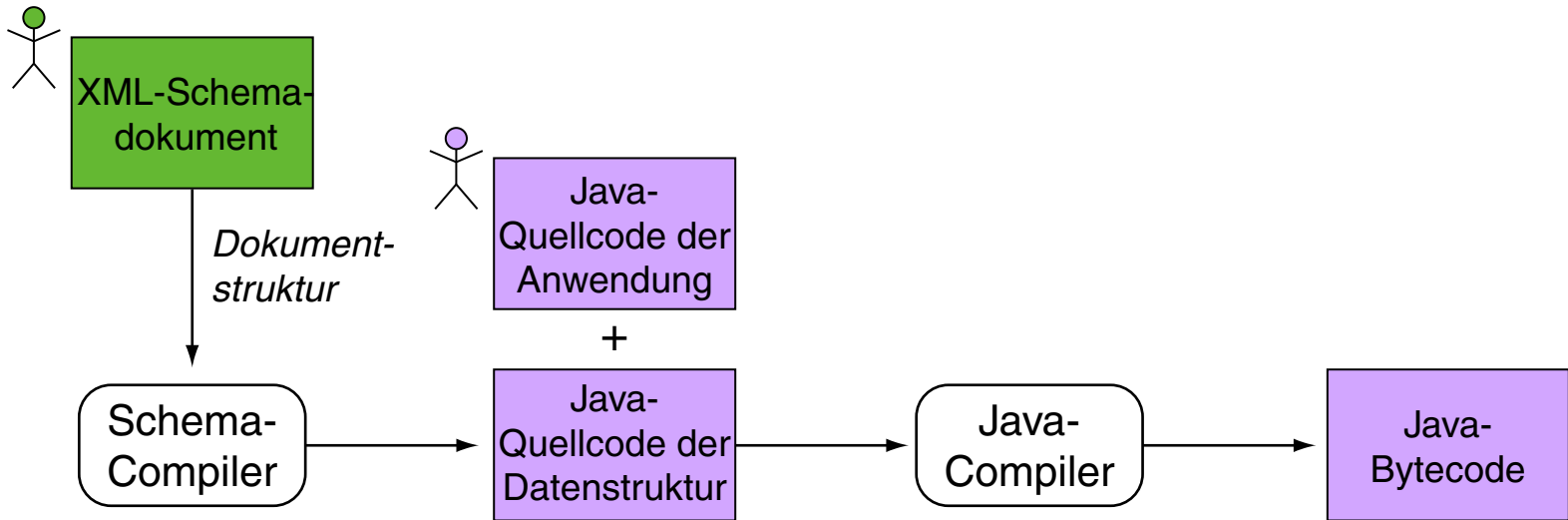
- ❑ Unter Data Binding versteht man die Abbildung eines Datenmodells (hier: XML-Schema) auf die Konzepte einer Zielsprache. Data Binding macht die Daten auf *Basis der Datenstrukturen der gewählten Zielsprache* verfügbar.
- ❑ Data Binding wird attraktiv, wenn Mechanismen für dessen Automatisierung existieren: die Konzepte der Zielsprache (Typ-Constraints, Datenstruktur-Mapping, Setter / Getter-Methoden, etc.) werden aus Sicht des Programmierers transparent gehandelt.
- ❑ JSR 31. XML Data Binding: “A facility for compiling an XML schema into one or more Java classes which can parse, generate, and validate documents that follow the schema.” [[jcp.org](http://jcp.org)]
- ❑ 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.

Unmarshalling an XML document with the appropriate JAXB method 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.” [[Oracle](http://Oracle), [GlassFish](http://GlassFish)]

