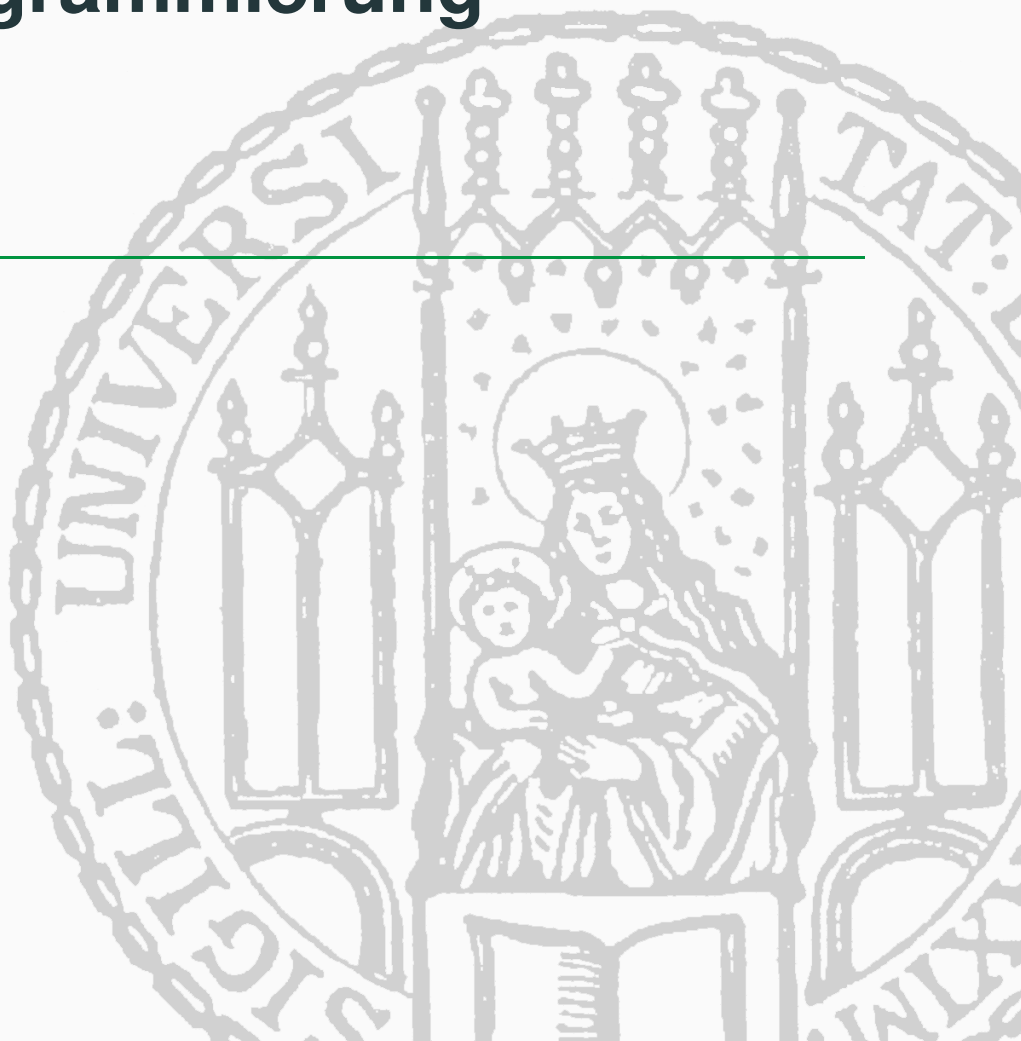
Einführung in die Programmierung

Teil 1: Einführung

Prof. Dr. Peer Kröger,
Florian Richter, Michael Fromm

Wintersemester 2018/2019



Übersicht

1. Was ist Informatik?

2. Die Programmiersprache Java

Kapitel 1: Was ist Informatik?

Was ist Informatik?

Franz. *informatique* (= information + mathématiques)

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.

