LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Skript zur Vorlesung:

Einführung in die Programmierung

WiSe 2009 / 2010

Prof. Dr. Christian Böhm
Annahita Oswald
Bianca Wackersreuther

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Was ist Informatik?

Franz.: *Informatique* (= information + mathématique)

Engl.: *Computer Science* (neuerdings auch *Informatics*)

- DUDEN Informatik:
  „Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Computern.“
- Gesellschaft für Informatik (GI):
  „Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung und Übermittlung von Informationen.“
- Association for Computer Machinery (ACM):
  „Systematic study of algorithms and data structures.“

Primäres Lernziel: Nicht die Komponenten eines Computers, Anwendungsprogramme oder Programmiersprachen sondern *Prinzipien und Techniken zur Verarbeitung von Information.*

Lehr- und Lernformen in der Informatik

Informatik an einer Universität ist ein *wissenschaftliches Studium*.

Was bedeutet dies?

- Neue Denkweisen
- Grundlagenorientiert
- Steilerer Anstieg, höheres Niveau
- Angebote statt Zwang und Anwesenheitspflicht

Lernziel Nummer 1 an einer Universität für jedes Studienfach

Eigenverantwortung

Eigenverantwortung

Was bedeutet Eigenverantwortung?

- Teilweise ist der Stoff sehr schwierig/theoretisch (noch mehr in den Mathematik-Vorlesungen)
- Trotzdem: Im Gegensatz zur Schule ist nicht mehr der Lehrer für Ihren Lernerfolg verantwortlich.
- Das Erarbeiten des Stoffes ist *Ihre eigene Verantwortung.*

Zusätzlich zum Besuch von Vorlesungen und Übungen erforderlich:

- Recherche von Literatur zum Verstehen des Stoffes
- Selbständiges Lösen der Aufgaben
- Vor Besuch der Vorlesung Sichtung des Materials

Grundlagenorientiertes Studium

Informatik hat sich u.a. aus der Mathematik entwickelt.

Viele Vorgehensweisen aus der Mathematik entlehnt:

- Definition, Satz, Beweis…
- Ein Computer hat die Aufgabe, *Berechnungen* durchzuführen.
- Er rechnet nicht nur mit Zahlen, sondern mit *Informationen*.
  - z.B. *Wörter* aus Websites bei einer Google-Recherche.

Analysis, Algebra, Statistik und Numerik sind wichtige Vorlesungen.

Was ist Informatik?

In dieser Vorlesung lernen wir vor allem:

- Einführung in die Programmierung
- Anhand der Programmiersprache Java
- Entwerfen von einfachen Algorithmen
- Einfache Datenstrukturen (Darstellungsmöglichkeiten für Daten)
- Grundlagen der Objektorientierung

In späteren Vorlesungen werden wir beispielsweise lernen

- Andere Paradigmen, z.B. das „funktionale Programmieren“
- Software-Engineering, also Software-Entwicklung in Teams
- Hardware-Grundlagen der Informatik
- Analysemethoden für Probleme, Algorithmen, Programme… uvm

Teilgebiete der Informatik

Meist unterscheidet man drei Säulen:

- Theoretische Informatik
- Praktische Informatik
- Technische Informatik

Sowie angewandte Informatik, z.B. Bio- und Medieninformatik.

Zu allen Säulen werden Sie viele Vorlesungen hören.

Zur praktischen Informatik gehören Gebiete wie:

- Programmiersprachen
- Software-Technik
- Datenbanksysteme

Teilgebiete der Informatik

Fragestellungen der theoretischen Informatik

- Welche Probleme sind überhaupt vom Computer berechenbar? („Unentscheidbares“ Problem: Terminiert ein Algorithmus?)
- Wie effizient ist ein Algorithmus?
- Wie effizient kann ein beliebiger Algorithmus zur Lösung eines Problems überhaupt sein?

Gebiete der technischen Informatik

- Rechnerarchitektur (Aufbau der Prozessoren usw.)
- Betriebssysteme (z.B. wie wird der Speicher verwaltet?)
- Rechnernetze und Mobilfunknetze
- Compiler und maschinennahe Programmierung

Algorithmus

Abgeleitet von *Muhammad ibn Musa al-Chwarizmi* (lat. al-Gorithmus), persischer Mathematiker 9. Jh. n. Chr.

- Algorithmus ist eine Handlungsvorschrift, um ausgehend von bestimmten Vorbedingungen ein bestimmtes Ziel zu erreichen
- Zentraler Begriff der Informatik

Wichtige Inhalte des Informatikstudiums:

- Entwicklung von Algorithmen
- Analyse von Algorithmen (Korrektheit, Laufzeit, Eigenschaften)
- Oft weniger wichtig: Umsetzung in Programmiersprachen

Algorithmus

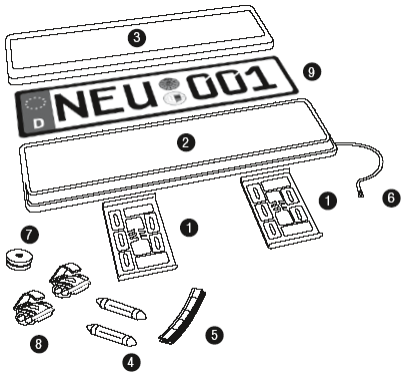
Typische Elemente, die in Algorithmen enthalten sind:

