

# Dateien, XML, Git

## Programmiermethodik 2

# Ausblick



# Agenda

- Organisation
- Dateien und I/O
  - Dateiinformationen
  - Dateien Lesen und Schreiben
  - Try-With-Resources
- Formate
  - Datenformate: XML
  - XML-Dokumente in Java verarbeiten
- Versionsmanagement, Git



# 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 (aktuelle Version)*
- 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=20166>*
- Schlüssel zur Selbsteinschreibung: **TIPM2WS1617**
- Suchen nach: "Programmierungsmethodik 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, Git
- Generics
- Lambdas
- Streams
- Objektorientierte Programmiertechniken
- Threads
- Grafische Benutzerschnittstellen
- Ereignisverarbeitung und Innere Klassen
- Entwurfsmuster
- Reguläre Ausdrücke/Reflection



# Dateiinformationen

## **Zum Nachlesen:**

Christian Ullenboom: Java ist auch eine Insel, Kapitel 15.1  
[http://openbook.rheinwerk-verlag.de/javainsel/javainsel\\_15\\_001.html](http://openbook.rheinwerk-verlag.de/javainsel/javainsel_15_001.html)

# File

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

# File

- Konstruktiven
  - 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 Verzeichniseinträge

- 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`

# Beispiel

```
File file = new File("files/affenbande.xml");
System.out.println("Vorhanden? " + file.exists());
System.out.println("Verzeichnis? " + file.isDirectory());
System.out.println("absoluter Pfad: " + file.getAbsolutePath());
System.out.println("Größe: " + file.getTotalSpace());
System.out.println("Schreibbar?: " + file.canWrite());
```

- Ausgabe:

*Vorhanden? true*

*Verzeichnis? false*

*absoluter Pfad: /Users/abo781/abo781/code/lehre/pm2/files/  
affenbande.xml*

*Größe: 120101797888*

*Schreibbar?: true*





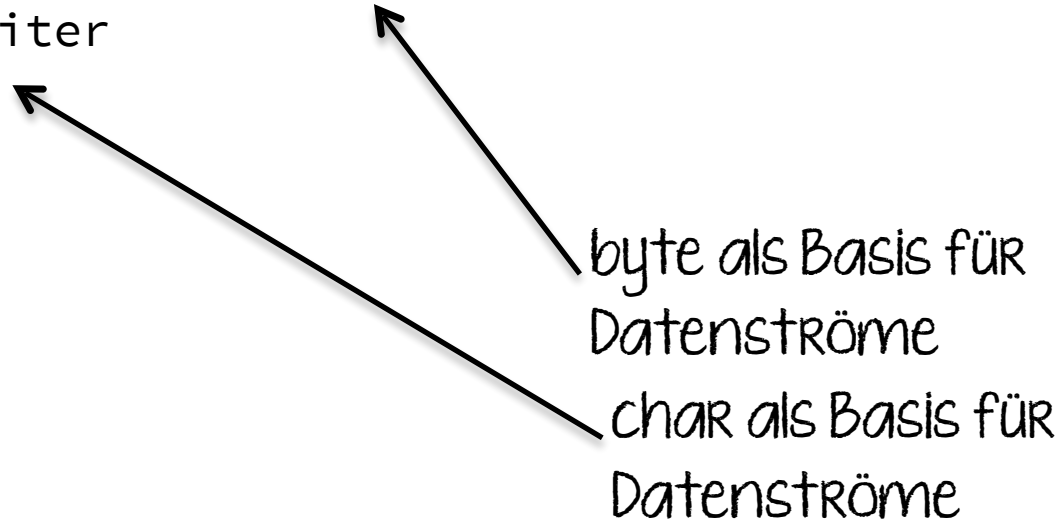
# Dateien: Lesen und Schreiben

## **Zum Nachlesen:**

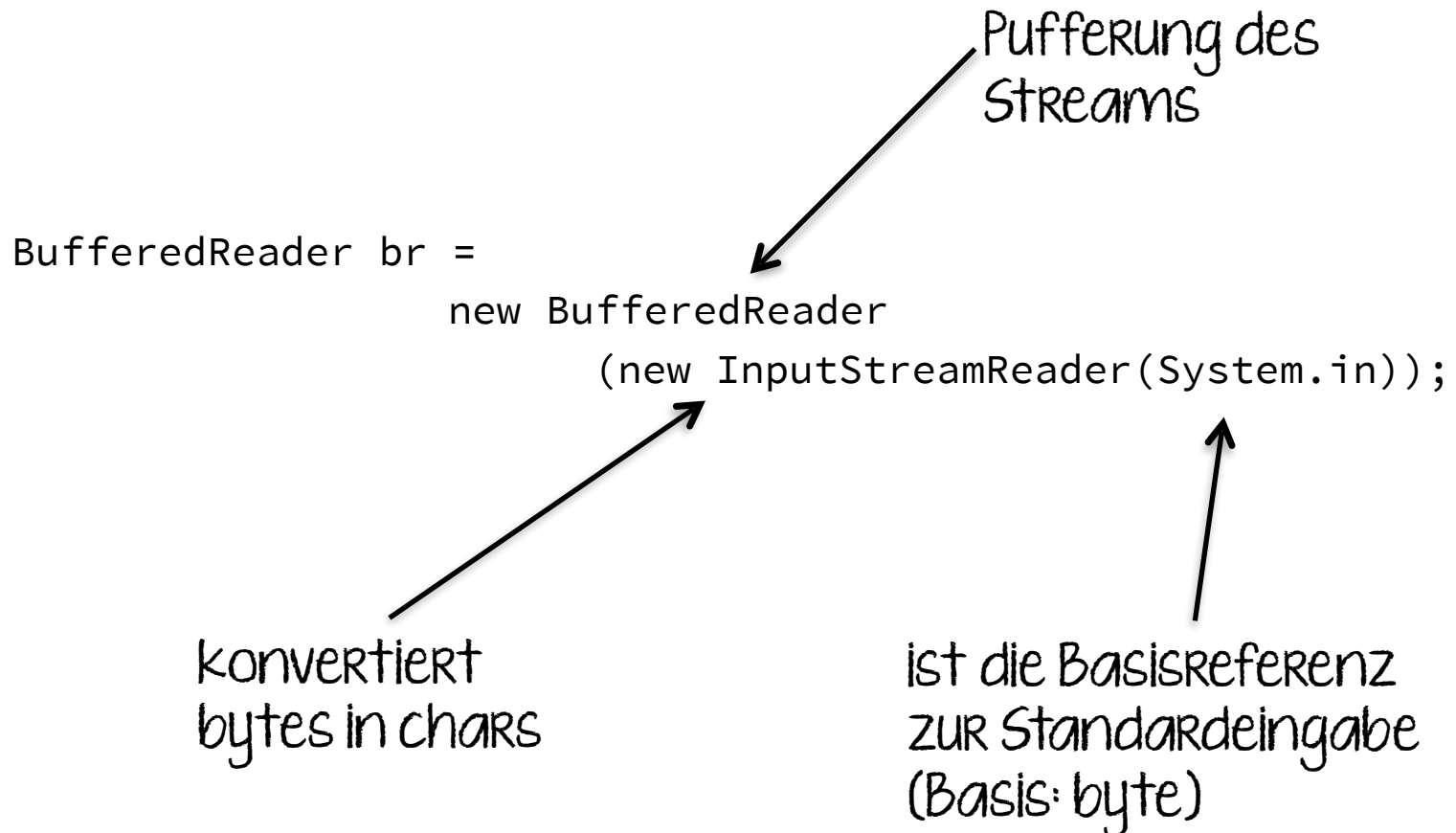
Christian Ullenboom: Java ist auch eine Insel, Kapitel 15.5  
[http://openbook.rheinwerk-verlag.de/javainsel/javainsel\\_15\\_005.html](http://openbook.rheinwerk-verlag.de/javainsel/javainsel_15_005.html)

# Lesen und Schreiben

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



# Beispiel: Lesen von der Standard-Eingabe



# Lesen

- 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

Reader/Stream öffnen

Solange es mehr Daten gibt

Daten lesen

Reader/Stream schliessen

# Lesen

- Reader öffnen

```
BufferedReader reader = null;  
try {  
    reader = new BufferedReader(new FileReader(filename));  
} catch (FileNotFoundException e) {  
    schliessen(reader);  
    e.printStackTrace();  
}
```

