Weitere Datenstrukturen: XML, JSON

Lernziele:

- Sie kennen die relevanten Parser um XML und JSON Strukturen zu lesen und in Objekte umzuwandeln
- Sie können die Parser entsprechend anwenden

1 XML und JSON

XML (eXtended Markup Language) und JSON (JavaScript Object Notation) sind beides Datenstrukturen, welche vor allem in Web-Services an Bedeutung gewonnen haben. XML gibt es schon eine Weile und war ursprünglich als Dokumenten-Format gedacht. Auch XML wird weit verbreitet in Webservices verwendet.

Um XML oder JSON zu verarbeiten, braucht es entsprechende Parser.

1.1 XML Parser

Bei XML gibt es zwei Varianten:

Simple API for XML (SAX)	***Document Object Model (DOM)***
Sehr schnell, da es Ereignis-gesteuert wird. D.h. die Struktur muss zum voraus bekannt sein, um auf die entsprechenden Elemente /Tags reagieren zu können.	Eher langsam, da das gesamte XML als DOM Baum abgebildet wird.
Keine Navigation in der XML-Struktur möglich.	Es kann über die Elemente navigiert werden mit *getParent*(), *getChild*(), etc.
Sehr handlich, wenn es sich um grössere XML-Daten handelt und nur ein bestimmter Ausschnitt (also ein Element) benötigt wird. Z.Bsp. In einer Nachricht (news) möchte	Handlich für kleinere Daten, bzw. wenn das Parsen nicht zuviel Zeit und Speicher benötigt.

ich nur die Schlagzeile in meiner Applikation verwenden (ohne Texte, Photos, etc.).	
Kann, muss aber nicht XML-Struktur prüfen.	Hier wird geprüft, dass die XML-Struktur gültig ist.

1.2 Aufgabe: Aus XML ein DOM erstellen

Wir wollen den DOM XML-Parser an einem einfachen Beispiel umsetzen, bevor wir eine Aufgabe mit einem Web-Service lösen.

Beispiel einer XML-Struktur:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
	<CD>
		<TITLE>Greatest Hits</TITLE>
		<ARTIST>Elvis</ARTIST>
		<COUNTRY>USA</COUNTRY>
		<YEAR>1977</YEAR>
	</CD>
	<CD>
		<TITLE>Eros</TITLE>
		<ARTIST>Eros Ramazzotti</ARTIST>
		<COUNTRY>EU</COUNTRY>
		<YEAR>1997</YEAR>
	</CD>
</CATALOG>
```

Jeder Tag im XML entspricht einem Knoten (= *Node*). Somit besteht z.Bsp. der Knoten „CD" aus den Sub-Knoten „TITLE", „ARTIST", etc. Dies entspricht einer Liste von Knoten (= *NodeList*).

Das XML wird in der Klasse Document als DOM abgebildet. Dabei parst der DocumentBuilder das XML und wandelt es in ein Document um (Punkt 1):

```
public NodeList generateNodeList(String filename){

	DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
	DocumentBuilder db;
```

```
Document doc = null;

        try {
            db = dbf.newDocumentBuilder();
            doc = db.parse(new File(filename));  //parse XML
        } catch (SAXException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ParserConfigurationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        //puts all text nodes into full-depth underneath this node:
        doc.getDocumentElement().normalize();

        //get list of CDs:

        NodeList cdElements = doc.getElementsByTagName("CD");
        return cdElements;
    }
```

1

2

Eine Knotenliste wird an einem bestimmten Tag-Element definiert (Punkt 2). Somit hat diese *NodeList* alle Knoten mit „CD" und somit auch alle Kinder (children).

In einer for-Schlaufe können wir nun über diese Liste iterieren und die weiteren Sub-Knoten auslesen, bzw. wiederum einer Liste zuweisen:

```
for (int i=0; i < list.getLength(); i++){
            Node node = test.findSubNode("ARTIST", list.item(i));
            //the text value in a node is considered to be a child
            //so we have to get the childNode:
            NodeList textList = node.getChildNodes();
            System.out.println("Artist =" +textList.item(0).getNodeValue());
}
```

Die Methode *findSubNode* geht dabei sämtliche Kinder-Knoten unterhalb des Knotens „CD" durch und sucht explizit nach einem bestimmten Kind-Knoten (Child-Node):