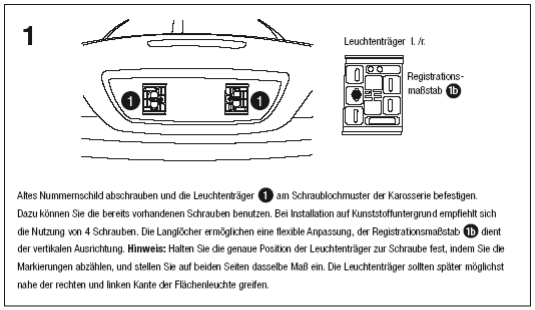
- Schrittweises Vorgehen
- Fallunterscheidungen und Wiederholungen („Schleifen“)
- Zur Berechnung erforderliche Informationen in „Variablen“
- Eingabe: Parameter der Problemstellung
  - z.B. das Wort nach dem im Web gesucht wird
- Ausgabe: Lösung des Problems
  - Liste der Treffer-Websites geordnet nach statist. Relevanz

Interaktive Programme (Word/Excel) sind selbst keine Algorithmen, setzen aber zur Lösung von Teilproblemen (z.B. Spelling Checker) Programmteile ein, die auf Algorithmen beruhen.

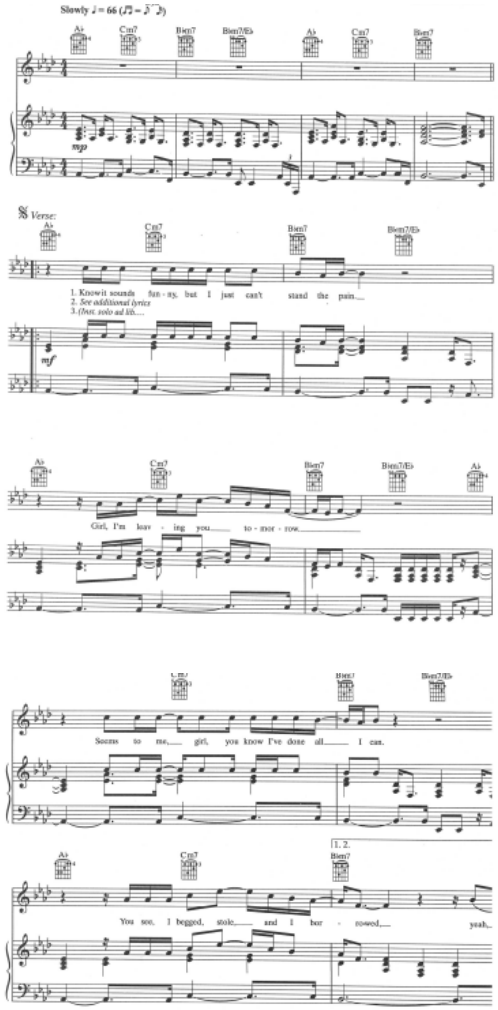
Beispiele aus alltäglichem Leben

Montageanleitung:

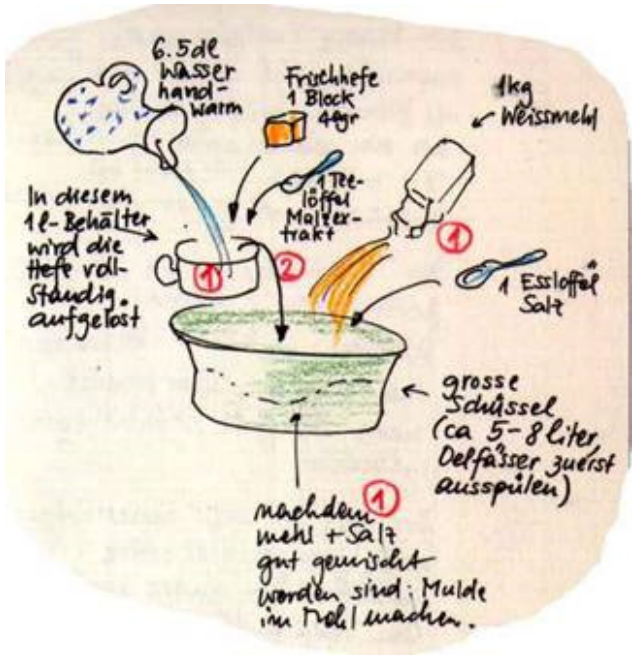




Notenpartitur:



Kochrezept:



Beispiel: GGT-Algorithmus

Algorithmus zur Berechung des größten gemeinsamen Teilers (GGT) von zwei natürlichen Zahlen *a* und *b*.

