XML - Datei Zugriff mit Java

XML in Java

<?xml?>

Inhalte

- XML-Dateien für Experimente
- SAX (Simple API fort XML)
- STAX (Streaming API fort XML)
- Übung: SAX - STAX
- DOM
- JDOM und XPath
- Exkurs: DAO-Pattern
- Übung: DAO-Pattern gelöst mit JDOM und XPath

XML-Datei für Experimente

<?xml?>

party.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<party datum="31.12.13">
    <gast name="Albert Angsthase">
        <getraenk>Wein</getraenk>
        <getraenk>Bier</getraenk>
        <zustand ledig="true" nuechtern="false"/>
    </gast>
    <gast name="Martina Mutig">
        <getraenk>Apfelsaft</getraenk>
        <zustand ledig="true" nuechtern="true"/>
    </gast>
    <gast name="Zacharias Zottelig">
    </gast>
</party>
```

XML-Datei für Experimente

<?xml?>

employees.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<employees>
	<employee id="1">
		<age>29</age>
		<name>Pankaj</name>
		<gender>Male</gender>
		<role>Java Developer</role>
	</employee>
	<employee id="2">
		<age>35</age>
		<name>Lisa</name>
		<gender>Female</gender>
		<role>CEO</role>
	</employee>                    ....
</employees>
```

Fachklasse für Experimente

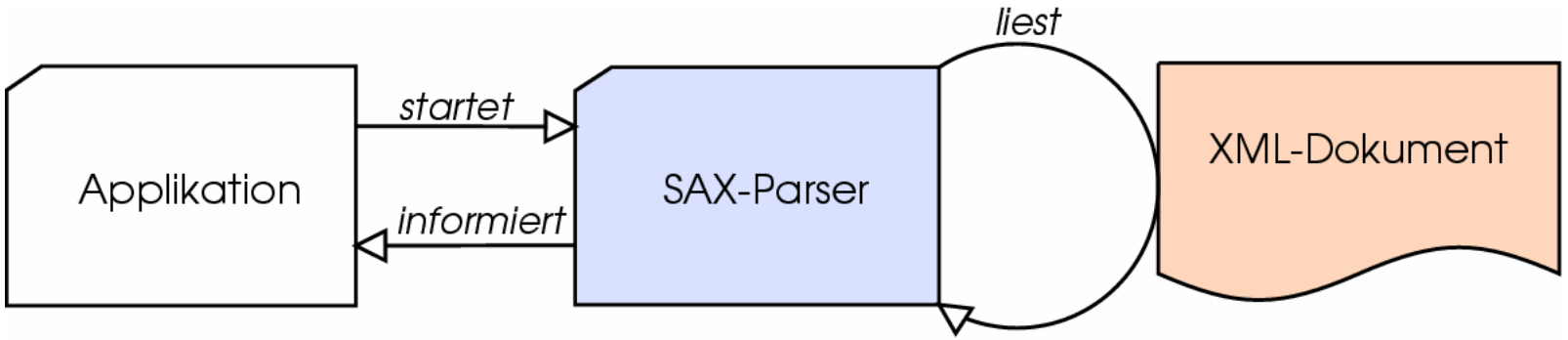
<?xml?>

mitarbeiter.java

```
package fachklassen;
public class Mitarbeiter {

    private int id;
    private String name;
    private String geschlecht;
    private int alter;
    private String funktion;

    // Konstruktoren Default und mit allen Attributen als Parameter

    // diverse setter und getter

    @Override
    public String toString() {
        return "Mitarbeiter: ID="+this.id+" Name=" + this.name +
               " Alter=" + this.alter + " Geschlecht=" +
               this.geschlecht + " Funktion=" + this.funktion;
    }
}
```

Ansatz

- Sequentielle Abarbeitung des Dokuments
  Der Parse-Baum wird nicht aufgebaut.
  → kaum Speicher wird verbraucht.
- SAX kann auch sehr große Dokumente parsen.
- Event-basiertes Framework.
  Während der Parser das File analysiert, ruft dieser die vom Anwendungsentwickler implementierten Callback-Methoden auf.