- Daten lesen

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

- Reader schließen

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

# Schreiben

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

- Pseudocode

Reader/Stream öffnen

Solange es weitere Daten gibt

Daten schreiben

Reader/Stream schliessen

# Schreiben

- Writer öffnen

```
PrintWriter writer = null;  
try {  
    writer = new PrintWriter(new BufferedWriter(new  
        FileWriter(filename)));  
} catch (IOException e) {  
    schliessen(writer); // Hilfsmethode  
    e.printStackTrace();  
}
```

- Daten schreiben

```
AusDateiLesen readTest = new AusDateiLesen();  
List<String> list = readTest.lesen();  
Collections.sort(list);  
PrintWriter writer = oeffnen("files/sortiert.txt");  
for (String line : list) {  
    writer.println(line);  
}
```

- Writer schließen

```
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





# Try-with-Resources

## **Zum Nachlesen:**

Christian Ullenboom: Java 7: Mehr als eine Insel, Kapitel 7.1  
[http://openbook.rheinwerk-verlag.de/java7/1507\\_01\\_001.html](http://openbook.rheinwerk-verlag.de/java7/1507_01_001.html)

# Ausnahmebehandlung in Java 6

## – Beispiel

try-catch  
innerhalb von  
try-catch

```
FileInputStream inFile = null;
FileOutputStream outFile = null;
try {
    inFile = new FileInputStream("files/unsortiert.txt");
    outFile = new FileOutputStream("files/unsortiert_copy.txt");
    // A buffer is required for the copied data
    byte[] buffer = new byte[65536];
    int len;
    // Read to buffer, write to destination
    while ((len = inFile.read(buffer)) > 0) {
        outFile.write(buffer, 0, len);
    }
    try {
        inFile.close();
        outFile.close();
    } catch (IOException e1) {
        // Fehler beim Schliessen
    }
} catch (FileNotFoundException e) {
    try {
        inFile.close();
        outFile.close();
    } catch (IOException e1) {
        // Fehler beim Schliessen
    }
} catch (IOException e) {
    try {
        inFile.close();
        outFile.close();
    } catch (IOException e1) {
        // Fehler beim Schliessen
    }
}
```

mehrere  
Aufrufe von  
close()

# try-with-resources

- Erweiterung des try-Blocks um die Angabe von Ressourcen


```
try (Ressource1; Ressource2; ...)  
{  
    <Anweisung>  
    ...  
}
```

- Ressource
  - Objekt, das das Interface `java.lang.AutoCloseable` implementiert
  - z.B. alle Stream-Klassen aus `java.io`
- Ressource nach Beendigung des try-Blocks automatisch geschlossen
  - auch nach einer Exception
  - automatischer Aufruf der `close()`-Methode

# Ausnahmebehandlung in Java 6

## - Beispiel

Autoclosable  
in try-Aufruf



```
try (FileInputStream inFile = new FileInputStream("file.txt");) {  
    // A buffer is required for the copied data  
    byte[] buffer = new byte[65536];  
    int len;  
    // Read to buffer, write to destination  
    while ((len = inFile.read(buffer)) > 0) {  
        System.out.println(buffer);  
    }  
} catch (FileNotFoundException e) {  
    // File not found  
    e.printStackTrace();  
} catch (IOException e) {  
    // I/O error  
    e.printStackTrace();  
}
```

kein close()-  
Aufruf  
notwendig!

# Übung: Geräusch-Sensor

- Schreiben Sie eine Klasse `GeraeuschSensor`
- Beim Verbinden mit dem Sensor (Konstruktor) und beim Trennen der Verbindung (`verbindungBeenden()`) kann eine `IOException` auftreten
- Stellen Sie sicher, dass der `GeraeschSensor` zusammen mit `try-with-resources` verwendet werden kann
- Schreiben Sie einen kleinen Code-Abschnitt in dem ein Objekt der Klasse erzeugt wird, ein Wert (Lautstärke) ausgelesen wird und die Verbindung zum Sensor wieder geschlossen wird



## Datenformate: XML

### **Zum Nachlesen:**

Christian Ullenboom: Java ist auch eine Insel, Kapitel 16  
[http://openbook.rheinwerk-verlag.de/javainsel/javainsel\\_16\\_001.html](http://openbook.rheinwerk-verlag.de/javainsel/javainsel_16_001.html)

# 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

Groß-/  
Kleinschreibung  
unterscheiden