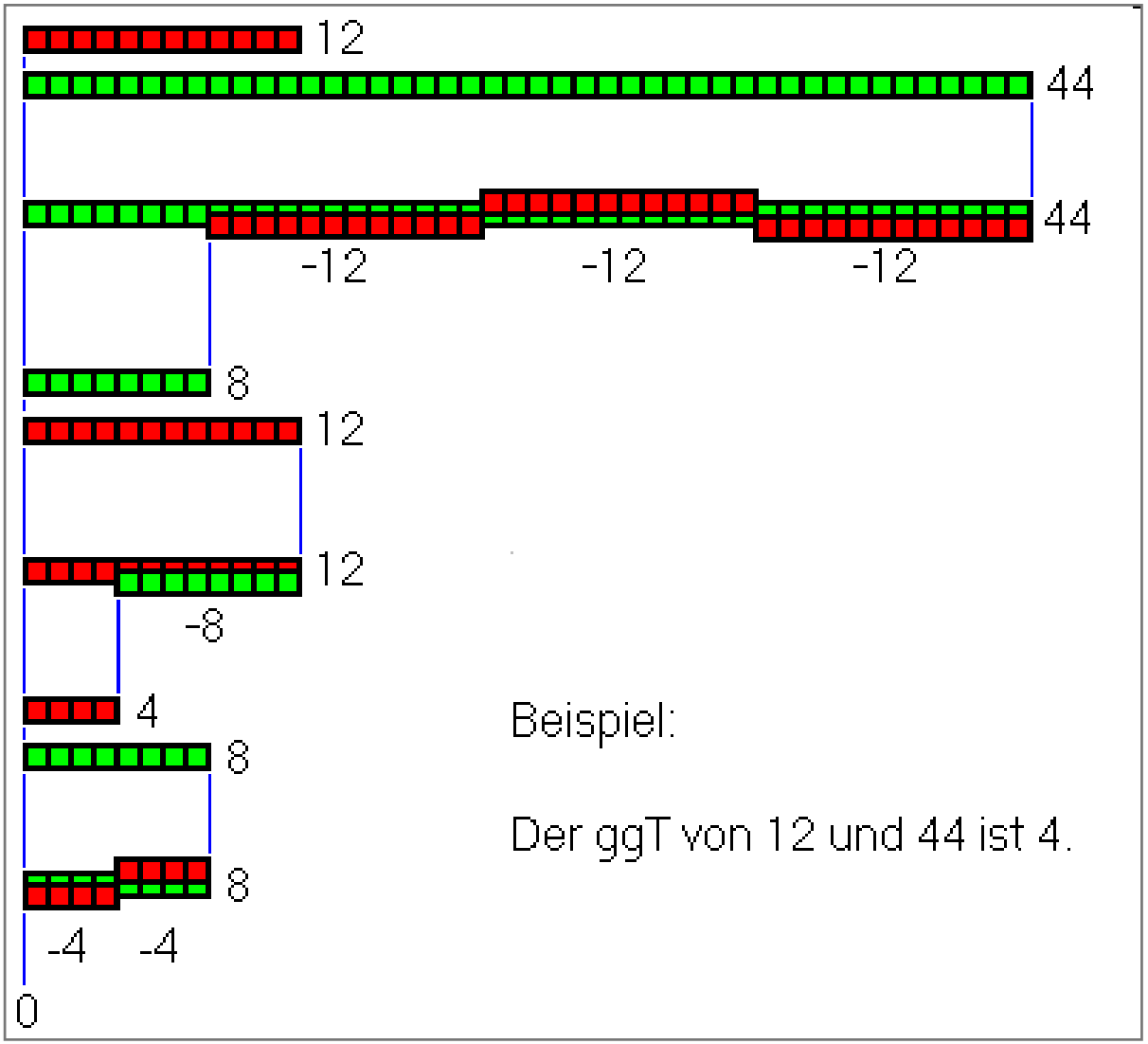
Beschrieben von *Euklid von Alexandria* (300 v.Chr.)
im Werk „Die Elemente“.

Algorithmus *EUKLID*
Eingabe: $a, b \in \mathbb{N}$

```
1    solange b ≠ 0
2        wenn a > b
3            dann a := a − b
4            sonst b := b − a
```

Ausgabe: *a*

Beispiel: GGT-Algorithmus



Beispiel: GGT-Algorithmus

Es ergeben sich folgende interessante Fragen:

- Ist der Algorithmus *korrekt*?
  - Also: berechnet er wirklich den GGT aus (a,b)?
  - Wieso funktioniert dieser Algorithmus überhaupt?
- Ist der Algorithmus *vollständig*?
  - Kann ich wirklich jedes Paar (a,b) eingeben?
- Ist der Algorithmus *terminierend* (evtl. Endlosschleife)?
- Ist der Algorithmus *effizient*?
  - Wie viel Speicher benötigt er?
  - Wie viel Zeit (d.h. Verarbeitungsschritte) benötigt er?
  - Wie hängt das von der Eingabe (a,b) ab?

Wir müssen solche Aussagen immer mathematisch *beweisen*.

