HAW Hamburg
Fakultät TI
Technik und Informatik

Einführung, Lernen, Datenformate, File I/O

Programmiermethodik 2

Ausblick

Agenda

- Organisation
- Lernen
- Dateizugriff
  - Dateiinformationen
  - Lesen und Schreiben
- Datenformate
  - XML
  - XML mit Java verarbeiten
  - JSON

Organisation

Voraussetzungen

- Erfolgreiche Teilnahme an PM1 und PT
  - Der Stoff dieser Vorlesung wird vorausgesetzt.
  - Achtung: inhaltliche, keine formale Voraussetzung
- Umgang mit *Java 8 (SE)* und *Eclipse*
  - *Java Version 1.8*
  - *Eclipse Luna/Mars*
- Hohe Motivation
- Fähigkeit, systematisch und gewissenhaft zu arbeiten
- Bereitschaft, ein Buch oder Online-Dokumentation zu lesen
- Bereitschaft zum intensiven Üben

Literatur

- Guido Krüger, Thomas Stark: Handbuch der Java-Programmierung, 7. Auflage, Addison-Wesley, 2011
- Reinhard Schiedermeier: Programmieren mit Java, 2. Auflage Pearson Studium, 2010
- Christian Ullenboom: Java 7 - Mehr als eine Insel, Rheinwerk
  - *http://openbook.galileocomputing.de/javainsel/*
  - *http://openbook.galileocomputing.de/java7/*
- Kathy Sierra, Bert Bates: Java von Kopf bis Fuß, O'Reilly, 2006
- Dietmar Ratz, Jens Scheffler, Detlef Seese, Jan Wiesenberger: Grundkurs Programmieren in Java, 6. Auflage, Hanser Fachbuch, 2011

EMIL

- URL: *http://www.elearning.haw-hamburg.de/course/view.php?id=10652*
  Schlüssel zur Selbsteinschreibung: *TIPM2WS15*
- Suchen nach: "Programmiermethodik 2 Jenke"
- alle Informationen zur Vorlesung
- alle Materialien zur Vorlesung
- alle Informationen zum Praktikum
- alle Materialien zum Praktikum

Vorlesung

- 12 x Vorlesung (letzter Termin Wiederholung)
- daher: Veranstaltung endet einige Wochen vor Semesterende
  - genaue Auflistung in EMIL

Praktikum

- 4 Aufgabenblätter
- Bearbeitung in 2er-Teams
- Abgabe
- Abgabe/Vorstellung im Praktikum
  - Aufgabe muss vollständig bearbeitet sein – nur noch punktuelle Anpassungen im Praktikumstermin
  - Vorstellung/Finalisierung im Praktikum, wie gehabt
  - offensichtliche Plagiate werden geahndet – jedes Team muss eine eigene Lösung entwickeln
  - jedes Teammitglied muss gesamte Lösung erläutern können

Inhalt

- Dateien, Datenformate
- Generics
- Lambdas
- Streams
- Threads
- Grafische Benutzerschnittstellen
- Ereignisverarbeitung und Innere Klassen
- Entwurfsmuster
- Reguläre Ausdrücke
- Reflection