```
<?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>  
</party>
```

Inhalt

Attribut = Name + Wert

# 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" />
```

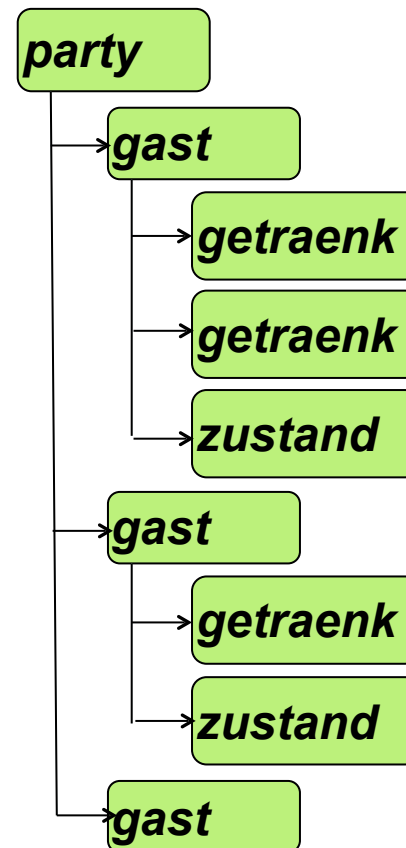
# Tags

- 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 Wurzelement 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: &amp;, &lt;, &gt;, &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"?>`

XML-Version



Zeichen-Kodierung  
(Default: UTF-8)



- 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	Wurzelement 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

EMPTY, falls keine  
Unterelemente

Häufigkeit



## - Untergeordnete Elemente

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

- Beispiele:

`<!ELEMENT party (gast)*>` ← Element gast enthalten

`<!ELEMENT getraenk (#PCDATA)>`

← Text enthalten

Operator	Häufigkeit	Bezeichner	repräsentiert
?	1x oder nicht	PCDATA	Text (wird geparkt)
+	mindestens 1x	ANY	beliebig
*	egal (auch 0x)		

# 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>
```





# XML-Dokumente mit Java Verarbeiten

**Zum Nachlesen:**

Christian Ullenboom: Java ist auch eine Insel, Kapitel 16.3  
[http://openbook.rheinwerk-verlag.de/javainsel/javainsel\\_16\\_003.html](http://openbook.rheinwerk-verlag.de/javainsel/javainsel_16_003.html)

# Ansätze zur XML-Verarbeitung

- Java Klassenbibliothek bietet verschiedene 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, ...

← betrachten  
wir weiter

# 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;  
        ...  
    }  
}
```

← Type-Cast  
erforderlich

# Neuen DOM aufbauen

- DOM erstellen

```
DocumentBuilderFactory docFactory =  
    DocumentBuilderFactory.newInstance();  
DocumentBuilder docBuilder =  
    docFactory.newDocumentBuilder();  
Document doc = docBuilder.newDocument();  
Element rootElement = doc.createElement("company");  
doc.appendChild(rootElement);
```

...

- In Datei schreiben

```
TransformerFactory transformerFactory =  
    TransformerFactory.newInstance();  
Transformer transformer = transformerFactory.newTransformer();  
DOMSource source = new DOMSource(doc);  
StreamResult result = new StreamResult(new File(<Dateiname>));  
transformer.transform(source, result);
```