# APIs für XML-Dokumente

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

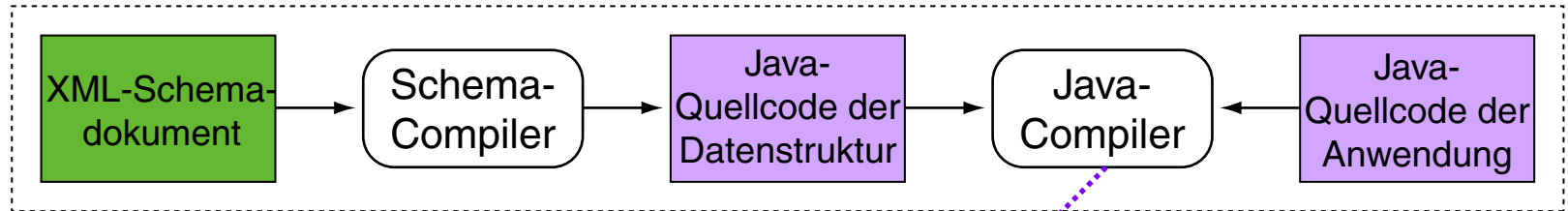


- ❑ Ein Schema-Compiler erzeugt für die Datentypen im XML-Schema entsprechende Java-Klassen, mit denen sich die Elemente von schemavaliden XML-Dokumenten in einem Java-Programm erzeugen und manipulieren lassen. [[Oracle](#)]
- ❑ Die eigene Anwendung setzt direkt auf den erzeugten Java-Klassen auf.

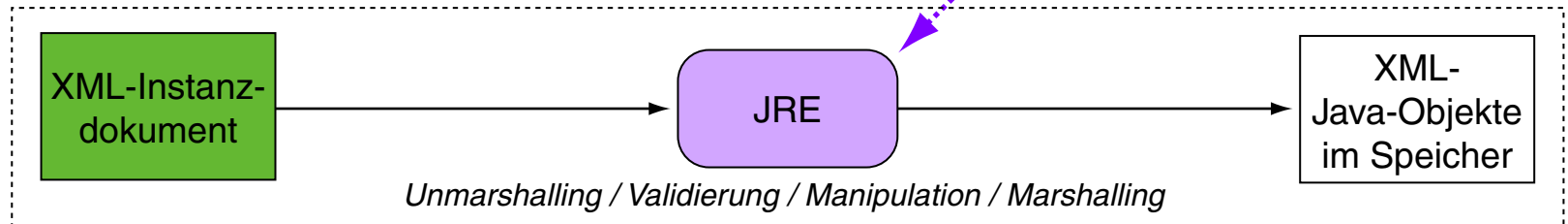
# APIs für XML-Dokumente

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

### Programmerstellung



### Programmausführung



- ❑ XML-Schema für Programmerstellung, Instanzdokument für Programmausführung.
- ❑ Unmarshalling: Lese- und Validierungsvorgang, der eine XML-Datei aus einem Eingabestrom liest und die notwendigen Speicherobjekte erzeugt.
- ❑ Marshalling: Schreiben der Speicherobjekte als XML-Datei.

# APIs für XML-Dokumente

## XML Data Binding: Konzepte I (Fortsetzung)

### 1. xjc.

XML-Schema-Compiler. Erzeugt für die Datentypen im XML-Schema entsprechende Java-Klassen.

### 2. JAXBContext.

Erzeugung einer Factory-Klasse für folgende Klassen:

- (a) `unmarshaller`      Bildet ein XML-Instanzdokument mit Hilfe von Java-Objekten (Instanzen der von `xjc` erzeugten Klassen) im Speicher ab.
- (b) `marshaller`      Serialisiert die gespeicherte Objektstruktur als XML-Datei.

### 3. Getter- und Setter-Methoden.

Manipulation von Element- und Attributwerten.