Informatik an der Universität

Primäres Lernziel:

Nicht die Komponenten eines Computers oder die Sprachen zu seiner Programmierung sondern die *Prinzipien und Techniken zur Verarbeitung von Information.*

Informatik an der Universität

Informatik an einer Universität ist ein *wissenschaftliches* Studium:

- Neue Denkweisen, Abstraktion
- Grundlagenorientiert, Theorie vor Praxis
- Steilerer Anstieg, höheres Niveau
- Angebote statt Zwang und Anwesenheitspflicht

Zusätzlich: Wesentliches Lernziel an einer Universität für jedes Studienfach

Eigenverantwortung

- Was bedeutet Eigenverantwortung?
  - Teilweise ist der Stoff sehr schwierig und eher theoretisch abstrakt als praktisch konkret (noch mehr in den Mathematik-Vorlesungen)
    Wenn Sie der Meinung sind, schon ganz gut programmieren zu können, haben Sie daher nicht notwendigerweise einen Vorteil, denn es geht nicht darum, wer „der beste Java-Programmierer“ ist.
  - Im Gegensatz zur Schule ist jetzt aber nicht mehr der Lehrer für Ihren Lernerfolg verantwortlich. (ach nein? wer denn dann????)
  - Das Erarbeiten des Stoffes ist Ihre *eigene Verantwortung*. (ach so)

Ein letztes Mal der erhobene Zeigefinger

- Zusätzlich zum Besuch von Vorlesungen und Übungen meist erforderlich:
  - Recherche von Literatur zum Verstehen des Stoffes
  - Selbständiges Lösen der Aufgaben
  - Vor Besuch der Vorlesung: Sichtung des Materials
- Aber niemand (außer Sie selbst) wird das von Ihnen einfordern.

Sie sollen Ihr Studium genießen, aber denken Sie dran: in “work hard, play hard” kommt “work” *vor* “play”!!!

Informatik und Mathematik

- 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.

- In dieser Vorlesung lernen wir u.a.:
  - Einführung in die Programmierung und Programmierparadigmen
  - Anhand der Programmiersprache Java[1]
  - Entwerfen von einfachen (hauptsächlich *imperativen*, teilweise *funktionalen*) Algorithmen
  - Einfache Datenstrukturen (Darstellungsmöglichkeiten für Daten)
  - Grundlagen der *objektorientierten* Modellierung und Programmierung

[1] *ABER: dies ist kein Java-Programmierkurs!!!*

- In späteren Vorlesungen werden wir beispielsweise lernen
  - Andere Paradigmen, z.B. das *funktionale* oder das *logische* Programmieren
  - Software-Engineering, also Software-Entwicklung in Teams
  - Hardware-Grundlagen der Informatik
  - Analysemethoden für Probleme, Algorithmen, Programme
  - uvm.

Der Algorithmus-Begriff

Begriff: *Algorithmus*

- Zentraler Begriff der Informatik
- Grundlage jeglicher maschineller Informationsverarbeitung
- Systematische, „schematisch“ („automatisch“, „mechanisch“) ausführbare Verarbeitungsvorschrift

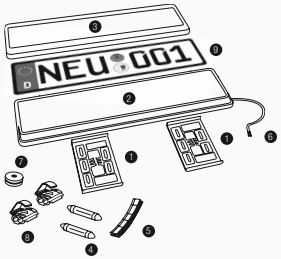
Wichtige Inhalte des Informatikstudiums:

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

Unser Alltag ist von Algorithmen geprägt . . .

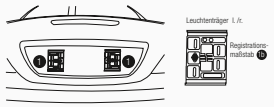
Montageanleitung

Einzelteile



❶ Leuchtenträger
❷ Flächenleuchte
❸ Abdeckrahmen
❹ Lastwiderstände
❺ Klebemasse zur Vibrationsdämpfung
❻ Kabel
❼ Kabeldurchführung
❽ Abzweigklemmen
❾ Nummernschild (nicht in Packung enthalten)

1

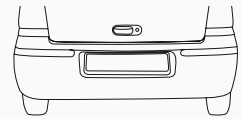


Altes Nummernschild abschrauben und die Leuchtenträger ❶ am Schraublochmuster der Karosserie befestigen. Dazu können Sie die bereits vorhandenen Schrauben benutzen. Bei Installation auf Kunststoffuntergrund empfiehlt sich die Nutzung von 4 Schrauben. Die Langlöcher ermöglichen eine flexible Anpassung, der Registrationsmaßstab ⓫ dient der vertikalen Ausrichtung. **Hinweis:** Halten Sie die genaue Position der Leuchtenträger zur Schraube fest, indem Sie die Markierungen abzählen, und stellen Sie auf beiden Seiten dasselbe Maß ein. Die Leuchtenträger sollten später möglichst nahe der rechten und linken Kante der Flächenleuchte greifen.