# 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` = ...

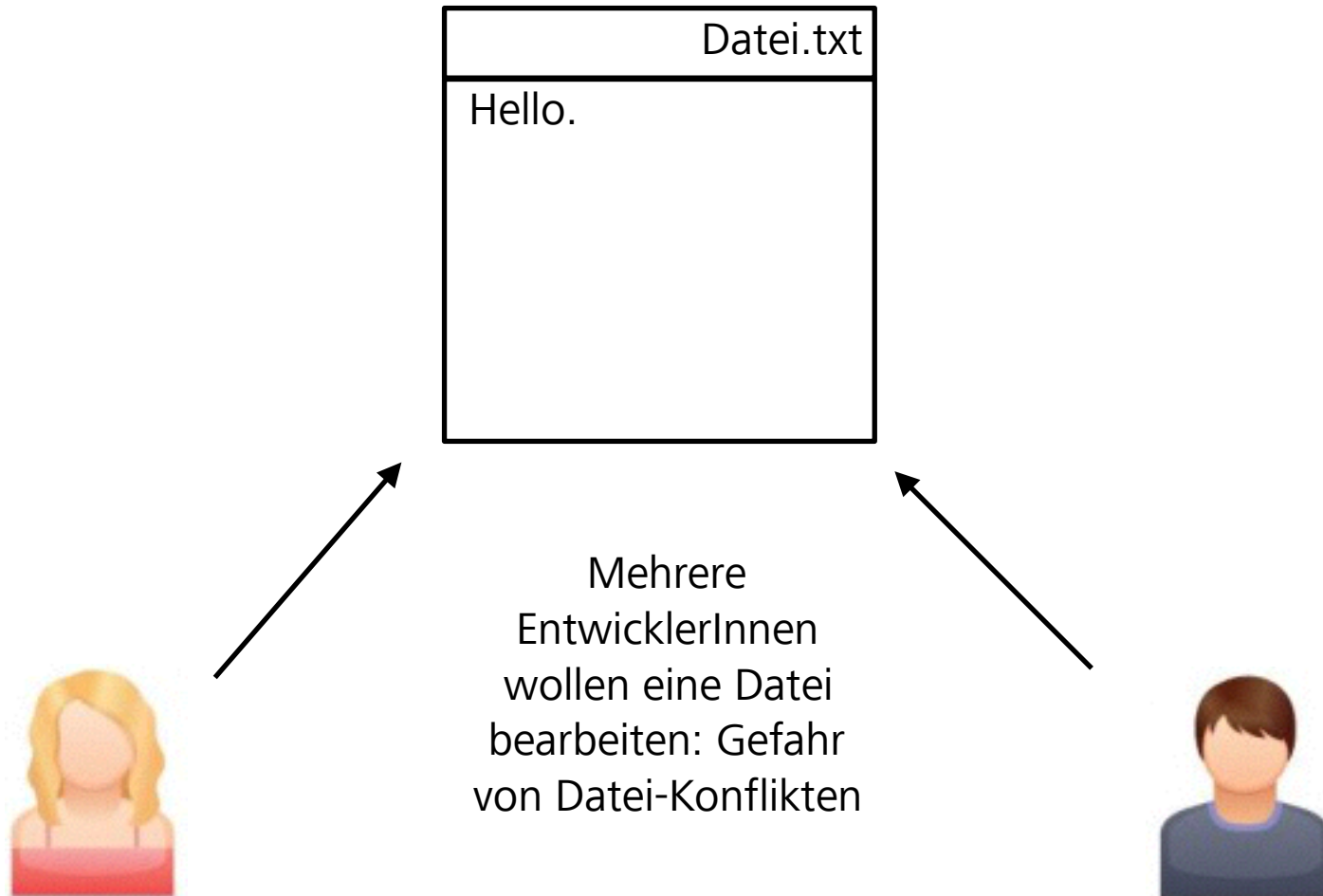


# Versionsmanagement, Git

**Zum Nachlesen:**  
Git Dokumentation

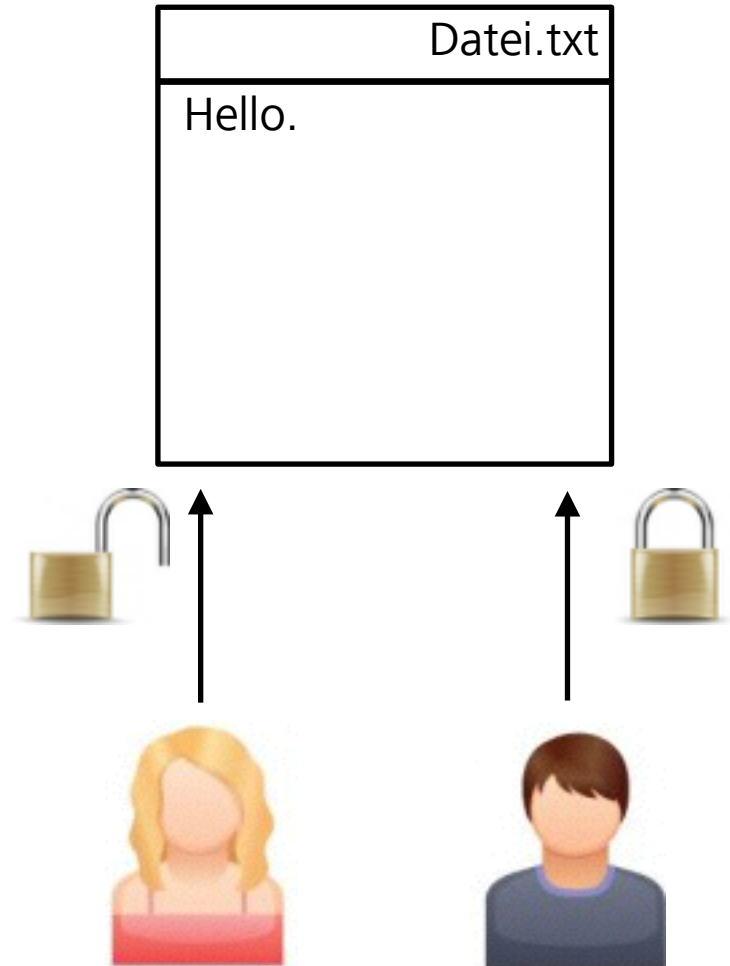
<https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>