# APIs für XML-Dokumente

## XML Data Binding: Anwendung

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

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

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

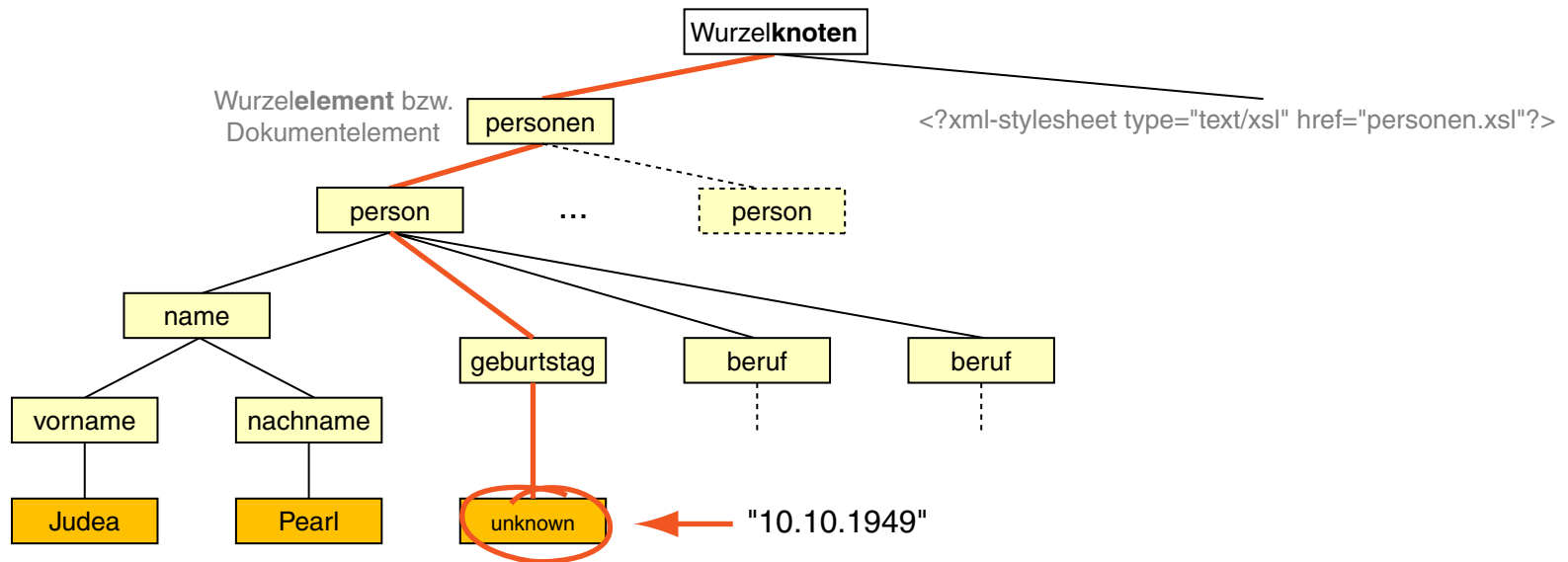
[\[personen.xml\]](#)

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

### Aufgabe:

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



# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

```
<?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>
```

[\[personen.xsd\]](#)

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

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

1. Ort der Schema-Datei für personen.xml:

*SOURCEDIR*/documentlanguages/xmlparser/personen.xsd



# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

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

1. Ort der Schema-Datei für personen.xml:

*SOURCEDIR*/documentlanguages/xmlparser/personen.xsd

2. `xjc`-Aufruf mit Option `-p` derart, dass die neuen Java-Klassen im richtigen Package landen:

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

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

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

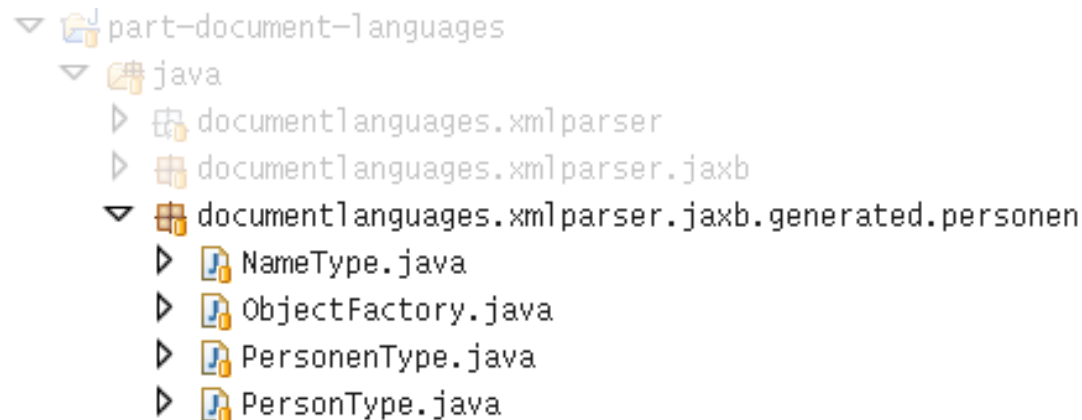
1. Ort der Schema-Datei für personen.xml:

*SOURCEDIR*/documentlanguages/xmlparser/personen.xsd

2. `xjc`-Aufruf mit Option `-p` derart, dass die neuen Java-Klassen im richtigen Package landen:

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

3. Entstandene Package-Struktur [[NameType.java](#), [PersonenType.java](#), [PersonType.java](#)] :



# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

Java-Klasse [\[Konzepte II\]](#) :

```
package documentlanguages.xmlparser.jaxb;

import java.io.*;
import java.util.*;
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");
    }
}
```

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung)