2a

Montage bei Kennzeichen in der Heckklappe: Die Innenverkleidung der Heckklappe so weit entfernen, dass Sie von innen an den Kennzeichenbereich gelangen. Die Demontage ist von Fahrzeug zu Fahrzeug unterschiedlich. Fragen Sie im Zweifel einen Fachbetrieb. Bohren Sie jetzt mit einem Metallbohrer (Ø 7 mm) ein Loch für die Kabel unter der vorhandenen Kennzeichenbeleuchtung. Bohrloch entgraten und mit Lackstift gegen Rostbefall schützen. **Hinweis:** Wählen Sie für das Bohrloch eine Stelle hinter dem Kennzeichen, die von innen zugänglich ist und später von der Flächenleuchte verdeckt wird.

2b



Montage bei Kennzeichen im Stoßfänger: Bohren Sie mit einem Metallbohrer (Ø 7 mm) ein Loch für die Kabel unter der vorhandenen Kennzeichenbeleuchtung. Bohrloch entgraten und mit Lackstift gegen Rostbefall schützen. **Hinweis:** Wählen Sie für das Bohrloch eine Stelle hinter dem Kennzeichen, die von innen bzw. unten zugänglich ist und später von der Flächenleuchte verdeckt wird.

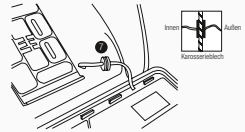
3

Platzieren Sie die umlaufende Randprägung des Nummernschildes ❾ und den Dichtungsring exakt übereinander. Abdeckrahmen ❸ durch **kräftiges Andrücken** auf der Flächenleuchte ❷ befestigen! Der Rahmen muss an 4 Stellen hörbar einrasten. Drehen Sie das Schild um und prüfen Sie, ob alle Rastnasen eingerastet sind. Der Abdeckrahmen ❸ kann zum Kennzeichenwechsel oder zur Reinigung gelöst werden. Hierzu biegen Sie mit einem Schraubendreher an den 4 dafür vorgesehenen Öffnungen den Kennzeichenrahmen nach außen.

4

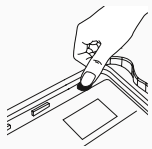
Nehmen Sie die Original-Kennzeichenbeleuchtung außer Betrieb (vorgeschrieben gemäß StVZO), indem Sie beide Leuchtmittel der Kennzeichenbeleuchtung durch die mitgelieferten Lastwiderstände ❹ ersetzen. Sollten bei Ihrem Fahrzeug andere Leuchtmittel eingesetzt werden, nehmen Sie die werkseitig angebrachte Nummernschildbeleuchtung durch Demontage der Leuchtmittel außer Betrieb.

5



Jetzt die mitgelieferte Kabeldurchführung ❼ mit Kabel montieren und in den Innenbereich führen. Prüfen Sie, ob die Aussparung der Kabeldurchführung fest im Karosserieblech sitzt.

6

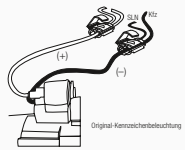


Nehmen Sie nun die Klebemasse zur Vibrationsdämpfung ❺ und bringen Sie diese auf der Rückseite des Kennzeichens an. Die genaue Menge bemessen Sie je nach Abstand zwischen Heckblech und Kennzeichen.

7

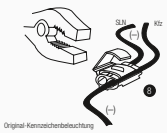
Richten Sie die Flächenleuchte ❷ auf den Leuchtenträgern horizontal aus. Jetzt kräftig aufdrücken (obere Kante zuerst) und **hörbar einrasten** lassen.

8



Verbinden Sie mit den mitgelieferten Abzweigklemmen ❽ die Kabel mit der Spannungsversorgung der ursprünglichen Kennzeichenbeleuchtung: 1 x mit „plus", 1 x mit „minus". Achtung: Nennspannung 12 V Gleichstrom, Nennleistung 2,9 Watt.