Definition Algorithmus

Ein *Algorithmus* ist ein Verfahren zur Verarbeitung von Daten mit einer präzisen, endlichen Beschreibung unter Verwendung effektiver, elementarer Verarbeitungsschritte:

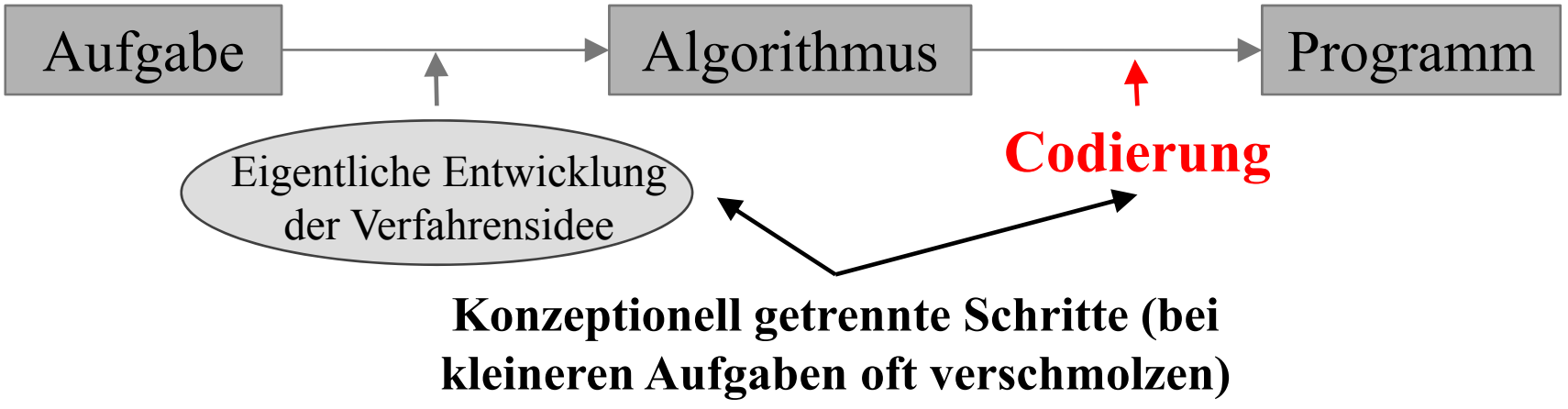
- **Daten**:
  Die Repräsentation und der Wertebereich von Eingabe und Ergebnissen müssen eindeutig definiert sein.
- **Präzise, endliche Beschreibung**:
  Die Abfolge von Schritten muss in einem endlichen Text in einer eindeutigen Sprache genau festgelegt sein.
- **Effektiver Verarbeitungsschritt**:
  Jeder einzelne Schritt ist tatsächlich ausführbar.
- **Elementarer Verarbeitungsschritt**:
  Jeder Schritt ist entweder eine Basisoperation, oder ist selbst durch einen Algorithmus spezifiziert.

Aufgabe des Informatikers

Zentrale Aufgabe des Informatikers: Entwicklung von Algorithmen (und oft auch deren Realisierung auf dem Rechner als Programm)

Programm: Formale Darstellung eines Algorithmus (oder) mehrerer in einer Programmiersprache

Programmiersprache: Formale (eindeutige) Sprache, stellt insbesondere elementare Verarbeitungsschritte und eindeutig definierte Datentypen für die Ein-/Ausgabe zur Verfügung.



Imperative und andere Paradigmen

Algorithmen werden meist in einer Form „Schritt für Schritt“ definiert.
Die meisten Programmiersprachen folgen diesem Prinzip.

Grundelemente der *imperativen* (befehlsbezogenen) Programmierung:

- Fallunterscheidung (wenn – dann – sonst)
- Wiederholung (Schleife: solange Bedingung gilt…)
- Variablen, deren Inhalt durch den Algorithmus verändert werden kann.

Es gibt auch andere Programmierprinzipien, z.B. Logik-basierte:
„Welche Eigenschaften soll das Ergebnis meines Algorithmus haben?“

Objektorientierung

Idee: Der Computer rechnet nicht nur mit Zahlen, Zeichenfolgen, etc.

- also mit den *primitiven* (d.h. eingebauten) *Datentypen*

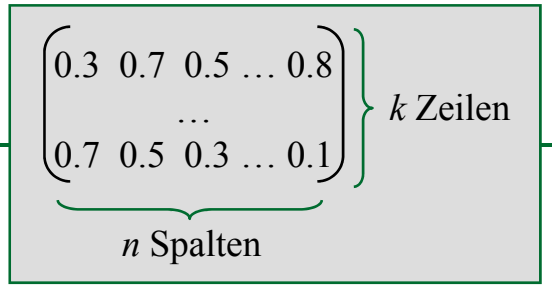
Sondern er rechnet auch mit *komplexen Objekten*.

Die Objekte sind gemeinsam mit den Operationen, die mit/auf ihnen berechnet werden können in einer sog. *Klasse* definiert:

- Klassen für häufig verwendete Objekte kann man z.B. kaufen (z.B. Klassenbibliotheken für Matrixrechnung, Sortieren, etc.)
- Klassen für eigene Algorithmen/Programme kann man selbst definieren/programmieren (z.B. Verwaltung der eigenen CD-Sammlung: Klassen für Künstler, CDs, …)

Durch die Definition von neuen Klassen werden die Fähigkeiten des Computers gewissermaßen Schritt für Schritt erweitert.