```
public Node findSubNode(String name, Node node) {

          NodeList list = node.getChildNodes();  //get the children
          for (int i=0; i < list.getLength(); i++) {
              Node subnode = list.item(i);
              if (subnode.getNodeType() == Node.ELEMENT_NODE) {
```

```
                if (subnode.getNodeName().equals(name))   //check if we have the child
                    return subnode;
            }
        }
        return null;
    }
```

Aufgabe: Parsen Sie ein XML-File wie im obigen Beispiel in ein DOM ab und geben Sie die Namen der „ARTIST" aus.

1.3 Aufgabe: XML aus Webservice in ein DOM parsen

Als Erweiterung wollen wir von einem Webservice die aktuellen Orte mit Erdbeben der Stärke 5 oder höher.

Verwenden Sie folgenden Service:

http://earthquake.usgs.gov/fdsnws/event/1/query?format=xml&starttime=2018-12-18&endtime=2018-12-19&minmagnitude=5

wobei „starttime" und „endtime" das Zeitfenster angeben und „minmagnitude" die minimale Stärke.

Um den Service aufzurufen, benötigen Sie eine URL-Verbindung. Untenstehender Code zeigt Ihnen eine Verbindung sowie eine einfache Ausgabe als String:

```
//call web-service and get input stream result:
        URL url = new
URL("http://earthquake.usgs.gov/fdsnws/event/1/query?format=xml&starttime=2018-12-18&endtime=2018-12-19&minmagnitude=5”);
        URLConnection yc = url.openConnection();

        //test and show result as String:
        BufferedReader in = new BufferedReader
        (new InputStreamReader(yc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
```

Welche Elemente erhalten Sie vom Service?

Wir wollen nur das Element *Description* mit den Attributen *type* und *text* dem Benutzer anzeigen. Wie müssen Sie über die XML-Struktur iterieren, damit Sie diesen Wert erhalten?

2 JSON als Webservice Datenformat

JSON ist eine weitere Variante, um Information abzubilden. Die Struktur ist bei weitem einfacher als XML.
JSON basiert auf einer Datenstruktur für JavaScript. Es ist in der Web-Programmierung weit verbreitet.

Ein einfaches Beispiel:

```
"name":"John"
```

JSON Daten bestehen immer aus einem key-value Paar. Sie sind somit wie eine *Map* abgebildet (je nach Programmiersprache auch als *Dictionary* bekannt).

Ein weiteres Beispiel:

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

Wie Sie sehen, können diese Strukturen beliebig verschachtelt werdern. Ein *value* kann wiederum aus einer Menge von JSON Objekten bestehen.
Geschweifte Klammern halten ein Objekt. Eckige Klammern halten ein Array von Objekten.

2.1 JSON Parser für Attribute oder Objekte

Je nach Zweck, gibt es verschiedene Parser. Wenn man nur einzelne Attribute auslesen möchte, eignet sich z.Bsp. das *javax* Package.
Wenn man ganze JSON Objekte in entsprechende interne Objekte abbilden möchte, eignen sich Parser wie GSON oder JACKSON.

Hinweise zu den verschiedenen Varianten finden Sie hier:
https://stackoverflow.com/questions/2591098/how-to-parse-json-in-java
(last accessed Dec. 2018)

2.2 Aufgabe: Webservice mit JSON verwenden

Verwenden Sie denselben Erdbeben-Webservice um die Daten in JSON-Format zu erhalten.
Sie müssen nur „format" in der URL auf „geojson" anpassen ("format=geojson").

Benützen Sie einen Parser, der die entsprechenden Attribute auslesen kann.

Was müssten Sie tun, wenn Sie einen Parser für Objekt-Mapping verwenden möchten?

➔ **Wählen sie Aufgabe 1.3 (mit XML) oder 2.2 (mit JSON) aus und zeigen Sie Ihr Resultat der Lehrperson.**