9



Pressen Sie beide Abzweigklemmen mit einer Zange fest zusammen. Führen Sie einen Funktionstest in abgedunkelter Umgebung durch. **Hinweis:** Aufgrund gesetzlicher Bestimmungen darf die Flächenleuchte bei Dämmerung und Dunkelheit nicht überstrahlen. Aus diesem Grund werden Sie unter Umständen bei Tageslichtbedingungen den Betrieb der Flächenleuchte nicht ohne weiteres erkennen. Anschließend die Innenverkleidung wieder montieren.

Algorithmus-Beispiele aus dem Alltag

Kochrezept für einen Afrikanischen Hühnertopf (von `www.chefkoch.de`)

- Zutaten: 1 Huhn (küchenfertige Poularde), 125 ml Hühnerbrühe, 8 Tomaten, 150 g Erdnussbutter, 2 Zucchini, Salz und Pfeffer, Rosmarin.
- Zubereitung:
  - Haut von Poularde abziehen, Huhn zerteilen, Hühnerteile abspülen und in einen breiten Topf legen.
  - Brühe angießen und langsam zum Kochen bringen.
  - Tomaten mit kochendem Wasser übergießen, enthäuten und unzerteilt zum Fleisch geben.
  - Zugedeckt bei geringer Hitze ca. 40 Minuten köcheln.
  - Nach 30 Minuten Garzeit Erdnussbutter einrühren, mit Salz, Pfeffer und Rosmarin abschmecken; Zucchini putzen und in kleine Würfel schneiden, Blüten und Stengelansatz entfernen; alles in den Topf geben.

Algorithmus-Beispiel aus der Mathematik

Berechnung des größten gemeinsamen Teilers (GGT) von zwei natürlichen Zahlen

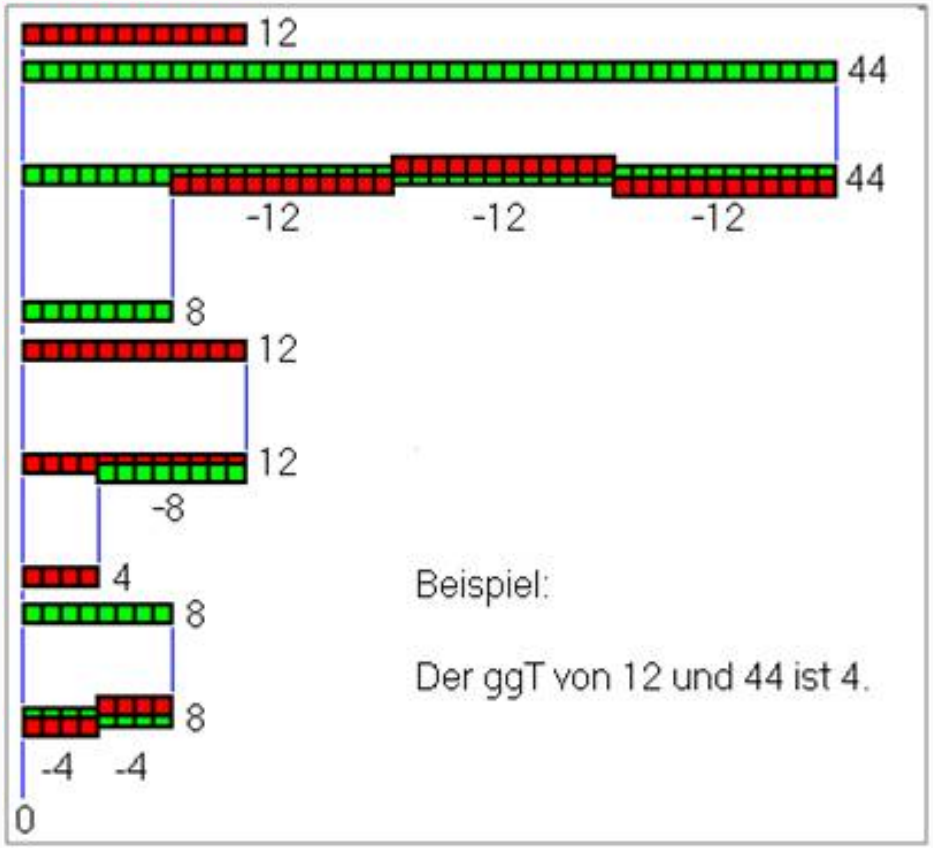
(Beschrieben von Euklid von Alexandria (300 v.Chr.) im Werk *Die Elemente*)