Objektorientierung



Beispiel für eine Anwendung zur Matrixrechnung:

- Es wird eine Klasse Matrix definiert.
- In dieser Klasse wird definiert, aus welchen Komponenten eine $(k \times n)$-Matrix besteht…:
  - Den Dimensionsangaben k und n sowie
  - einer Reihe von n mal k vielen Elementen der Matrix
- … und welche Operationen auf Matrizen ausgeführt werden können und wie das gemacht wird (d.h. der Algorithmus)
  - z.B. Multiplizieren von zwei Matrizen
  - Multiplizieren einer Matrix mit einer Zahl
  - Invertieren einer Matrix
  - usw.

Objektorientierung

Es ergibt sich folgende Rollenverteilung:

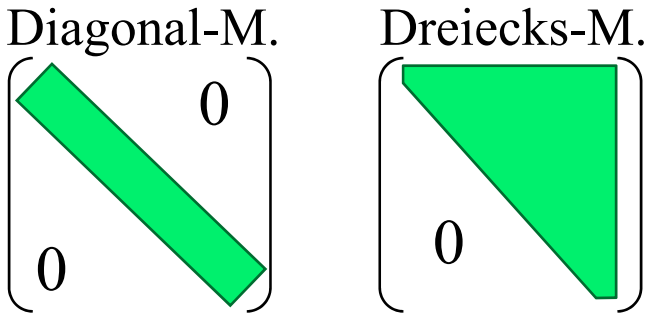
- Der *Anwender* der Klasse Matrix kann:
  - Matrizen erzeugen und z.B. in Variablen speichern:
    Matrix m_1 = new Matrix ( );
  - Operationen wie das Multiplizieren von Matrizen anwenden:
    m_1.multiply (m_2) ;
- Der *Implementierer* der Klasse Matrix muss:
  - Definieren, aus welchen Elementen eine Matrix besteht.
  - Definieren, wie man eine neue Matrix erzeugt.
  - Definieren, wie die Operationen funktionieren (d.h. z.B. den Matrix-Invertierungs-Algorithmus in einer objektorientierten Programmiersprache programmieren).

Objektorientierung/Vererbung

Oft gibt es Anwendungen, wo zueinander ähnliche Klassen auftreten.

Dann sollen diese Verwandtschaftsbeziehungen modelliert werden.

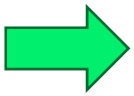
Beispiel:



- Es gibt spezielle Matrizen, z.B.
  - Diagonalmatrizen,
  - Dreiecksmatrizen,
  - Vektoren (= $(1 \times n)$ -Matrix, bzw. $(k \times 1)$ -Matrix).
- Man kann dann sagen, die Klasse Diagonalmatrix ist eine *Subklasse* der Klasse Matrix:
  - Die Operationen (z.B. invertieren) können *vererbt* werden
  - oder neu definiert werden (z.B. weil für Diagonalmatrizen effizientere Algorithmen zum Invertieren existieren).

Vorteile der Objektorientierung

- Größter Vorteil der Objektorientierung: Wiederverwendbarkeit
  Durch Vererbung ist es möglich, Attribute und Verhaltensweisen an eine Subklasse zu vererben, und dadurch vorhandenen Code wiederzuverwenden.
- Bessere Erweiterbarkeit und Wartbarkeit von Programmcode
- Programmcode wird übersichtlicher, da einzelne Aufgaben leichter in Module aufgeteilt werden können und unabhängig voneinander erstellt werden.



Objektorientierte Programmierung hat sich durchgesetzt

Java

- Objektorientierte Programmiersprache
- Plattformunabhängig: Übersetzung in Virtuelle Maschine (JVM)
- Netzwerkfähig, nebenläufig
- Sicherheitsaspekt in der Entwicklung der Sprache wichtig

Nachteile:

Laufzeithandicap durch Interpretation (JVM), wird aber stetig verbessert

Vorteile:

- Plattformunabhängigkeit
- Verteilte Anwendungen,Web-Anwendungen
- Rechnerunabhängigkeit von Graphikanwendungen

Grober Aufbau eines Java-Programms

- Ein Java-Programm besteht aus Klassen und Schnittstellen.
- Eine Klasse besteht aus:
  - Klassenvariablen: Beschreiben Eigenschaften aller Objekte dieser Klasse.
  - Attributen (fields, Instanzvariablen): Beschreiben den Zustand eines Objekts.
  - Statischen Methoden: Prozeduren einer Klasse, unabhängig vom Zustand eines Objekts.
  - Objekt-Methoden: Operationen, die ein Objekt ausführen kann, abhängig vom Zustand des Objektes.
  - Konstruktoren: Operationen zur Erzeugung von Objekten einer bestimmten Klasse.

⇨ Zunächst: Statische Elemente Später: Aspekte der Objektorientierung

Ein einfaches imperatives Java-Programm

Keine Angst!!!

All das lernen wir später noch genauer kennen. Für den Anfang merken wir uns: Ein einfaches imperatives Java-Programm besteht aus nur einer Klassendeklaration und einer Methode "main":

```
public class KlassenName
{
    public static void main(String[] args)
    {
    // Hier geht's los mit Anweisungen
    // (elementare Verarbeitungsschritte)
    // ...
    }
}
```

Die Textdatei, die den Java-Code enthält, heißt `KlassenName.java`, also genauso wie die enthaltene Klasse, mit der Endung `java`.