# Versionsmanagement



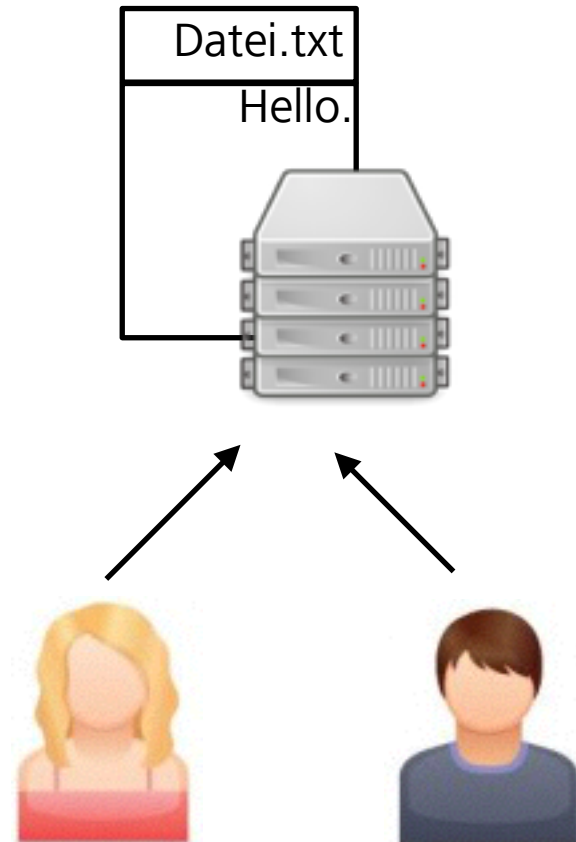
# Versionsmanagement

- Idee 1: Locking
  - eine Person bekommt exklusiven Zugang
  - alle anderen Person haben keine Zugang
- Nachteil:
  - Mehrheit der EntwicklerInnen ist blockiert