- Eingabe: zwei natürliche Zahlen a und b
- Berechnung:
  - Solange die Zahl $b \neq 0$ tue das folgende:
    - wenn $a > b$ dann ziehe b von a ab (d.h. $a := a - b$)
    - andernfalls ziehe a von b ab (d.h. $b := b - a$)
- Ergebnis ist die “übrig gebliebene” Zahl a

Z.B. ist der GGT von 12 und 44 die Zahl 4.

Algorithmus-Beispiel aus der Mathematik

Ablaufbeispiel für $a = 12$ und $b = 44$:



Eigenschaften von Algorithmen

Es ergeben sich folgende interessante Fragen:

- Ist der Algorithmus *korrekt*?
  - Berechnet er wirklich den GGT aus zwei natürlichen Zahlen a und b?
  - Wieso funktioniert dieser Algorithmus überhaupt?
- Ist der Algorithmus *vollständig*?
  - Kann ich wirklich jedes Paar von natürlichen Zahlen a und b eingeben?

Eigenschaften von Algorithmen

- Ist der Algorithmus *terminierend*?
  - Hört die Berechnung für jedes Paar von natürlichen Zahlen a und b immer auf?
- Ist der Algorithmus *effizient*?
  - Wie viel Speicher benötigt er?
  - Wie viel Zeit (d.h. Verarbeitungsschritte) benötigt er?
  - Wie hängt das von der Eingabe ab?

Um wirklich sicher zu sein, müssen solche Aussagen mathematisch *bewiesen* werden.

Algorithmus-Begriff

Definition (Algorithmus)

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

Diese Definition enthält u.a. drei wesentliche Aspekte, die wir uns noch etwas genauer vornehmen.

Aspekt 1: Daten

- Die Repräsentation und der Wertebereich von Eingabe und Ergebnissen müssen eindeutig definiert sein.
- Beispiel: im Algorithmus GGT ist vielleicht nicht ganz klar, ob die 0 eine natürliche Zahl ist oder nicht, d.h. eine erlaubte Eingabe bzw. Ausgabe darstellt (dies betrifft den Wertebereich, die Darstellung[2] ist tatsächlich “egal”).

[2] dazu später mehr ...

Aspekt 2: Präzise, endliche Beschreibung

- Die Abfolge von Schritten muss in einem endlichen Text in einer eindeutigen Sprache genau festgelegt sein.
- Beispiel: Montageanleitungen sind oft nicht eindeutig beschrieben/bebildert. .

Aspekt 3: Effektiver Verarbeitungsschritt

- Jeder einzelne Schritt ist tatsächlich durch die verarbeitende Einheit ausführbar.
- Verarbeitbar bedeutet dabei typw., dass es sich um eine Basisoperation der verarbeitenden Einheit handelt, oder selbst durch einen Algorithmus spezifiziert ist.
- Beispiel: in einem Kochrezept gibt es vielleicht Schritte, die ich nicht kann (weil ich die entsprechende Tätigkeit vielleicht noch nie gemacht habe, ...); dann kann ich das gesamte Rezept nicht kochen.

Typische Elemente in Algorithmen

Wir haben bereits in den ersten Beispielen ganz typische Elemente in Algorithmen gesehen:

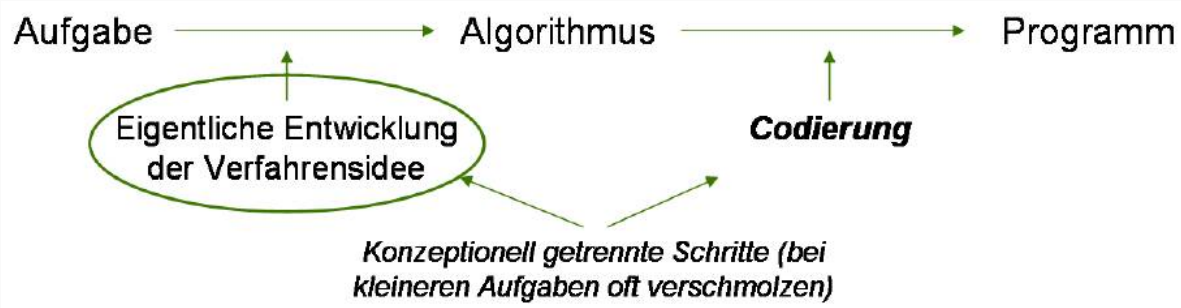
- Schrittweises Vorgehen bzw. sequentielles Abarbeiten von Verarbeitungsschritten
- Fallunterscheidungen und Wiederholungen („Schleifen“)
  wenn . . . **dann** . . . **sonst** und **solange** . . .
- Zur Berechnung erforderliche Informationen werden in „Variablen“ „zwischengespeichert“

Typische Elemente in Algorithmen

- Abstraktion von konkreten Eingabewerten:
  - Statt konkrete Werte werden Eingabeparameter (Parameter der Problemstellung) verwendet
  - z.B. die beiden Zahlen *a* und *b* deren GGT gesucht ist, anstelle nur den Algorithmus für die Berechnung des GGT von zwei konkreten Zahlen (z.B. 12 und 21) anzugeben
  - Dadurch wird eine *Klasse* von Problemen statt einem ganz speziellen Problem gelöst
  - z.B. einen Algorithmus zur Suche eines Wortes in Webseiten abstrahiert vom konkret gesuchten Wort; dadurch kann man den Algorithmus für beliebige Wörter benutzen
- Ausgabe: Lösung des Problems z.B. der GGT

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, die insbesondere elementare Verarbeitungsschritte und eindeutig definierte Datentypen für die Ein-/Ausgabe zur Verfügung stellt

Zentrale Aufgabe des Informatikers

LMU

In dieser Vorlesung

- Konzepte, Methoden und Techniken
  - zur Darstellung und Strukturierung von Daten
  - zur Entwicklung von Algorithmen
- *KEIN* Programmierkurs
- *ABER* Anwendung des Erlernten mit Java, denn Programmieren in einer konkreten Sprache gehört zum Handwerkszeug eines jeden Informatikers!!!

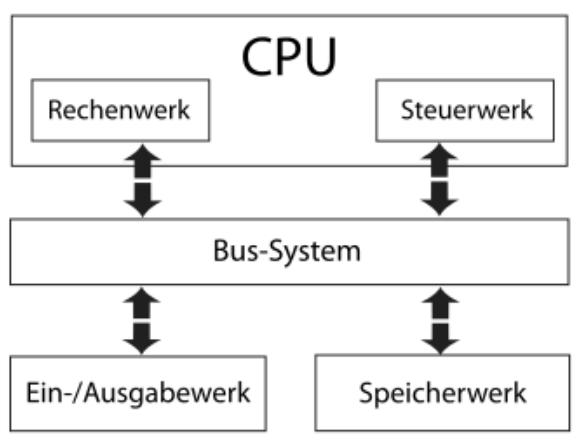
Teilgebiete der Informatik

- Theoretische Informatik:
  Theoretische Durchdringung und Grundlegung von Fragen und Konzepten der Informatik (z.B. “Berechenbarkeit” von Aufgabenstellungen, “Komplexität” von Aufgabenstellungen und Lösungen).
- Technische Informatik:
  Beschäftigung mit der *Hardware* (z.B. maschinengerechte Darstellung von Daten und Algorithmen).
- Praktische Informatik:
  Konstruktion, Darstellung und Ausführung von Algorithmen (*Software*).

Aufbau von Computern

- In der Architektur moderner Rechenanlagen (von-Neumann Modell) ist der Prozessor (CPU) für die „Berechnungen“ zuständig.
- Die CPU besitzt eine kleine Menge (ca. 50) an *Maschinenbefehlen* (plattformspezifisch).