<personen>-Parser instantiieren, Dokument parsen und <personen>-Element im Speicher als Java-Objekt anlegen [[DOM](#), [SAX](#), [Javadoc](#)] :

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

    // Create JAXBContext.
    JAXBContext jc = JAXBContext.newInstance(PersonType.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.
    PersonType personen = u.unmarshal(new StreamSource(filename),
                                      PersonType.class).getValue();

    return personen;
}
```

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (Fortsetzung) [[Dokumentbaum](#)]

<geburtstag>-Element suchen und neu setzen:

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

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

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (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));

}
```

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (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: 14
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
```

# APIs für XML-Dokumente

## XML Data Binding: Anwendung (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: 14**

**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
```



# APIs für XML-Dokumente

## XML Data Binding: Anwendung (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: 14**

**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**

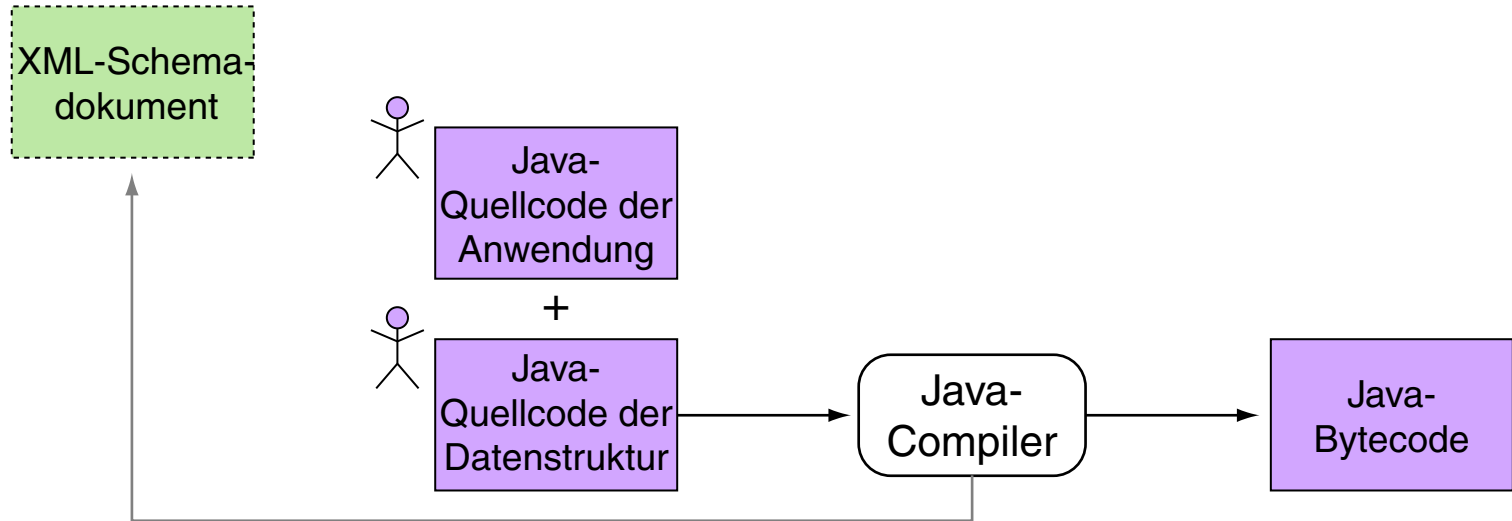
## Bemerkungen:

- ❑ In den Java-Versionen 7 bis 10 wurde JAXB noch zusammen mit dem JDK installiert und ist bei Aufrufen von `javac` und `java` mittels `--add-modules java.xml.bind` zu aktivieren.
- ❑ Ab Java-Version 11 wurde JAXB aus dem JDK ausgegliedert, um es unabhängig weiterzuentwickeln. Die JAXB-Bibliotheken sind zusätzlich bereitzustellen (als jar) und dem Classpath hinzu zu fügen (`-cp ".:jaxb-ri/mod/*"`).