Beispiel: HelloWorld.java

```
public class HelloWorld
{
     public static void main(String[] args)
     {
          System.out.println("Hello, world!");
     }
}
```

Konventionen

In der Java Programmierung gibt es einige Konventionen. Deren Einhaltung erleichtert das Lesen von Programmen. Beispiele solcher Konventionen sind:

- Klassennamen beginnen mit großen Buchstaben.
  - `HelloWorld`
- Methodennamen, Attributnamen und Variablennamen beginnen mit kleinen Buchstaben.
  - Methoden: `main, println`
  - Klassenvariable: `out`
  - Variable: `args` (weder Instanz- noch Klassenvariable, sondern Parametervariable)
- Zusammengesetzte Namen werden zusammengeschrieben, jeder innere Teilname beginnt mit einem großen Buchstaben.
  - Klasse `HelloWorld`

Konventionen

Warum Konventionen?

- In der "Lebenszeit" eines Programms werden typischerweise 80% der Kosten von Wartung verursacht.
- Selten wird ein Software-Produkt dauerhaft vom ursprünglichen Autor gewartet.
- Code-Konventionen erleichtern die Lesbarkeit von Programmen und ermöglichen Programmierern, fremden (und eigenen!) Code schneller und besser zu verstehen.

Die von Sun empfohlenen Konventionen für Java finden Sie unter:

http://java.sun.com/docs/codeconv/

Momentan ist dort vieles vielleicht noch unverständlich, aber schlagen Sie im Lauf des Semesters immer mal wieder dort nach!

Compiler und Interpreter

- Ein Programm einer höheren Programmiersprache ist für Menschen lesbar, folgt aber einer festen Syntax (Grammatik), so dass *elementare Verarbeitungsschritte* klar definiert sind.
- Ein Übersetzer (*Compiler)* erzeugt aus der Programmdatei Maschinencode, der von Rechnern ausgeführt werden kann.
  - Der Compiler kennt die Syntax der Programmiersprache und bemerkt *syntaktische* Fehler (Verstöße gegen die Grammatik).
  - Nur, wenn das Programm syntaktisch korrekt ist, wird auch Maschinencode erzeugt. Andernfalls reagiert der Compiler mit Fehlermeldungen.
  - Achtung: Der Compiler erkennt nur *syntaktische,* nicht aber *semantische* Fehler, d.h. Fehler durch die das Programm eine andere *Bedeutung* annimmt als vom Programmierer beabsichtigt.

Compiler und Interpreter

- Bei vielen Sprachen (z.B. C/C++) erzeugt der *Compiler* plattformabhängigenMaschinencode (kann nur auf bestimmten Rechnerarchitekturen/Betriebssystemen ausgeführt werden).
- Sogenannte Skript-Sprachen (z.B. Perl, PHP, SML) werden nicht kompiliert, sondern von einem plattformspezifischen *Interpreter* interpretiert, wobei auch eine Syntaxprüfung durchgeführt werden muss.
- Vorteil: Die Programme sind plattformunabhängig.
- Nachteil: Der Sourcecode bleibt unübersetzt und wird bei jeder Ausführung des Programmes von neuem interpretiert.

Best of both worlds: Kompilierung und Plattformunabhängigkeit

- Plattformunabhängigkeit eines Java-Programmes wird durch einen Kompromiss erreicht:
  - Der Sourcecode wird durch einen Compiler übersetzt in Bytecode, der plattformunabhängig verwendet werden kann.
  - Der Bytecode ist also bereits syntaktisch überprüft, kann aber nicht direkt von einem Computer verarbeitet werden.
  - Bytecode wird von einer *virtuellen Maschine* ausgeführt (interpretiert).
  - Diese Interpretation muss aber keine syntaktische Prüfung mehr vornehmen.
  - Die virtuelle Maschine gibt es in verschiedenen Versionen für verschiedene Plattformen (JVM = Java Virtual Machine, Teil des JRE = Java Runtime Environment).

Übersetzung in Bytecode

- Aus einer Textdatei `KlassenName.java` erzeugt der Java Compiler `javac` eine Binärdatei `KlassenName.class`. Beispiel:

```
javac HelloWorld.java
```

  erzeugt die Binärdatei `HelloWorld.class`.
- Die Binärdatei `KlassenName.class` enthält den Bytecode für die JVM.
- Der Compiler `javac` ist Teil des JDK (= Java Development Kit). Das JDK enthält JRE, Sie benötigen also das JDK für die Übungen zu dieser Vorlesung.