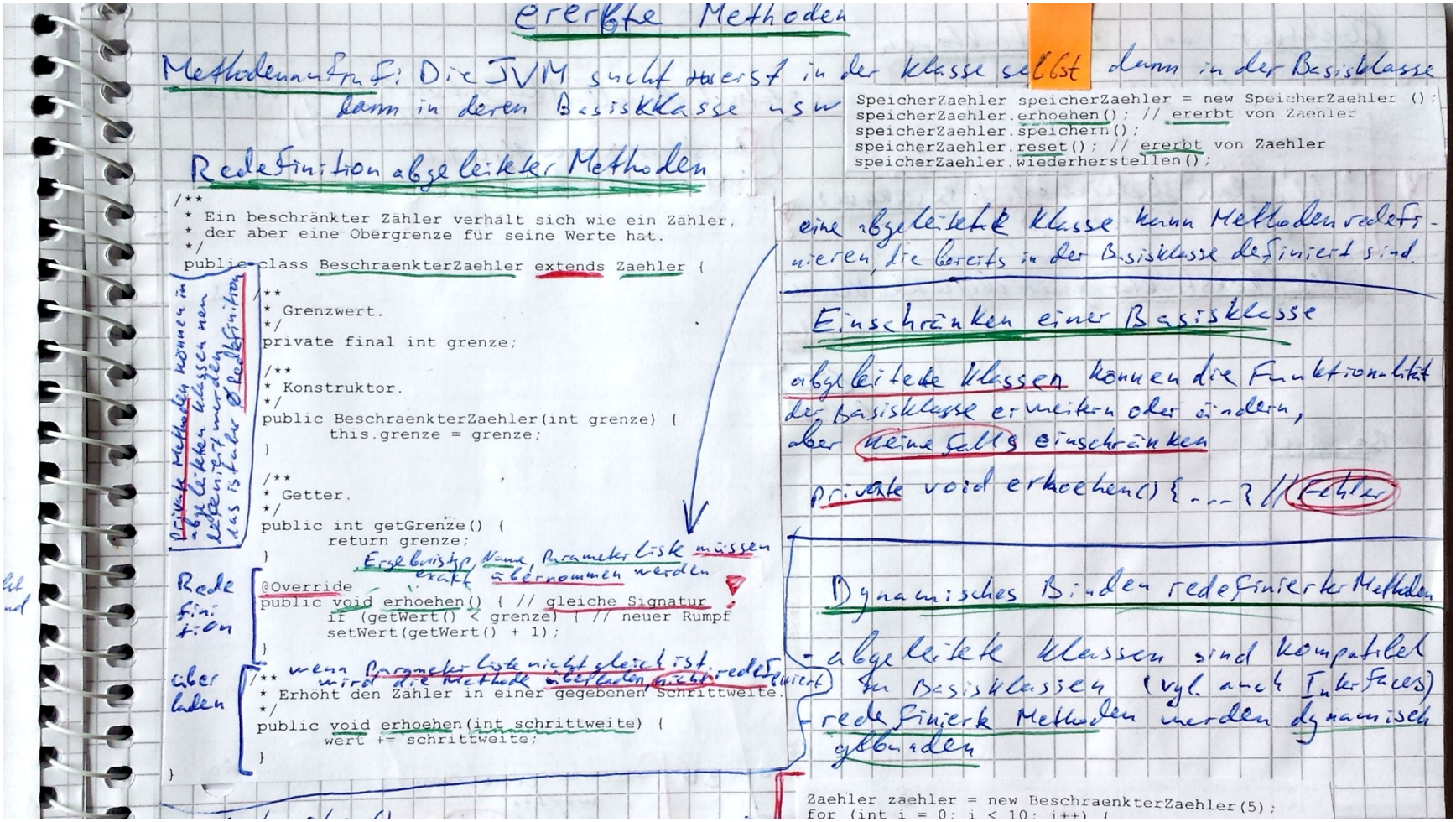
Lernen

Worum geht es?

- Ziele
  - Bestehen der Prüfung
  - erfolgreicher Abschluss des Studiums
  - gute Prüfungsnote/gute Abschlussnote

- Lernen
  - Wie lerne ich?
  - Was muss ich lernen?
  - Wieviel Zeit benötige ich?
  - ...

Idee: Lerntagebuch



Lernen

1. Fachliche Anforderung (z.B. PM1/PT)
2. Wahrnehmung fachlicher Anforderung (Was wird von mir gefordert?)
3. Selbsteinschätzung (Was gelingt mir und was nicht?)
4. Selbstreflexion (Warum gelingt mir etwas nicht?)
5. Individuelle Lernplanung (Was kann ich tun, um in diesem Fach voranzukommen?)
6. (Abstimmung) Zeitmanagement (Wieviel Zeit habe ich realistisch! für Lernen und Studium?)
7. Lern- und Prüfungsstrategie (Wieviel lerne ich und wie bereite ich mich auf die Prüfung vor?)

Dateiinformationen

File

- Klasse `java.io.File`
- Betriebssystem-unabhängige Repräsentation
  - einer Datei ("File")
  - oder eines Verzeichnisses ("Directory")
- Abfrage/Veränderung von Informationen über eine Datei/ein Verzeichnis
  - Verzeichnisse/Dateien angelegt oder gelöscht werden
  - Zugriffstests durchgeführt werden
  - Verzeichnis- Listings erzeugt werden – auch mit Filter
  - Voraussetzung: ausreichende Reche
- aber
  - kein Zugriff auf den Datei-Inhalt
  - keine open/close- oder read/write-Operationen

File

- Konstruktoren
  - Es wird keine physikalische Datei erzeugt, nur ein Java-Objekt!

```
File(String pathname)
File(String parent, String child)
```

- Zugriff auf den Pfadnamen

```
String getName()
String getPath()
String getAbsolutePath()
String getParent()
```

- Betriebssystem-abhängiges Trennzeichen ("/" oder "\") über Konstante `File.separator` ermittelbar

File

- Informationen über die Datei

```
boolean exists()
boolean canWrite()
boolean canRead()
boolean isFile()
boolean isDirectory()
long length()
long lastModified()
```

- Lesen von Verzeichniseinträgen
  - File-Objekt muss vom Typ "Directory" sein
  - beide Methoden liefern alle Verzeichniseinträge
    - Dateien und direkte Unterverzeichnisse

```
String[] list()
File[] listFiles()
```

File

- Ändern von Verzeichniseinträgen

```
boolean createNewFile()
static File createTempFile(String prefix, String suffix)
boolean mkdir()
boolean renameTo(File dest)
boolean delete()
```

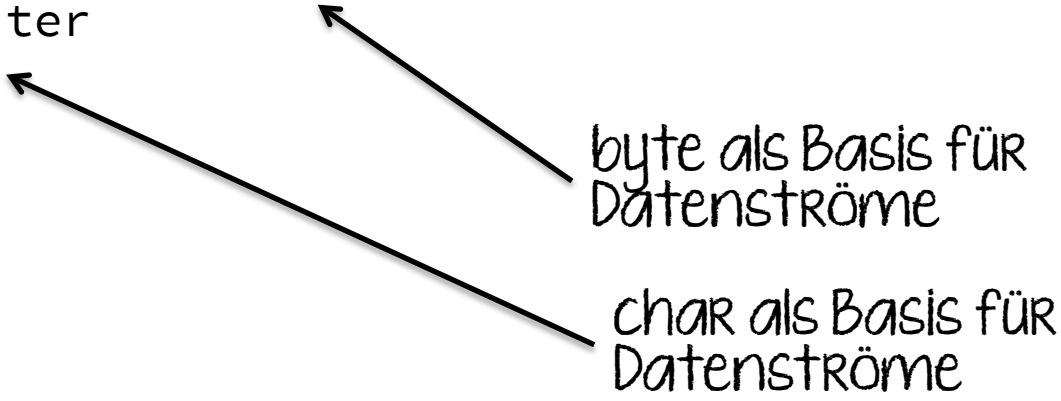
- Weitere Methoden
  - Package `java.nio.file`