Beispiel: Mitarbeiterverwaltung mit SAX

```
public class MitarbeiterSaxParser {
    public static void main(String[] args) throws Exception {
        SAXParserFactory parserFactory = SAXParserFactory.newInstance();
        SAXParser saxParser = parserFactory.newSAXParser();
        SAXHandler handler = new SAXHandler();
        saxParser.parse(new File("employees.xml"), handler);
       // saxParser.parse(ClassLoader.getSystemResourceAsStream(
                          "xml/employees.xml"), handler);
    }
}
```

- SAXParser und SAXParserFactory befinden sich im Paket java.xml.parsers
- Der SAXParser wird per Factory erzeugt und ist der eigentliche Parser. Er verfügt über die wichtige Methode parse(). Während die XML-Datei geladen und durchgegangen wird, werden die Methoden des SAXHandlers aufgerufen.

Beispiel: Mitarbeiterverwaltung mit SAX

```
class SAXHandler extends DefaultHandler {

    private List<Mitarbeiter> empList = null;   // Arrayliste für alle Mitarbeiter
    private Mitarbeiter currEmp = null;         // aktuell gelesener Mitarbeiter
    private String content = null;              // für den Textinhalt von Elementknoten

    @Override
    public void startDocument() throws SAXException {
        // wird vom Parser zu Beginn des Lesens aufgerufen
    }

    @Override
    public void endDocument() throws SAXException {
        // Ende des Dokuments wurde erreicht
        // Mitarbeiterliste aus der XML-Datei ausgeben
        for (Mitarbeiter emp : empList) {
            System.out.println(emp);
        }
    }
```

Beispiel: Mitarbeiterverwaltung mit SAX

```
@Override
// Wird aufgerufen bei jedem Start-Tag
// wichtige Parameter: der Elementname und seine Attribute
public void startElement(String uri, String localName,
        String elementName, Attributes attributes) throws SAXException {
    if (elementName.equals("employees")) { // alle Mitarbeitern werden
        empList = new ArrayList<>();       // unterhalb von employees verwaltet
    } else if (elementName.equals("employee")) {
        currEmp = new Mitarbeiter();       // neuer Mitarbeiter in der XML-Datei
        currEmp.setId(Integer.parseInt(attributes.getValue("id")));
    }                                      // id kann als Attribut gelesen werden
}

@Override
public void characters(char[] ch, int start, int length) throws SAXException {
    // Die Inhalte zwischen Start-Tag und Ende-Tag auslesen
    content = String.copyValueOf(ch, start, length).trim();
}
```

Beispiel: Mitarbeiterverwaltung mit SAX

```
    @Override
    public void endElement(String uri, String localName,
            String elementName) throws SAXException {
        if (elementName.equals("employee")) {
            // Füge den neuen Mitarbeiter zur Liste hinzu
            empList.add(currEmp);
            // Alle anderen Tags dienen zum Setzen der Attribute
        } else if (elementName.equals("age")) {
            currEmp.setAlter(Integer.parseInt(content));
        } else if (elementName.equals("name")) {
            currEmp.setName(content);
        } else if (elementName.equals("gender")) {
            currEmp.setGeschlecht(content);
        } else if (elementName.equals("role")) {
            currEmp.setRole(content);
        }
    }
}
```

Wichtiges zum DefaultHandler

- befindet sich im Paket java.xml.sax.helpers
- Implementiert vier verschiedene Handler-Interfaces
  `EntityResolver, DTDHandler, ContentHandler, ErrorHandler`
- Alle Methoden zu den Interfaces werden in der Klasse `DefaultHandler` als leere Methoden { } angeboten.
- Die wichtigsten Methoden zum Parsen des Dokuments werden durch das Interface `ContentHandler` deklariert.
  z.B startDocument(), endDocument(), startElement(...), endElement(...), characters(…)
- Der „eigene“ `SAXHandler` erbt von `DefaultHandler` und überschreibt die relevanten Methoden sinnvoll.

STAX (Streaming API for XML)

<?xml?>

Ansatz

- Sehr effizienter Zugriff da auch hier der XML-Baum nicht im Speicher hinterlegt wird. Vor allem mit der Cursor-API
- Bietet API zum Lesen und Schreiben
- Der STAX-Parser muss aktiv von Komponente zu Komponente bewegt werden (pull-Style).
- Bei der Cursor-Verarbeitung wird die Komponente direkt mit dem Parser verarbeitet. Dies sind z.B. `START_ELEMENT`, `CHARACTERS`, `END_ELEMENT`, `START_DOCUMENT`, `END_DOCUMENT`
- Beim Iterator Zugriff werden XML-Event-Objekte geliefert.
- `XMLInputFactory`, `XMLStreamConstants`, `XMLStreamException` und `XMLStreamReader` befinden sich im Package `javax.xml.stream`