Ausführen des Programms

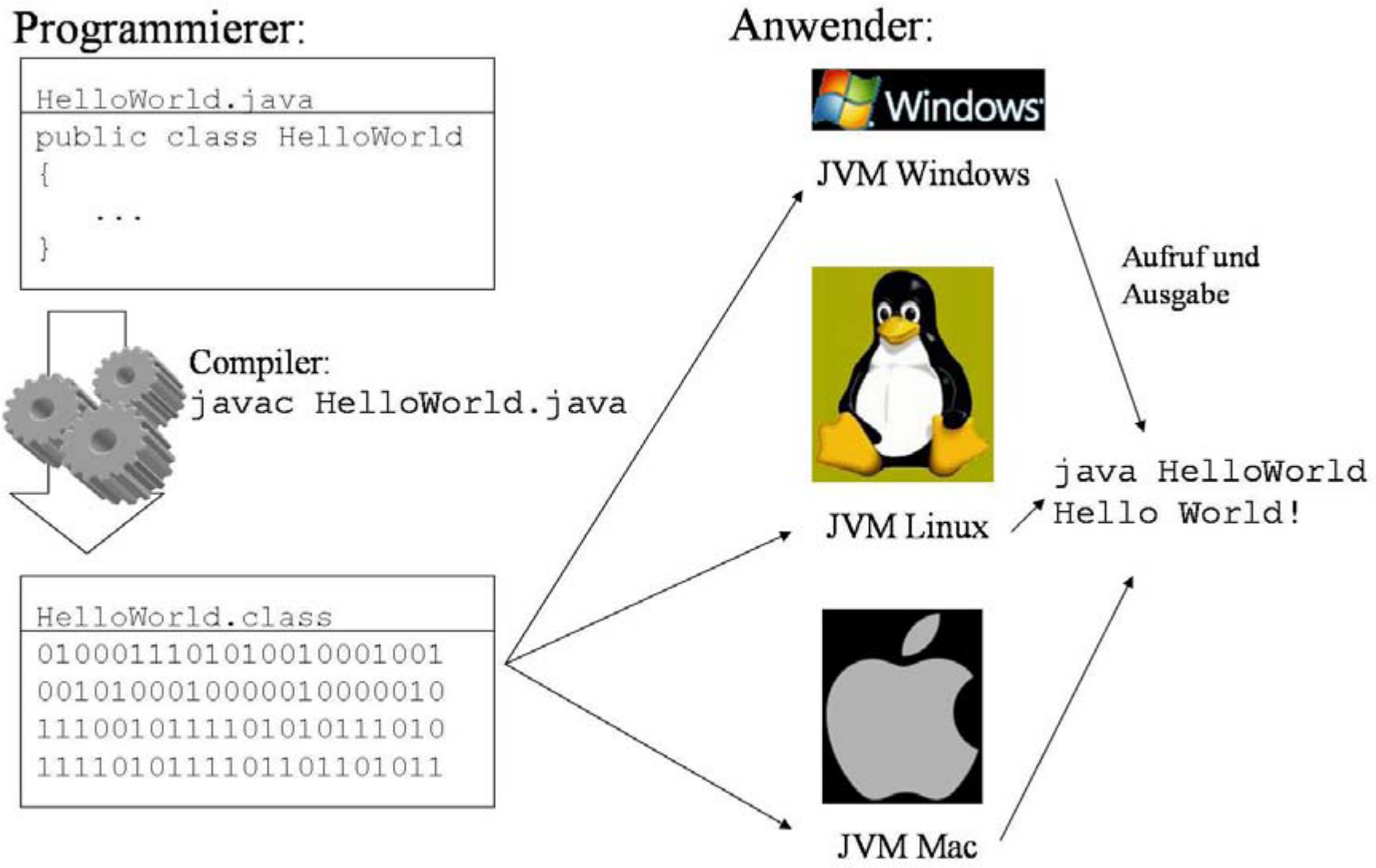
Die Binärdatei `KlassenName.class` wird der JVM übergeben und von dieser ausgeführt (interpretiert). Durch den Aufruf `java KlassenName` wird die `main`-Methode der Klasse `KlassenName` aufgerufen.

Beispiel:

```
java HelloWorld
```

gibt “Hello, World!” auf dem Bildschirm aus.

Übersetzung und Ausführung



Prinzip: Programmcode und Kommentar gehören zusammen

"The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle, and counterproductive in practice. Instead, documentation must be regarded as an integral part of the process of design and coding."

C. A. R. Hoare (Turing-Preisträger):
Hints on Programming Language Design,
1973

C. A. R. Hoare, *1934
Erfinder von Quicksort, Hoare Logik,
Strukt. Programmierung,
CSP, Occam
Turing-Preis 1980

Verschiedene Arten von Kommentaren

Es gibt in Java drei Arten von Kommentaren:

- **Einzeilige Kommentare** beginnen mit `//` und enden am Ende der aktuellen Zeile.
- **Mehrzeilige Kommentare** beginnen mit `/*` und enden mit `*/`
- **Dokumentationskommentare** beginnen mit `/**` und enden mit `*/` und können sich ebenfalls über mehrere Zeilen erstrecken.

Kommentare derselben Art sind nicht schachtelbar. Ein Java-Compiler akzeptiert aber einen einzeiligen innerhalb eines mehrzeiligen Kommentars.

Dokumentationskommentare dienen dazu, Programme im Quelltext zu dokumentieren. Sie werden in den mit dem Befehl `javadoc` erzeugten Report mit aufgenommen.