HAW Hamburg
Fakultät TI
Technik und Informatik

Dateien: Lesen und Schreiben

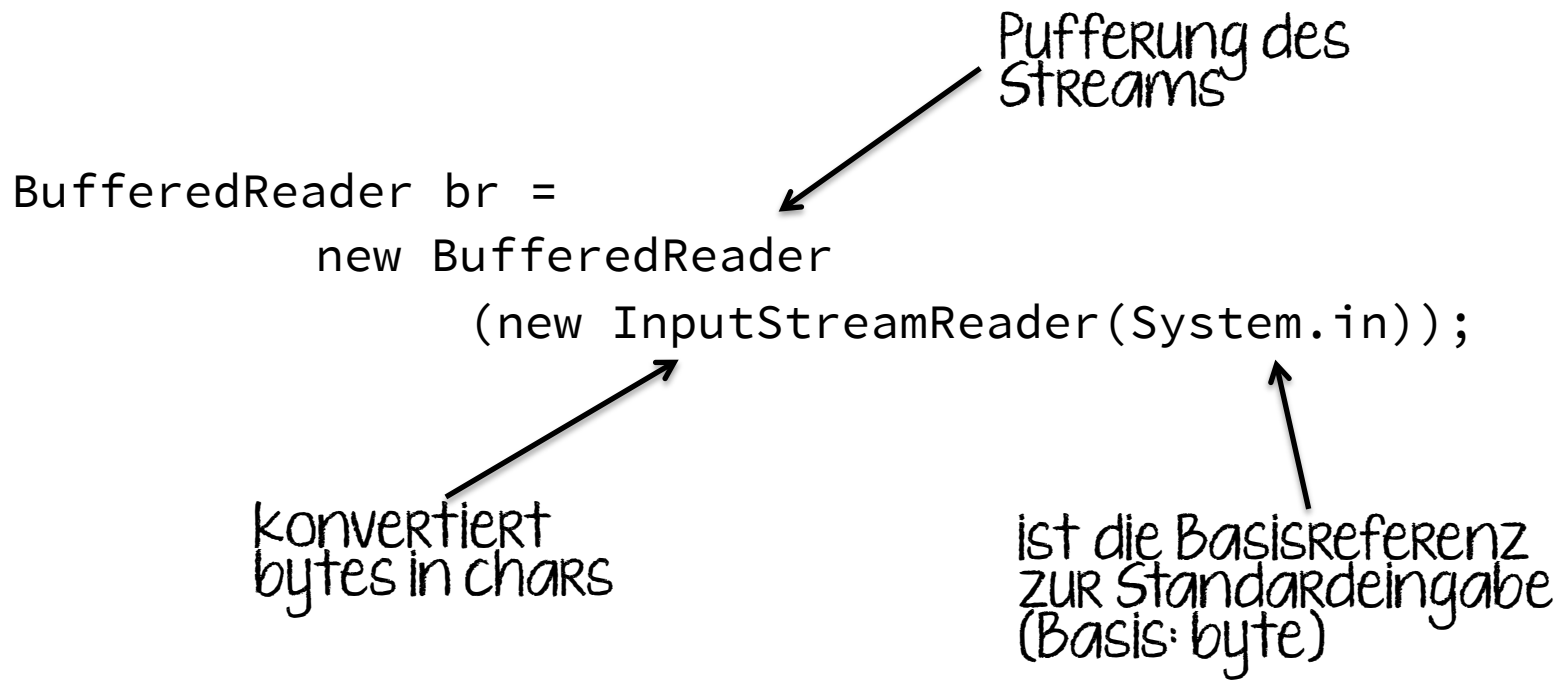
Lesen und Schreiben

- Unterscheidung im `java.io`-Package
  - `InputStream/OutputStream`
  - `Reader/Writer`