Beispiel: Mitarbeiterverwaltung mit STAX

```
public class MitarbeiterStaxParser {

  public static void main(String[] args)
              throws XMLStreamException, FileNotFoundException {
    List<Mitarbeiter> empList = null;   // Arrayliste für alle Mitarbeiter
    Mitarbeiter currEmp = null;         // aktuell gelesener Mitarbeiter
    String content = null;              // für den Textinhalt von Elementknoten
    String elementName;                 // für den Element-Namen
    XMLInputFactory xmlInputFactory = XMLInputFactory.newInstance();
    XMLStreamReader xmlStreamReader =
            xmlInputFactory.createXMLStreamReader(
                new FileInputStream("employees.xml"));
```

- Erzeuge den passenden XML-Parser mit der **`XMLInputFactory`**.
  Wähle **`XMLStreamReader`** für die Cursor-Verarbeitung
  oder `XMLEventReader` für die Iterator-Verarbeitung

Beispiel: Mitarbeiterverwaltung mit STAX

```
while (xmlStreamReader.hasNext()) {
    int event = xmlStreamReader.next();
    switch (event) {
        case XMLStreamConstants.START_ELEMENT:
            elementName = xmlStreamReader.getLocalName();
            if (elementName.equals("employees")) {
                empList = new ArrayList<>();
            } else if (elementName.equals("employee")) {
                currEmp = new Mitarbeiter();
                currEmp.setId(Integer.parseInt(
                        xmlStreamReader.getAttributeValue(0)));
            }
            break;
        case XMLStreamConstants.CHARACTERS:
            content = xmlStreamReader.getText().trim();
            break;
```

Beispiel: Mitarbeiterverwaltung mit STAX

```
case XMLStreamConstants.END_ELEMENT:
    elementName = xmlStreamReader.getLocalName();
    if (elementName.equals("employee")) {
        // Füge den neuen Mitarbeiter zur Liste hinzu
        empList.add(currEmp);
        // Alle anderen Tags dienen zum Setzen der Attribute
    } else if (elementName.equals("age")) {
        currEmp.setAlter(Integer.parseInt(content));
    } else if (elementName.equals("name")) {
        currEmp.setName(content);
    } else if (elementName.equals("gender")) {
        currEmp.setGeschlecht(content);
    } else if (elementName.equals("role")) {
        currEmp.setRole(content);
    }
    break;
```

Beispiel: Mitarbeiterverwaltung mit STAX

```
        case XMLStreamConstants.START_DOCUMENT:
            //
            break;
        case XMLStreamConstants.END_DOCUMENT:
            // Daten von allen Mitarbeitern ausgeben
            for (Mitarbeiter emp : empList) {
                System.out.println(emp);
            }
            break;
    }
  }
 }
}
```

Übung zu SAX - STAX

<?xml?>

- Greifen Sie auf die XML-Datei party.xml per SAX oder STAX zu.
- Geben Sie alle relevanten Daten aus.

Mögliche Bildschirmausgabe:

```
Datum der Party: 31.12.01
Neuer Gast: Albert Angsthase
trinkt: Wein
trinkt: Bier
ist ledig: true und nuechtern: false
Neuer Gast: Martina Mutig
trinkt: Apfelsaft
ist ledig: true und nuechtern: true
Neuer Gast: Zacharias Zottelig
```

Ansatz

- Erstellung eines baumförmigen Modells der Dokumentstruktur (XML oder HTML)
- Auswertung und Manipulation dieser Struktur über Methoden der Objekte im Baum
- Sammlung von Klassen („Interfaces“), die im Baum verwendet werden

DOM (Document Object Model)

<?xml?>

Vorteile

- einheitlicher Zugriff in verschiedenen Sprachen
- gezielter und schneller Zugriff
- geringerer Programmieraufwand als bei SAX
- besser geeignet zur Generierung dynamischer Inhalte

Nachteile

- zunächst hoher Overhead für die Baumerstellung
- automatisches Durchschreiten des Baumes nicht enthalten
- Schreiben von XML nicht enthalten

Beispiele

- MitarbeiterDomParser.java
- PartyDomParser.java

JDOM (Java Document Object Model)

<?xml?>

- Open-Source API
- Aktuell jdom-2.0.5.jar oft in Kombination mit jaxen-1.1.6.jar für XPath
- Speziell auf Java zugeschnittenes Objekt-Modell für XML
  Dieses vereint die Stärken von SAX und DOM
- Schreiben von XML möglich
- XML-Dokument wird vollständig im Speicher abgebildet
  → Hohe Anforderung an Ressourcen
- kann auf vorhandenen SAX und DOM Parsern aufsetzen