Beispiel: Die Klasse `HelloWorld` dokumentiert

```
/**
* HelloWorld Klasse um eine einfache Benutzung einer java-Klasse zu illustrieren.
*
* Diese Klasse dient dem Anzeigen des Strings "Hello, world!" auf dem Bildschirm
*
* @author Christian Böhm
*/
public class HelloWorld
{
     /**
      * Die main-Methode wird automatisch aufgerufen, wenn die Klasse mit
      * <code>java HelloWorld</code> aufgerufen wird.
      *
      * Die Methode main druckt "Hello, world!" auf die Standard-Ausgabe.
      *
      @param args Array mit Parametern - wird von dieser Methode nicht
      * verwendet.
      */
      public static void main(String[] args)
      {
            // Ausgabe von "Hello World!" auf die Standard-Ausgabe
            System.out.println("Hello World!");
      }
}
```

Erzeugen der Dokumentation

Mit dem Befehl

```
javadoc HelloWorld.java
```

wird automatisch eine Beschreibung der Klasse `HelloWorld` erzeugt und in die Datei

```
HelloWorld.html
```

geschrieben.

Die Klassenbeschreibung wird eingebettet in eine organisierte Menge von html-Dokumenten.

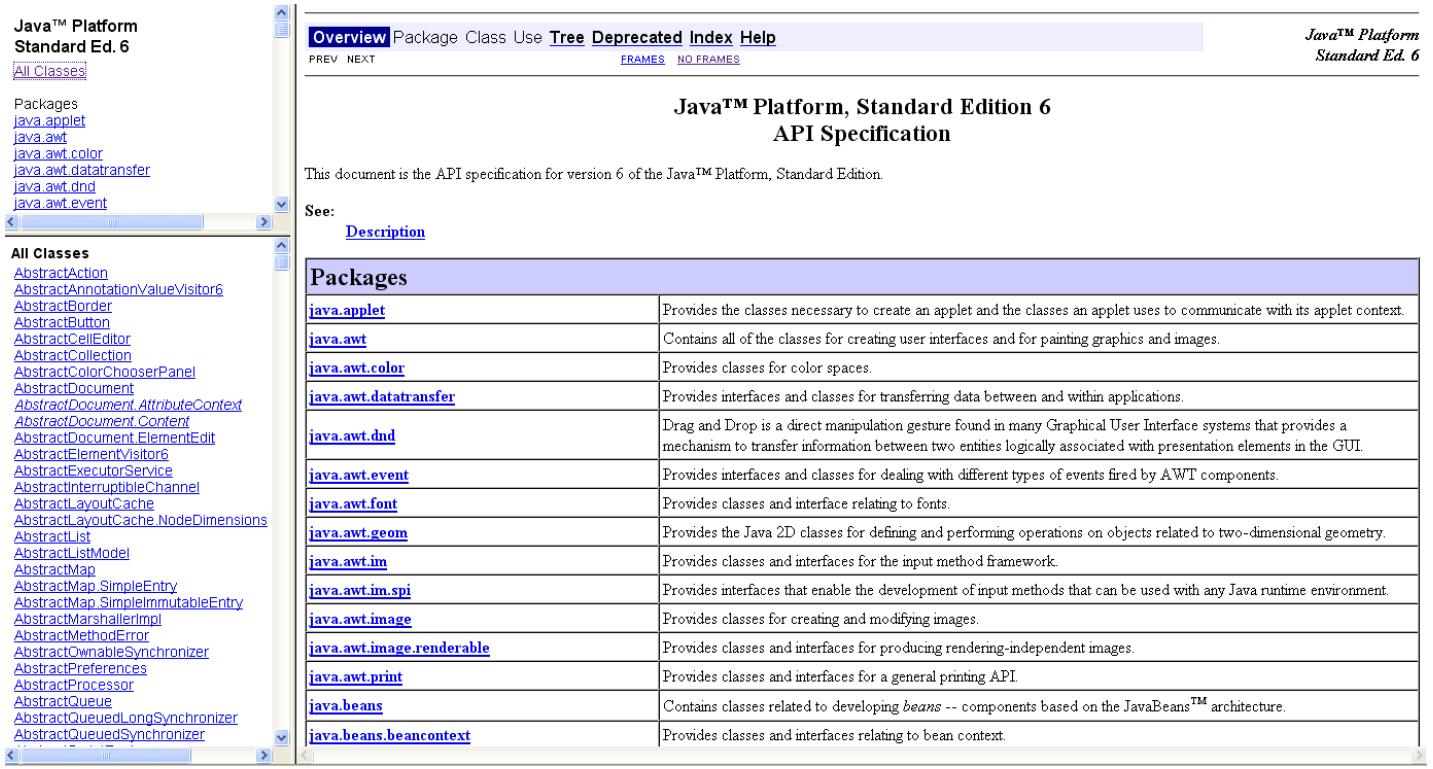
Diese Dokumentation kann auch für viele Klassen gleichzeitig erfolgen (`javadoc *.java`).

Spezielle Variablen (tags) für javadoc

Die durch `@` eingeleiteten Elemente in einem Dokumentations-kommentar haben eine besondere Bedeutung, z.B.:

- `@see` für Verweise
- `@author` für Name des Autors / Namen der Autoren
- `@version` für die Version
- `@param` für die Methodenparameter
- `@return` für die Beschreibung des Ergebnisses einer Methode

Java Standard Edition API Dokumentation



Auch die Bibliothek der Standard Edition ist mit javadoc erzeugt:

http://java.sun.com/javase/6/docs/api/

Für das fortgeschrittene Programmieren mit Java ist diese API Doc ein sehr wichtiges Hilfsmittel.