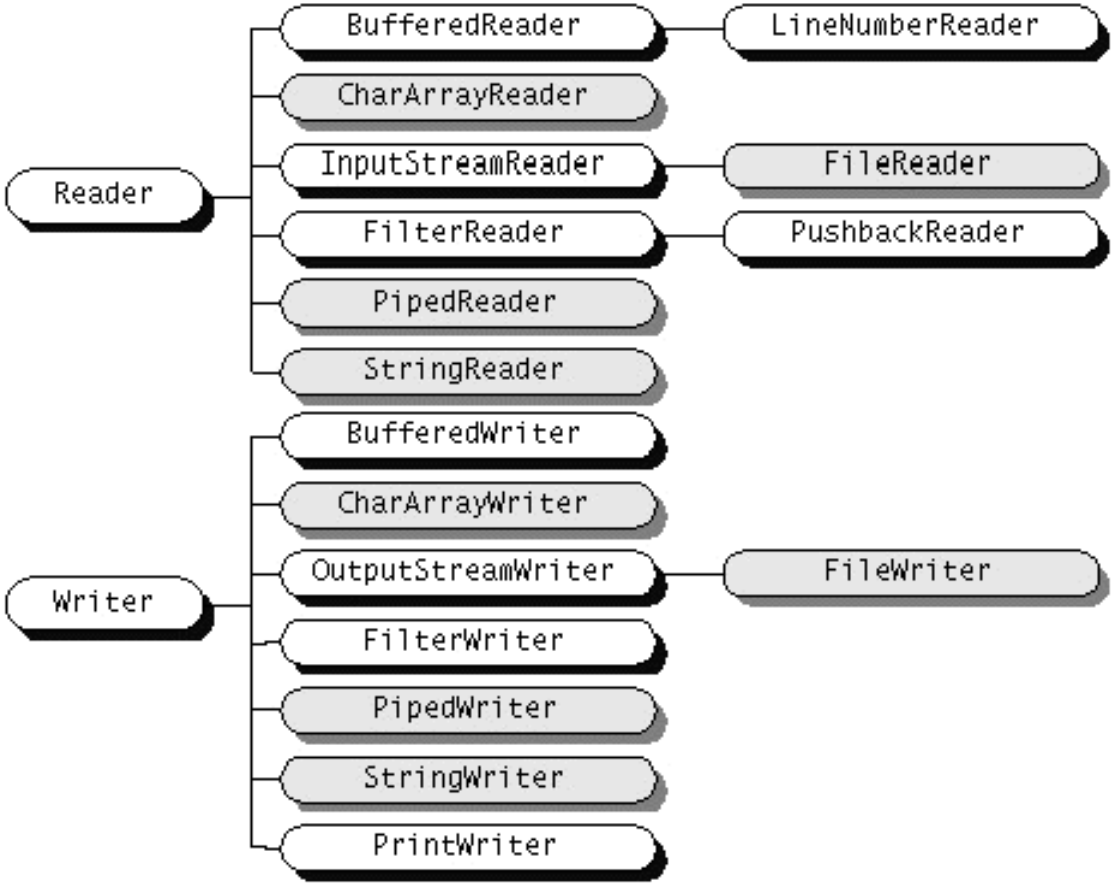
Lesen und Schreiben

Beispiel: Lesen von der Standard-Eingabe



Lesen und Schreiben

- Überblick `Reader/Writer`



Lesen

Vorgehen

- Gerät anschalten (durch Konstruktor)
- solange prüfen, ob noch mehr gelesen werden kann oder das Ende erreicht ist
  - Info aus dem Stream lesen (bytes, chars, String, …)
- Gerät ausschalten

- Pseudocode

```
open stream
while more information
    read information
close stream
```

Lesen

- Reader öffnen

```
private BufferedReader openReader(String filename) {
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(new FileReader(filename));
    } catch (FileNotFoundException e) {
        closeReader(reader);
        e.printStackTrace();
    }
    return reader;
}
```

- Daten lesen

```
List<String> list = new ArrayList<String>();
String zeile = null;
try {
    while ((zeile = reader.readLine()) != null) {
        list.add(zeile);
    }
} catch (IOException e) {
    closeReader(reader);
    e.printStackTrace();
}
```

- Reader schließen

```
private void closeReader(BufferedReader reader) {
    try {
        if (reader != null) {
            reader.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Schreiben

Vorgehen

- Gerät anschalten (durch Konstruktor)
- solange prüfen, ob noch mehr geschrieben werden kann
  - Info in den Stream schreiben (bytes, chars, String, …)
- Gerät ausschalten

- Pseudocode

```
open stream
while more information
    write information
close stream
```

Schreiben

- Writer öffnen

```
private static PrintWriter openWriter(String filename) {
    PrintWriter writer = null;
    try {
        writer = new PrintWriter(new BufferedWriter(
                 new FileWriter(filename)));
    } catch (IOException e) {
        closeWriter(writer);
        e.printStackTrace();
    }
    return writer;
}
```

- Daten schreiben

```
PrintWriter writer = openWriter("sorted.txt");
for (String line : list) {
    writer.println(line);
}
closeWriter(writer);
```

- Writer schließen

```
private static void closeWriter(PrintWriter writer) {
    if (writer != null) {
        writer.close();
    }
}
```

Übung: File

- Schreiben Sie Pseudocode für die folgenden Anforderungen
  - Prüfen Sie, ob die Datei "*dummy.txt"* existiert"
  - Falls ja
    - lesen Sie die letzte Zeile
    - gehen Sie davon aus, dass dort eine Zahl steht
    - addieren Sie 1 zu der Zahl und schreiben das Ergebnis zusätzlich in die Datei
  - falls nein
    - Schreiben Sie die Zahl 1 in die Datei

HAW Hamburg
Fakultät TI
Technik und Informatik

Datenformate: XML

Auszeichnungssprache

- strukturierten Gliederung von Texten und Daten
- Idee: besondere Bausteine durch Auszeichnung hervorzuheben
- Anwendungsgebiete
  - Text: Überschriften, Fußnoten und Absätzen
  - Vektorgrafik: Grafikelementen wie Linien und Textfelder
  - ...
- Beispiel

```
<Überschrift>
Mein Buch
<Ende Überschrift>
Hui ist das <fett>toll<Ende fett>.
```

Auszeichnungssprache

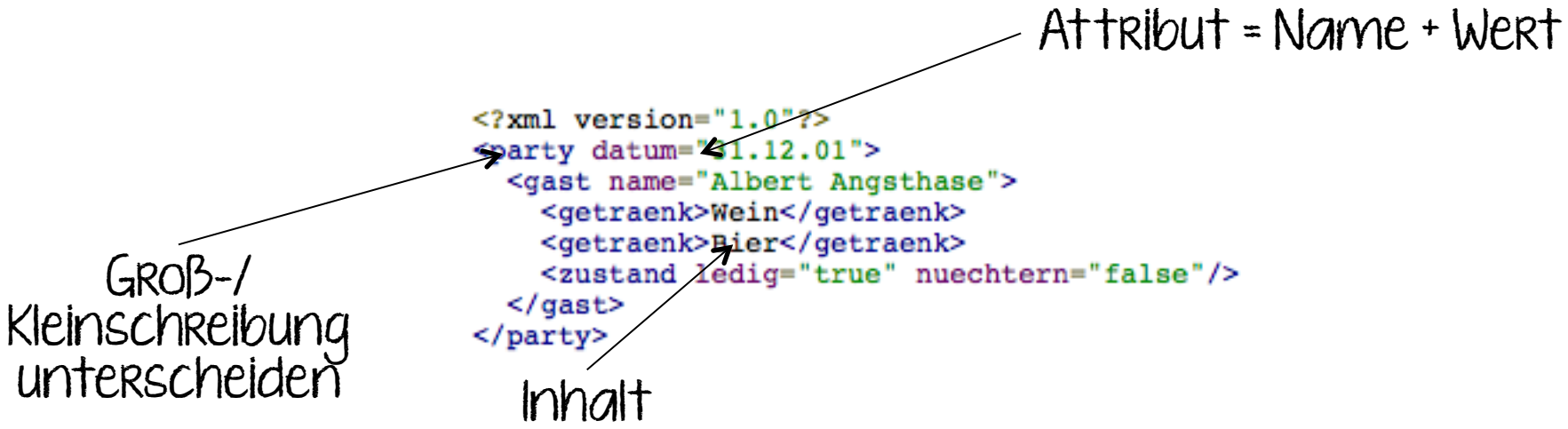
- Definition einer Auszeichnungssprache (Metasprache)
- Mitte der 1980er-Jahre: ISO-Standard die *Standard Generalized Markup Language* (SGML)
- ab. Version 2: HTML als SGML-Anwendung
- Vorteile:
  - Korrektheit
  - Leistungsfähigkeit
- Nachteil:
  - sehr (zu?) stringent

XML

- entwickelt durch W3C
- Basis: SGML
- Ziele:
  - einfach zu nutzen
  - flexibel
- Ergebnis: eXtensible Markup Language (XML)

Aufbau

- XML-Dokument besteht aus
  - hierarchische Schachtelung
  - strukturierte Elemente
  - dazwischen: Inhalt
  - Elemente können Attribute beinhalten
- Beispiel



Zwei Varianten für Elemente

Variante 1: Element mit Inhalt

- Syntax: Element = öffnendes Tag + Inhalt + schließendes Tag
- Beispiel:

```
<getraenk>Wein</getraenk>
```

Variante 2: Element ohne Inhalt

- Syntax: <[...] />
- Beispiel:

```
<zustand ledig="true" nuechtern="false" />
```

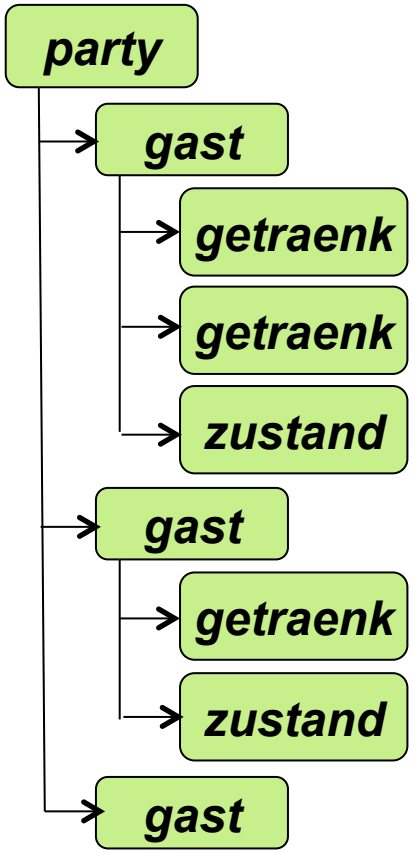
Tags

- keine vordefinierten Tags (wie bei HTML)
- daher keine automatische Formatierung möglich
- dennoch:
  - **wohlgeformtes** Dokument nur, falls Bedingungen erfüllt werden
  - ansonsten kein XML-Dokument
- wohlgeformt:
  - Elemente wie auf vorheriger Folie
  - hierarchische Elemente: umgekehrte Reihenfolge ihrer Öffnung wieder geschlossen
  - es muss Wurzelelement geben (beinhaltet alle anderen Elemente)

XML-Dokument als Baum

- Analogie: hierarchisches XML-Dokument vs. Baumstruktur

```
<?xml version="1.0" ?>
<party datum="31.12.01">
    <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>
```



Besonderheiten

- spezielle Zeichen
  - &, <, >, ", ' haben besondere Bedeutung
  - müssen im Text abgebildet werden (durch Entitäten)
  - Entitäten: `&`, `<`, `>`, `&quot;`, `&apos;`
- Kommentare
  - werden beim Lesen des Dokuments übergangen
  - Syntax: `<!-- Kommentar -->`
  - Hinweis: bester Kommentar = selbsterklärende Namen und Struktur

Kopfdefinition

- zusätzlich möglich: Meta-Informationen im Kopf
- Beispiel: `<?xml version="1.0" encoding="iso-8859-1"?>`

- falls vorhanden: muss am Anfang des Dokuments stehen

Übung: Affenbande

- Geben Sie ein XML-Dokument an, das folgende Information beinhaltet:
  - in einem Zoo gibt es verschiedenen Sorten von Affen: Paviane und Schimpansen
  - alle Affen leben im Affenhaus
  - jeder Affe hat einen Namen
  - es gibt folgende Affen: Hal (Pavian), Leo (Schimpanse), Sue (Schimpanse)

Es gibt verschiedene korrekte Lösungen!

Beschreibungssprache für Aufbau

- Beschreibung eines bestimmten Typs von XML-Dokumenten
- z.B. XML-Dokument für eine spezielle Anwendung
  - muss bekannten Aufbau haben
- zwei Formate:
  - Document Type Definition (DTD) ← betrachten wir weiter
  - XML Schema
- Format eingehalten → XML Dokument ist gültig

Document Type Definition (DTD)

- Wir entwickeln DTD für folgenden Anwendungsfall (Beispieldokument):