JDOM (Java Document Object Model)

<?xml?>

- Erzeugen eines JDOM-Dokuments auf Basis der Datei party.xml. Diese nutzt den SAXBuilder als Basis. Document aus org.jdom2

```
SAXBuilder builder = new SAXBuilder();
Document xmlDocument = builder.build(new File("party.xml"));
```

- Wurzelelement aus dem JDOM-Dokuments **xmlDocument** bestimmen

```
Element party = xmlDocument.getRootElement();
```

- Mögliche Exception‘s abfangen

```
try{
 //…
} catch (JDOMException | IOException ex) {
 // ex.printStackTrace();
}
```

Elemente und Attribute auslesen

```
// Wann fand die Party statt?
System.out.println(party.getAttribute("datum").getValue());

// Wie heißt der erste Gast?
Element albert = party.getChild("gast");
System.out.println(albert.getAttribute("name").getValue());

// Was hat dieser am Anfang getrunken
Element albertGetraenk = party.getChild("gast").
                         getChild("getraenk");
System.out.println(albertGetraenk.getText());

// Das Ganze in einer Zeile
System.out.println(party.getChild("gast").
                         getChildText("getraenk"));
```

Elemente und Attribute auslesen

```
// Ist der erste Gast noch nüchtern?
System.out.println("nüchtern: " + albert.getChild("zustand").
                    getAttribute("nuechtern").getBooleanValue());

// Was hat Albert alles auf der Party getrunken?
List<Element> albertGetraenke = albert.getChildren("getraenk");
for (Element e : albertGetraenke) {
    System.out.println(e.getText());
}

// Alle Attribute von Albert auslesen
List<Attribute> attributes = albert.getAttributes();
for(Attribute attr : attributes){
    System.out.println(attr.getName() + " " + attr.getValue());
}
```

Elemente löschen und hinzufügen

```
// Das erste Getränk von Albert wurde falsch erfasst. Er trank Wasser
albert.removeChild("getraenk");
Element wasser = new Element("getraenk");
wasser.addContent("Wasser");
albert.addContent(wasser);

// Zacharias Zottelig trinkt jetzt auch ein Glas Wein
Element wein = new Element("getraenk");
wein.addContent("Wein");
List<Element> gaeste = party.getChildren("gast");
for (Element gast : gaeste) {
   if (gast.getAttribute("name").getValue().
                                equals("Zacharias Zottelig")) {
      gast.addContent(wein);
 }  }
```

<?xml?>

Elemente löschen und hinzufügen

```
// neuen Gast erzeugen und hinzufügen.
// Dieser trinkt auch Wein

Element max = new Element("gast");
max.setAttribute("name", "Max Muster");
max.addContent(new Element("getraenk").
                              addContent("Wein"));
party.addContent(max);
```

Speichern in XML-Dateien

<?xml?>

- Das Ganze XML-Dokument soll in einer Datei gespeichert werden.
  Hierzu dient die Klasse XMLOutputter aus jdom2.outputter

```
XMLOutputter outp = new XMLOutputter();
```

- Die Ausgabe soll schön formatiert erfolgen

```
outp.setFormat(Format.getPrettyFormat());
```

- Es geht auch ohne Formatierung: `Format.getRawFormat()`

- Formatierte Ausgabe auf der Konsole

```
outp.output(xmlDocument, System.out);
```

- Formatierte Ausgabe in eine Datei

```
outp.output(xmlDocument,
                    new FileOutputStream(new File("party.xml")));
```

XPATH mit Java

<?xml?>

- XPATH-Fabrik per Singleton-Pattern erzeugen

```
XPathFactory xpf = XPathFactory.instance();
```

- XPATH-Ausdruck zum Suchen von Elementen erzeugen

```
XPathExpression<Element> xPath =
        xpf.compile("//getraenk", Filters.element());
```

- Funktioniert auch mit Attributen: `Filters.attribute()`

- XPATH-Ausdruck auf JDOM2.x Dokumente anwenden

```
List<Element> getraenke = xPath.evaluate(xmlDocument);
```

- veralteter XPATH-Zugriff mittels JDOM 1.x

```
XPath xp= XPath.newInstance("//getraenk");
List<Element> getraenke = xp.selectNodes(xmlDocument);
```