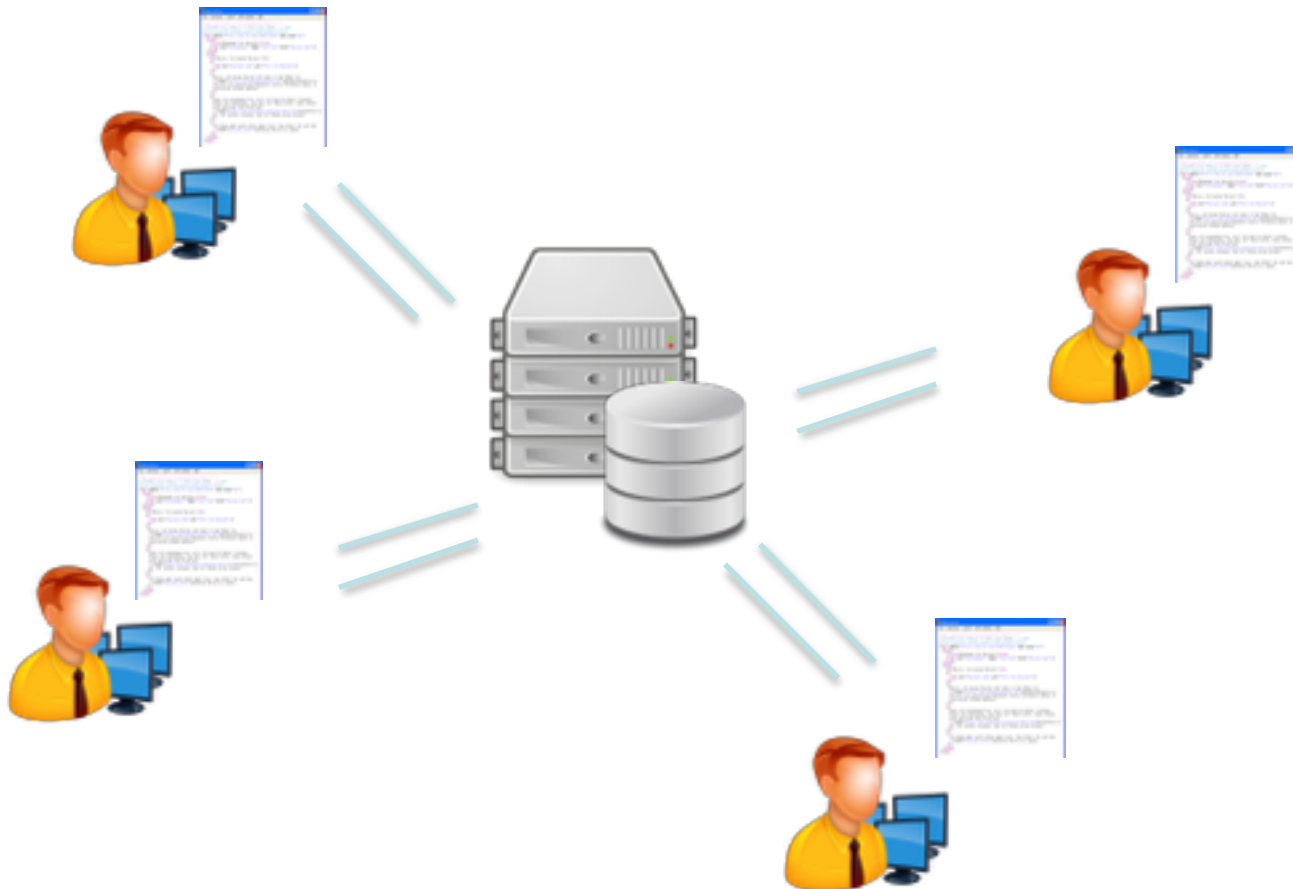
# Versionsmanagement

- Zentrale Verwaltung des Quellcodes (z.B. auf Server)
- Zusammenführen unterschiedlicher Veränderungen
- Erkennen und Behandeln von Konflikten