```
<?xml version="1.0" ?>
<party datum="31.12.01">
    <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>
```

Analyse Party-XML-Dokument

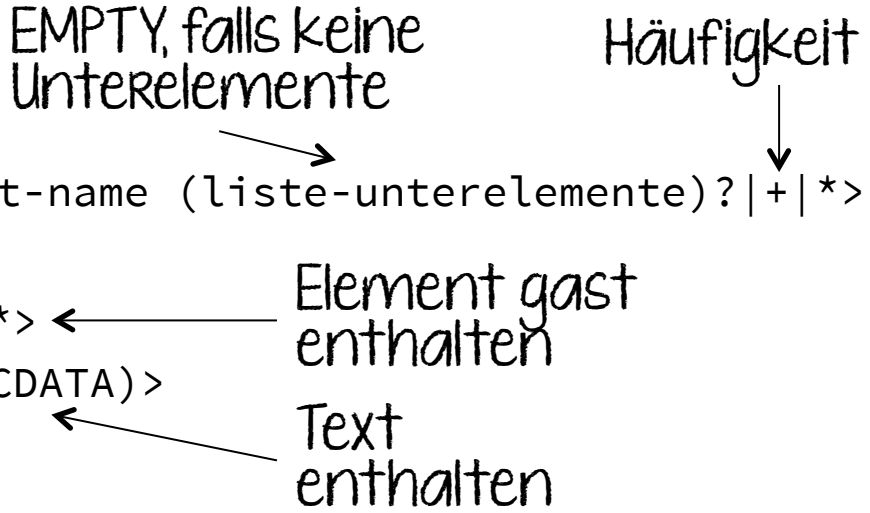
Element-name	Attribute	Untergeordnete Elemente	Aufgabe
party	datum Datum der Party	gast	Wurzelelement mit dem Datum der Party als Attribut
gast	name Name des Gastes	getraenk und zustand	Die Gäste der Party; Name des Gastes als Attribut
getraenk			Getränk des Gastes als Text
zustand	ledig und nuechtern		Familienstand und Zustand als Attribute

```
<?xml version="1.0" ?>
<party datum="31.12.01">
    <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>
```

Elementbeschreibung: Hierarchie

- Untergeordnete Elemente
  - Syntax: `<!ELEMENT element-name (liste-unterelemente)?|+|*>`
  - Beispiele:

```
<!ELEMENT party (gast)*>
<!ELEMENT getraenk (#PCDATA)>
```



Operator	Häufigkeit
?	1x oder nicht
+	mindestens 1x
*	egal (auch 0x)

Bezeichner	repräsentiert
PCDATA	Text (wird geparst)
ANY	beliebig

Elementbeschreibung: Attribute

- Beschreibung von Attributen
  - Syntax: `<!ATTLIST element-name attribut-name typ modifizierer>`

Modifizierer	Häufigkeit
`#IMPLIED`	Muss nicht vorkommen.
`#REQUIRED`	Muss auf jeden Fall vorkommen.
`#FIXED [Wert]`	Wert wird gesetzt und kann nicht verändert werden.

- Beispiel: `<!ATTLIST party datum CDATA #REQUIRED>`
- auch mehrere Attribute möglich:

```
<!ELEMENT zustand EMPTY>
<!ATTLIST zustand ledig CDATA #IMPLIED
          nuechtern CDATA #IMPLIED>
```

Übung: Gast

- Geben Sie die DTD-Beschreibung für einen Gast an. Ein Gast hat
  - einen Namen (Text, Attribut, erforderlich),
  - einen Zustand (Element-Typ `zustand`, Unterelement, einen oder keinen) und
  - beliebig viele Getränke (Element-Typ `getraenk`, Unterelemente)
- Beispiel:

```
<gast name="Albert Angsthase">
    <getraenk>Wein</getraenk>
    <getraenk>Bier</getraenk>
    <zustand ledig="true" nuechtern="true"/>
</gast>
```

Bezugnahme auf eine DTD

- Angabe im Kopf des XML-Dokuments:
- Beispiel:

```
<!DOCTYPE party SYSTEM "dtd\partyfiles\party.dtd">
```

Zusammenfassung


```
<!ELEMENT party (gast)*>
<!ATTLIST party datum CDATA #REQUIRED>
<!ELEMENT gast (getraenk*, zustand?)>
<!ATTLIST gast name CDATA #REQUIRED>
<!ELEMENT getraenk (#PCDATA)>
<!ELEMENT zustand EMPTY>
<!ATTLIST zustand ledig CDATA #IMPLIED nuechtern CDATA #IMPLIED>
```

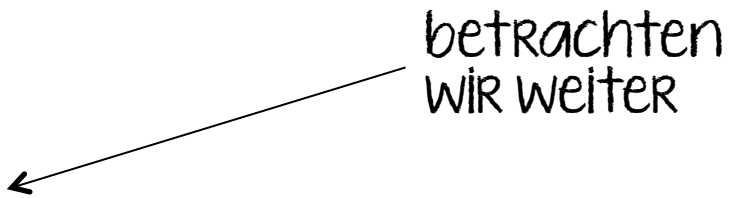
```
<?xml version="1.0" ?>
<!DOCTYPE party SYSTEM "party.dtd">
<party datum="31.12.01">
    <gast name="Albert Angsthase">
        <getraenk>Wein</getraenk>
        <getraenk>Bier</getraenk>
        <zustand ledig="true" nuechtern="false"/>
    </gast>
    <gast name="Martina Mutig">
        <getraenk>Apfelsaft</getraenk>
        <zustand ledig="true" nuechtern="true"/>
    </gast>
    <gast name="Zacharias Zottelig"></gast>
</party>
```

HAW Hamburg
Fakultät TI
Technik und Informatik

XML-Dokumente mit Java Verarbeiten

Ansätze zur XML-Verarbeitung

- Java Klassenbibliothek bietet verschiede Zugänge zur XML-Verarbeitung
- Ansätze
  - DOM-orientierte APIs
    - repräsentieren den XML-Baum im Speicher
    - W3C-DOM, JDOM, dom4j, XOM ...
  - Pull-API
    - wie ein Tokenizer wird über die Elemente gegangen
    - dazu gehören XPP (XML Pull Parser), wie sie der StAX-Standard definiert.
  - Push-API
    - nach dem Callback-Prinzip ruft der Parser Methoden auf und meldet Elementvorkommen)
    - SAX (Simple API for XML) ist der populäre Repräsentant.
  - Mapping-API
    - der Nutzer arbeitet überhaupt nicht mit den Rohdaten einer XML-Datei, sondern bekommt die XML-Datei auf ein Java-Objekt umgekehrt abgebildet
    - JAXB, Castor, XStream, ...



Document Object Model (DOM)

- programmiersprachen-unabhängiges Modell der Struktur
- strikte Hierarchie
- Vorgabe von klaren Schnittstellen
  - werden von Implementierungen umgesetzt
- wir betrachten hier die Implementierung JAXP
  - leichtgewichtige Referenzimplementierung
  - Java 8 beinhaltet JAXP 1.6

Parsen eines XML-Dokuments

- Parsen = Einlesen des Dokument und Aufbau einer internen Repräsentation
- Vorgehen
  - Erstellen eines Builders, der aus der Text-Datei einen DOM-Baum aufbauen kann
  - Umsetzung über Factory-Pattern (siehe Vorlesung "Entwurfsmuster")
  - Lesen der Datei und Dabei Aufbau des Baumes (DOM)

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(new File("files/party.xml"));
```