XPATH mit Java

<?xml?>

```
// Alle Getränke ausgeben
if (getraenke.isEmpty()) {
   System.out.println("keine Getränke");
} else {
   System.out.println("Anzahl Getränke " + getraenke.size());
   for (Element getraenk : getraenke) {
       System.out.println(getraenk.getTextTrim());
   }
}
// Wer hat alles Wein getrunken
xPathExpression = xpf.compile("//getraenk[node()='Wein']/..", Filters.element());
List<Element> weinTrinker = xPathExpression.evaluate(xmlDocument);
System.out.println("Weintrinker");
if (!weinTrinker.isEmpty()) {
    for (Element e : weinTrinker) {
        System.out.println("Name: " + e.getAttribute("name").getValue());
    }
}
```

XPATH mit Java

<?xml?>

```
// Alle Betrunkenen nach Hause schicken

xPathExpression = xpf.
     compile("//zustand[@nuechtern='false']/..", Filters.element());
List<Element> betrunkene = xPathExpression.evaluate(xmlDocument);
for (Element e : betrunkene) {
   System.out.println(e.getAttribute("name").getValue() +
                                   " geht nach Hause");
   party.removeContent(e);
}
```

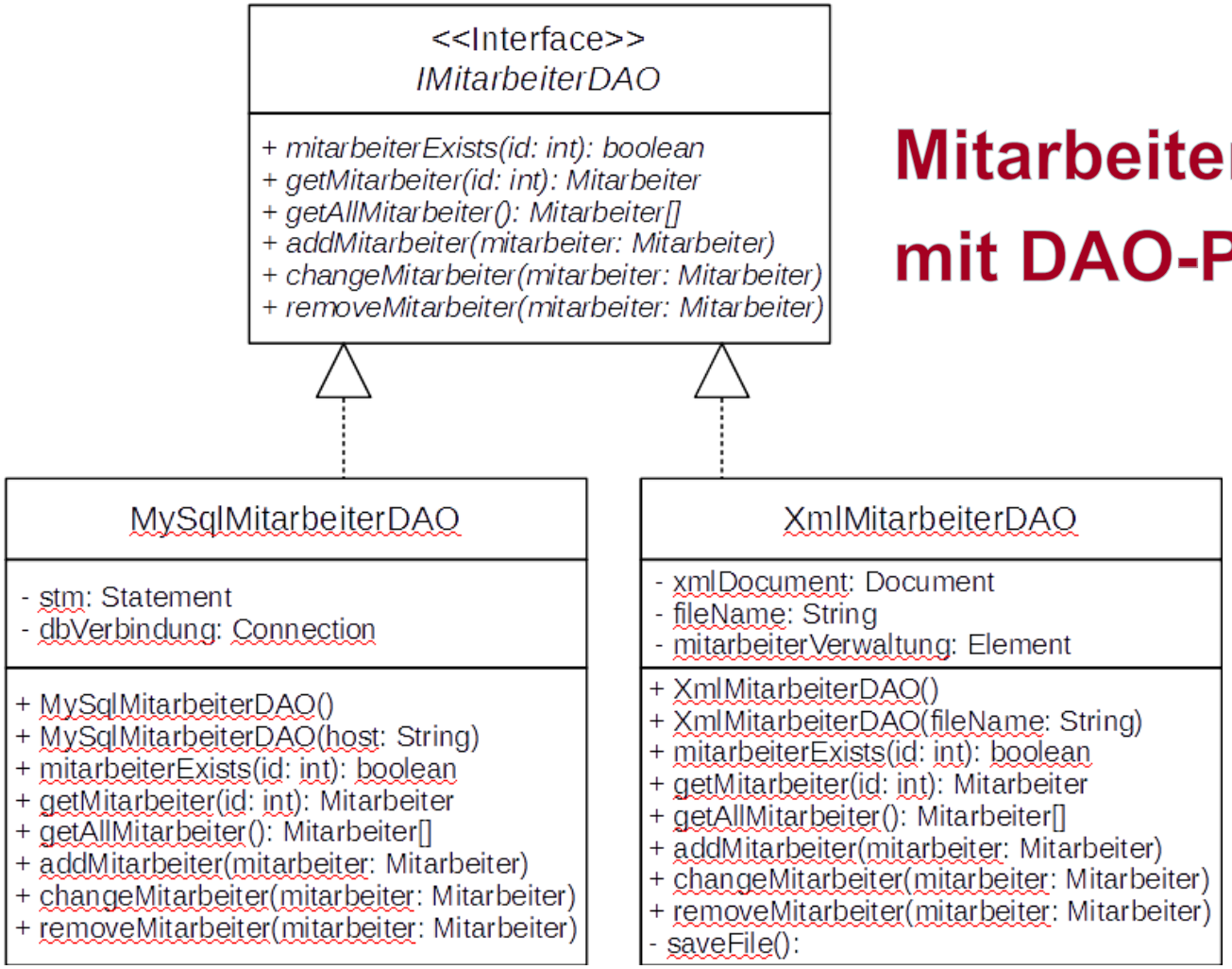
Exkurs: DAO-Pattern

<?xml?>

- **Data Access Object** (**DAO**, englisch für *Datenzugriffsobjekt*) ist ein Entwurfsmuster, das den Zugriff auf unterschiedliche Arten von Datenquellen (z. B. Datenbanken, Dateisystem, etc.) so kapselt, dass die angesprochene Datenquelle ausgetauscht werden kann, ohne dass der aufrufende Code geändert werden muss.
  Dadurch soll die eigentliche Programmlogik von technischen Details der Datenspeicherung befreit werden und flexibler einsetzbar sein.
  (Quelle: Wikipedia)
- Über ein **Data Access Object Interface** werden Standard Operation definiert, die alle realen Implementierungen mitbringen müssen.
- Hierdurch ist ein vereinheitlichter Zugriff auf eine MySql und andere SQL-Datenbanken, eine CSV-Datei, eine Objektorientierte Datenbank, eine JSON-Datei, Mapping mit Hibernate, eine XML-Datei … möglich.
- Nur geringe Änderungen am Quellcode bei der Umstellung auf eine andere (persistente) Speicherung erforderlich.

Exkurs: DAO-Pattern