„Von Neumann Rechner“



Quelle: Wikipedia

- Dies sind typ. logische und arithmetische Operationen.
- Die CPU kann diese Befehle sequentiell ausführen.
- Befehle und Daten sind *binär* repräsentiert.

Programmiersprachen

- Die binären Maschinenbefehle, die in der CPU verarbeitet werden können, sind das einzige, was eine CPU tun kann.
- Für den “normalen” Menschen sind diese Befehle praktisch unlesbar.
- Durch die Zuordnung von symbolischen Namen für Maschinenbefehle entstand eine erste abstrakte „Programmiersprache“.
- Diese symbolischen Namen heißen *Assembler-Befehle*, z.B. `ADD` für die Addition zweier Werte)
- Assembler-Sprachen waren die ersten Abstraktionen der Maschinensprachen in “Programmiersprachen”, in denen GAAAAANZ früher programmiert wurde.

Programmiersprachen

- Mit steigender Komplexität der Programme wurden die einfachen Assemblersprachen schnell unbrauchbar (es konnte zwar alles programmiert werden, aber leider sehr umständlich; die Programme wurden schnell sehr unübersichtlich).
- Mit der Zeit wurden daher noch höhere Abstraktionen dieser Maschinensprache entwickelt die schließlich in unterschiedlichen Programmierparadigmen mündeten.
- Aus diesen Paradigmen entstanden entsprechende "höhere" Programmiersprachen.
- Zur Umsetzung eines Programms einer höheren PGS in Maschinenbefehle benötigt man ein „Übersetzungsprogramm" (z.B. einen *Compiler*).

Die Programmiersprache Java

- Entwickelt von J. Gosling, Bill Joy, P. Naughton, u.a.
- Erste plattformunabhängige, objektorientierte Sprache, insbesondere zur Programmierung von Internet-Applikationen
- Sicherheitsaspekt in der Entwicklung der Sprache wichtig
- Erste Version 1.0 1995, heute Java 11
- ACHTUNG: Übungen sollten mit allen Java Versionen > 7 gelöst werden können. Auf den CIP-Rechnern ist Version 8 installiert.
- Ursprünglicher Name: OAK

Eigenschaften von Java

- Objektorientiert: Klassenkonzept, strenge Typisierung
- Pattformunabhängig: Übersetzung in Virtuelle Maschine (JVM)
- Netzwerkfähig, nebenläufig

Nachteil:

Laufzeithandicap durch Interpretation (JVM), wird aber stetig verbessert

Vorteile:

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

Ausführung von Programmen

Wie werden Programme auf Rechnern ausgeführt, d.h. wie wird der Sourcecode (der Programmtext) auf die Maschinenbefehle abgebildet?

Variante 1: Compiler-Sprachen

- Ein Compiler erzeugt vor dem Ausführen des Programms Plattform-abhängigen Maschinencode aus dem Sourcecode.
- Diese Übersetzung ist quasi die Abbildung auf die Menge der vorhandenen Maschinenbefehle (und diese ist natürlich abhängig von der Plattform)
- Der Maschinencode kann daher nur auf bestimmten Rechnerarchitekturen bzw. Betriebssystemen ausgeführt werden, ist dafür aber unabhängig vom Sourcecode.
- Beispiele: C/C++