- Zugriff auf den Wurzelknoten des Dokuments

```
document.getDocumentElement()
```

Verarbeiten eines Elements

- DOM-Baumstruktur besteht aus Knoten `org.w3c.dom.Node`
- Elemente in XML-Baum sind vom Typ `org.w3c.dom.Element`
  - abgeleitet von `org.w3c.dom.Node`
  - Type-Cast erforderlich
- `Node/Element` haben viele Eigenschaften, z.B.:
  - Name: `getNodeName()`
  - Wert: `getNodeValue()`

Attribute eines Elements

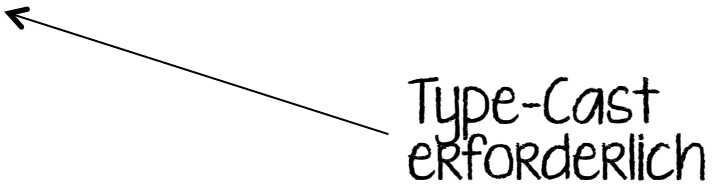
- Attribute sind selber wieder `Node`-Objekte (Schlüssel = Name, Wert = Value)
- Beispiel: Ausgabe aller Attribute in der Form `Schlüssel: Wert` auf der Konsole:

```
NamedNodeMap attribute = element.getAttributes();
for (int i = 0; i < attribute.getLength(); i++) {
  Node attribut = attribute.item(i);
  System.out.print(attribut.getNodeName() + ": " + attribut.getNodeValue());
}
```

Kind-Elemente eines Elements

- Iteration über die Liste der Kind-Elemente eines Elements

```
for (int i = 0; i < element.getChildNodes().getLength(); i++) {
  Node kindKnoten = element.getChildNodes().item(i);
  if (kindKnoten instanceof Element) {
    Element kindElement = (Element) kindKnoten;
     ...
```



Übung: DOM-Parser

- Gegeben ist ein `Element element` aus einem DOM. Schreiben Sie Code zur Ausgabe
  - des Namens,
  - der Namen aller Attribute,
  - der Namen aller Kind-Elemente,

```
Element element = ...
```

HAW Hamburg
Fakultät TI
Technik und Informatik

Datenformate: JSON

JSON

- steht für *JavaScript Object Notation*
- Was ist JSON?
  - schlankes Datenaustauschformat
  - für Menschen einfach zu lesen und zu schreiben
  - für Maschinen einfach zu parsen und zu generieren
  - basierend auf einer Untermenge der JavaScript Programmiersprache
  - programmiersprachenunabhängig
  - kompakter als XML

Vergleich JSON-XML


```json
{
    "gaeste": [
        {
            "name": "Albert Angsthase",
            "getraenke": ["Wein", "Bier"],
            "zustand": {
                "ledig": true,
                "nuechtern": false
            }
        },
        {
            "name": "Martina Mutig",
            "getraenke": ["Apfelsaft"],
            "zustand": {
                "ledig": true,
                "nuechtern": true
            }
        },
        {
            "name": "Zacharias Zottelig"
        }
    ],
    "datum": "31.12.01"
}
```

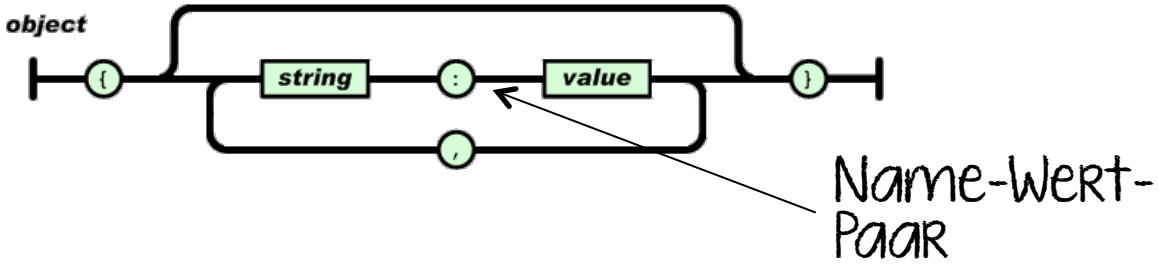
```xml
<?xml version="1.0" ?>
<!DOCTYPE party SYSTEM "party.dtd">
<party datum="31.12.01">
    <gast name="Albert Angsthase">
        <getraenk>Wein</getraenk>
        <getraenk>Bier</getraenk>
        <zustand ledig="true" nuechtern="false" />
    </gast>
    <gast name="Martina Mutig">
        <getraenk>Apfelsaft</getraenk>
        <zustand ledig="true" nuechtern="true" />
    </gast>
    <gast name="Zacharias Zottelig"></gast>
</party>
```