<?xml?>



Mitarbeiterverwaltung mit DAO-Pattern

Exkurs: DAO-Pattern

<?xml?>

```
public class TestMitarbeiterDAO {
  public static void main(String[] args) {
    try {  // Mitarbeiterverwaltung in MySql-DB oder in XML-Datei
   //     IMitarbeiterDAO mitarbeiterDAO = new MySqlMitarbeiterDAO();
         IMitarbeiterDAO mitarbeiterDAO = new XmlMitarbeiterDAO();
         Mitarbeiter lisa = mitarbeiterDAO.getMitarbeiter(2);
         lisa.setAlter(35);
         mitarbeiterDAO.changeMitarbeiter(lisa);
         System.out.println(mitarbeiterDAO.getMitarbeiter(2).toString());
         Mitarbeiter neuerMitarbeiter =
                        new Mitarbeiter(5, "Max", "Male", 33, "Chef");
         mitarbeiterDAO.addMitarbeiter(neuerMitarbeiter);
         Mitarbeiter[] alleMitarbeiter = mitarbeiterDAO.getAllMitarbeiter();
         for (Mitarbeiter mitarbeiter : alleMitarbeiter) {
             System.out.println(mitarbeiter.toString());
         }
         mitarbeiterDAO.removeMitarbeiter(neuerMitarbeiter);
      } catch (Exception ex) {
} } }
```

Übung zu JDOM + XPath

<?xml?>

- Realisieren Sie die Klasse XmlMitarbeiterDAO.java
- Fügen Sie Libraries jdom-2.0.5.jar und jaxen-1.1.6.jar zum Projekt hinzu
- Nutzen Sie hierfür JDOM und XPath
- Verwalten Sie ihre Mitarbeiter in der Datei employees.xml

Vergleich der Technologien

<?xml?>

Feature	StAX	SAX	DOM / JDOM
API Type	Pull, streaming	Push, streaming	In memory tree
Ease of Use	High	Medium	High
XPath Capability	No	No	Yes
CPU and Memory Efficiency	Good	Good	Varius
Forward Only	Yes	Yes	No
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes
Create, Read, Update, Delete	No	No	Yes

Quelle: https://docs.oracle.com/javase/tutorial/jaxp/stax/why.html

Quellen

- http://www.torsten-horn.de/techdocs/java-xml.htm
- http://openbook.galileocomputing.de/javainsel9/javainsel_18_007.htm
- http://www.journaldev.com/1206/jdom-parser-read-xml-file-to-object-in-java
- http://www.javacodegeeks.com/2013/05/parsing-xml-using-dom-sax-and-stax-parser-in-java.html
- https://docs.oracle.com/javase/tutorial/jaxp/stax/why.html