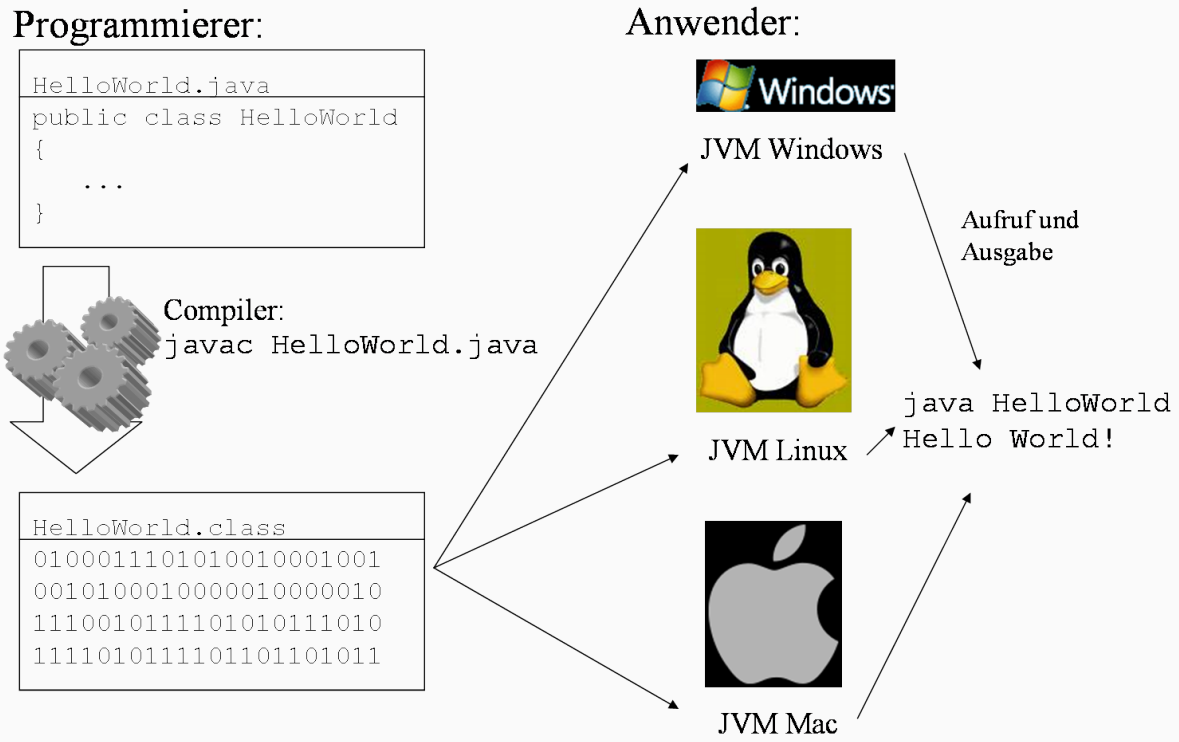
Variante 2: Interpreter-Sprachen

- Ein plattformspezifischen Interpreter interpretiert während dem Ausführen des Programms (zur Ausführungszeit) den Sourcecode (daher auch der Name: Skriptsprachen).
- Die Programme sind plattformunabhängig, aber der Sourcecode bleibt unübersetzt und sichtbar, was in vielen Anwendungen nicht erwünscht ist.
- Beispiele: Python, Perl, PHP, auch SML

Plattformunabhängigkeit

- Plattformunabhängigkeit eines Java-Programms wird durch einen Kompromiss erreicht:
  - Der Sourcecode wird übersetzt in Bytecode, der plattformunabhängig verwendet werden kann.
  - Bytecode wird von einer virtuellen Maschine ausgeführt (interpretiert).
  - Die virtuelle Maschine gibt es in verschiedenen Versionen für verschiedene Plattformen (JVM = Java Virtual Machine, Teil des JRE = Java Runtime Environment).
  - Der Bytcode sorgt dafür, dass das Programm nicht mehr (Menschen-)lesbar ist, die JVM übersetzt den Bytecode in die speziellem Maschinenbefehle der CPU.

Übersetzung und Ausführung



Wir unterscheiden dabei zwischen Übersetzungszeit und Laufzeit eines Programms.

Übersetzung und Ausführung

- Übersetzungszeit (Compilier-Zeit): der Zeitpunkt der Umwandlung des Quellcodes in Bytecode.
- Ausführungszeit: der Zeitpunkt der Ausführung des Bytecodes durch die JVM.
- Warum diese Unterscheidung?
  Antwort: es gibt wichtige Eigenschaften, die zu gewissen Zeitpunkten sichergestellt sein sollten — auch dazu später mehr.

Kommentare in Java

- Zu guter Letzt (zumindest in Bezug dieser Einleitung) noch ein gut gemeinter Tipp:
- Java unterstützt das Kommentieren von Source-Code in einer besonderen Qualität.
- Tun Sie sich und anderen einen Gefallen und nutzen Sie dieses Potential möglichst vollständig aus.
- Kommentare helfen Ihnen und anderen, Source-Code besser zu verstehen.

Kommentare in Java

Das wussten auch schon andere:

> *“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