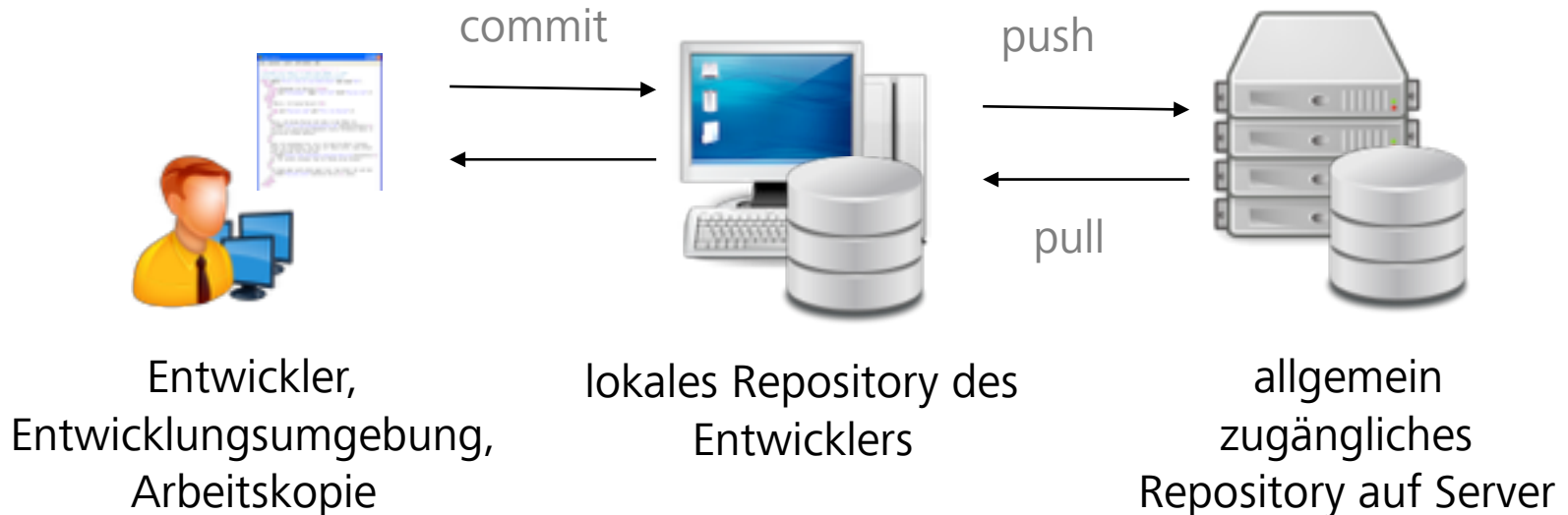
# Versionsverwaltung

- Zentraler (Server-)Ansatz
- Server synchronisiert die lokalen Kopien aller Entwickler



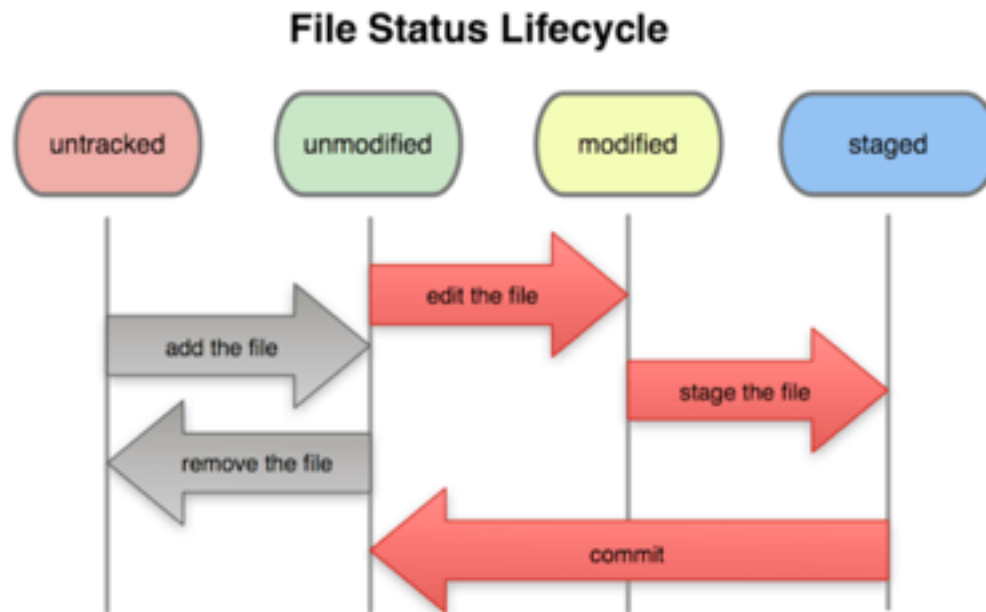
# Versionsverwaltung

- mittlerweile üblich
  - dezentrale Versionsverwaltung
  - z.B. Mercurial, GIT



# Git

- Verteiltes Versionsmanagement-System
- Idee:
  - jede(r) EntwicklerIn hat lokales Repository
  - verteilte Repositories werden synchronisiert



Quelle: <https://git-scm.com/>, abgerufen am 1.6.16



# **GIT-Repositories**

- Eigenes Netzlaufwerk
- Bitbucket (<https://bitbucket.org/>): Kostenlos für private Nutzung
- Github (<https://github.com/>): Kostenlos bei öffentlichen Repositories
- Sourceforge (<http://sourceforge.net/>): Open Source Projekte

# Zusammenfassung

- Organisation
- Dateien und I/O
  - Dateiinformationen
  - Dateien Lesen und Schreiben
  - Try-With-Resources
- Formate
  - Datenformate: XML
  - XML-Dokumente in Java verarbeiten
- Versionsmanagement, Git