JSON

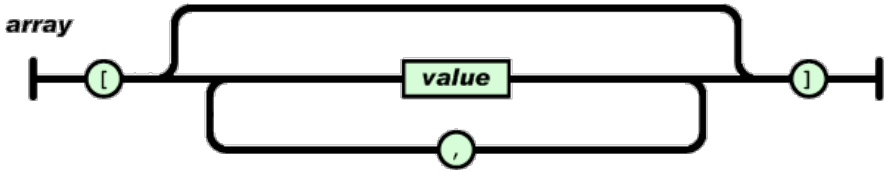
- zwei zentrale Strukturen
  - Name-Wert-Paare
  - geordnete Liste von Werten (Array)
- also: universelle Datenstrukturen
  - von im Prinzip allen modernen Programmiersprachen unterstützt

JSON Datentypen

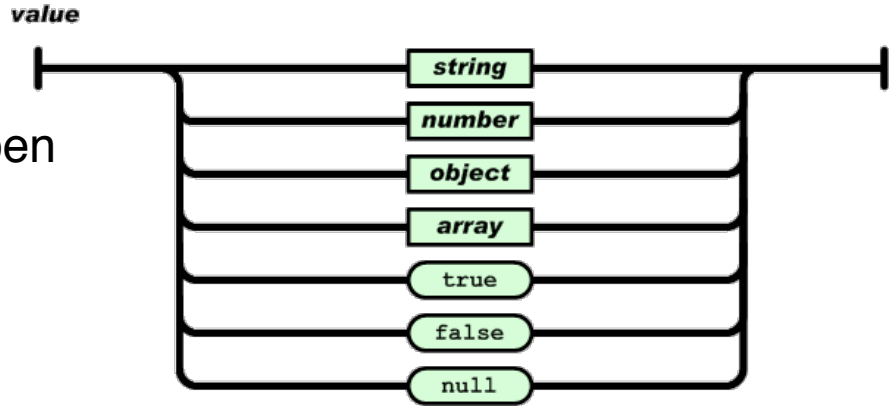
- Objekt (*object*)
  - eingefasst in `{}`



- Array (*array*)
  - eingefasst in `[]`



- Wert (*value*)
  - verschiedenen Typen



JSON: Werte

- Zeichenketten (`String`)
  - Zeichenketten, ähnlich Java (Unicode, Sonderzeichen)
  - eingefasst in Anführungszeichen (`"`)
- Zahlen (`Number`)
  - Ganzzahl (`123`)
  - Fließkommazahl (`3.1415`)
  - mit Exponent (`1.4e-5`)
- `true`/`false`
  - Wahrheitswerte
- `null`
  - leeres Objekt, Platzhalter

Hinweis: Validierung

- Automatisierte Validierung: *http://jsonlint.com/*
  - Gültigkeit eines JSON-Dokuments

Übung: Affenbande Reloaded

- Geben Sie ein JSON-Dokument an, das folgende Information beinhaltet:
  - in einem Zoo gibt es verschiedenen Sorten von Affen: Paviane und Schimpansen
  - alle Affen leben im Affenhaus
  - jeder Affe hat einen Namen
  - es gibt folgende Affen: Hal (Pavian), Leo (Schimpanse), Sue (Schimpanse)

Es gibt verschiedene korrekte Lösungen!

JSON mit Java Verarbeiten

- JSON-Zugriff nicht in Klassenbibliothek enthalten
- daher: externe Bibliothek benötigt
- viele Implementierungen verfügbar
- Referenzimplementierung: *https://jsonp.java.net/*
  - Einbinden in Eclipse: JAR-Bibliothek in Classpath aufnehmen

JSON-Dokument Parsen

- Erzeugen eines Readers
  - der ließt die Datei ein und baut eine interne Repräsentation auf
- Auslesen des Wurzel-Objektes
  - hier: Party-Objekt

```
JsonReader reader = Json.createReader(new FileReader("files/party.json"));
JsonStructure partyStructure = reader.read();
```

JSON-Objekt Verarbeiten


```
private static void verarbeitePartyObjekt(JsonStructure partyStructure) {
  if (partyStructure == null || partyStructure.getValueType() != ValueType.OBJECT) {
    return;
  }

  System.out.println("Party-Objekt gefunden.");

  JsonObject partyObject = (JsonObject) partyStructure;
  Set<Entry<String, JsonValue>> nameWertPaare = partyObject.entrySet();

  // Name-Wert-Paare des Objektes analysieren
  for (Entry<String, JsonValue> eintrag : nameWertPaare) {
    switch (eintrag.getKey()) {
      case "datum":
        System.out.println("  Datum: " + partyObject.getString("datum"));
        break;
      case "gaeste":
        verarbeiteGaeste(partyObject.getJsonArray("gaeste"));
        break;
      default:
        System.out.println("Nicht verarbeiteter Schlüssel: " + eintrag.getKey());
    }
  }
}
```

Prüfen, ob es ein Objekt ist

Name-Wert-Paare extrahieren

je nach Name (Schlüssel) verarbeiten

wir wissen, dass die Gäste in einem Array liegen

JSON-Array Verarbeiten


```
private static void verarbeiteGaeste(JsonArray gaesteArray) {
  if (gaesteArray == null) {
    return;
  }

  // Über Gäste iterieren
  for (Iterator<JsonValue> it = gaesteArray.iterator(); it.hasNext();) {
    JsonValue gast = it.next();
    System.out.println(" Gast gefunden: " + gast);
  }
}
```

Iterator über alle Elemente des Arrays

hier müsste man nun das Gäste-Objekt verarbeiten

Zusammenfassung

- Organisation
- Lernen
- Dateizugriff
  - Dateiinformationen
  - Lesen und Schreiben
- Datenformate
  - XML
  - XML mit Java verarbeiten
  - JSON

Quellen

- Die Folien basieren teilweise auf Vorlesungsfolien von Prof. Martin Hübner, Hochschule für Angewandte Wissenschaften Hamburg
- Der Abschnitt zu XML ist angelehnt an Christian Ullenboom: Java ist auch eine Insel, Rheinwerk, 10. Auflage
- JSON: *http://json.org/json-de.html*, abgerufen am 10.08.2015