Diese Entwicklung ist als [JDK Enhancement Proposal](#) (JEP) dokumentiert. [JEP [320](#)]

# APIs für XML-Dokumente

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



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

# APIs für XML-Dokumente

## XML Data Binding: Konzepte II (Fortsetzung)

Java-Klasse für das `<personen>`-Element [Konzepte I [xjc](#), [Java main](#)] :

```
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 PersonType {

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

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

# APIs für XML-Dokumente

## XML Data Binding: Konzepte II (Fortsetzung)

Java-Klasse [\[Konzepte I\]](#) :

```
package documentlanguages.xmlparser.jaxb;

import java.io.*;
import java.util.*;
import javax.xml.bind.*;

import documentlanguages.xmlparser.jaxb.manual.personen.*;

public class JAXBParserExample1 {

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

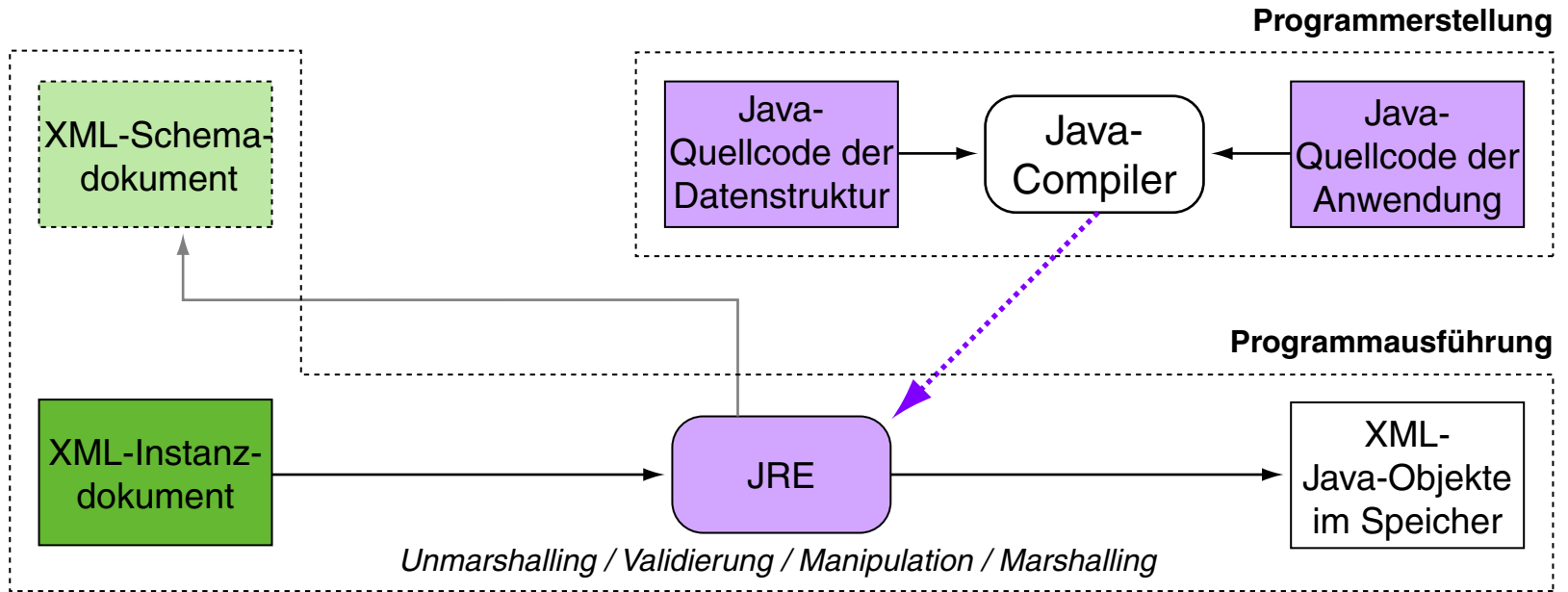
    public static void main(String[] args) throws JAXBException, IOException {
        JAXBParserExample1 pe = new JAXBParserExample1();
        PersonenType personen =
            pe.load("./bin/documentlanguages/xmlparser/personen.xml");
        pe.setBirthday(personen, "Judea", "Pearl", "10.10.1949");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

## Bemerkungen:

- ❑ Verwendung von Java Annotations: “Annotations, a form of metadata, provide data about a program that is not part of the program itself.” [\[Oracle\]](#)
- ❑ Hier dienen die Annotationen dazu, dem Java-Compiler mitzuteilen, dass XML-Elemente vom Typ `<person>` auf den Java-Datentyp `personenListe` gemappt werden. Vergleiche die [manuell erstellte Personenklasse](#) mit der via xjc automatisch generierten Personenklasse [PersonenType.java](#).
- ❑ Abstraktion über Datentypen mit Hilfe von Java Generics: “Generics allow you to abstract over types. The most common examples are container types, such as those in the Collections hierarchy.” [\[Oracle\]](#)

# APIs für XML-Dokumente

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



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

# APIs für XML-Dokumente

## Diskussion der API-Technologien

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

- Das gesamte Dokument befindet sich (scheinbar) im Speicher.
- + Random-Access und einfache Manipulation von Dokumentbestandteilen
- + Laden und Speichern ist in der API (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.
- Programmierer ist selbst verantwortlich für die Speicherung
- + Es kann flexibel bei der Speicherung von Inhalten entschieden werden.



# APIs für XML-Dokumente

## Diskussion der API-Technologien

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

- Das gesamte Dokument befindet sich (scheinbar) im Speicher.
- + Random-Access und einfache Manipulation von Dokumentbestandteilen
- + Laden und Speichern ist in der API (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.
- Programmierer ist selbst verantwortlich für die Speicherung
- + Es kann flexibel bei der Speicherung von Inhalten entschieden werden.

# APIs für XML-Dokumente

## Diskussion der API-Technologien (Fortsetzung)

XML Data Binding repräsentiert ein XML-Dokument als eine Menge von Klassen zum Dokument-Handling.

- 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 sind direkt in der Applikation verwendbar.
- + Hinsichtlich Speicherplatz und Laufzeit ist der Ansatz optimal.
- Änderung des XML-Schemas erfordert die Neugenerierung der Klassen.

## Bemerkungen:

- ❑ DOM ist vorzuziehen, falls viele Manipulationen zu machen sind und falls (fremder) Scripting-Code einfachen Zugriff haben soll. Stichwort: Browser
- ❑ SAX ist vorzuziehen, falls Effizienz eine Rolle spielt oder falls die Verarbeitung hauptsächlich datenstromorientiert ist.
- ❑ DOM und SAX lassen sich in *einer* Applikation kombinieren: ein SAX-Ereignisstrom kann als Eingabe für DOM genutzt werden.
- ❑ Die Java API for XML Processing, JAXP, stellt alle notwendigen Technologien für DOM, SAX und StAX bereit. [\[Oracle\]](#)
- ❑ Die Java Architecture for XML Binding, JAXB, stellt die Data-Binding-Technologien bereit. [\[Oracle\]](#)

# APIs für XML-Dokumente

## Quellen zum Nachlernen und Nachschlagen im Web: Konzepte

- ❑ W3C. *What is the Document Object Model?*  
[www.w3.org/TR/REC-DOM-Level-1/introduction.html](http://www.w3.org/TR/REC-DOM-Level-1/introduction.html)
- ❑ W3C. *DOM FAQ*.  
[www.w3.org/DOM/faq.html](http://www.w3.org/DOM/faq.html)
- ❑ W3C. *DOM4 Recommendation*.  
[www.w3.org/TR/dom](http://www.w3.org/TR/dom)
- ❑ W3 Schools. *XML DOM Tutorial*  
[www.w3schools.com/xml/dom\\_intro.asp](http://www.w3schools.com/xml/dom_intro.asp)

# APIs für XML-Dokumente

## Quellen zum Nachlernen und Nachschlagen im Web: Anwendung

- ❑ Oracle. *Java API for XML Processing, JAXP*.  
[docs.oracle.com/javase/tutorial/jaxp](https://docs.oracle.com/javase/tutorial/jaxp)
- ❑ Oracle. *Java API für das Document Object Model, DOM*.  
[docs.oracle.com/en/java/javase/14/docs/api](https://docs.oracle.com/en/java/javase/14/docs/api)
- ❑ Oracle. *Java API für SAX2-Transformationen*.  
[docs.oracle.com/en/java/javase/14/docs/api](https://docs.oracle.com/en/java/javase/14/docs/api)
- ❑ GlassFish. *Metro Web Service Stack*.  
[javaee.github.io/metro/](https://javaee.github.io/metro/)
- ❑ EE4J. *Java Architecture for XML Binding, JAXB*.  
[github.com/eclipse-ee4j/jaxb-ri](https://github.com/eclipse-ee4j/jaxb-ri)
- ❑ EE4J. *Java API for XML Web Services, JAX-WS*.  
[github.com/eclipse-ee4j/metro-jax-ws](https://github.com/eclipse-ee4j/metro-jax-ws)