2.

Java: Elementare Aspekte

Wir setzen gute Kenntnisse von C++ aus Teil I der Vorlesung voraus!

Für einige Aufgaben der Prüfungsklausur sollte man alle wesentlichen in der Vorlesung behandelten Java-Konzepte und -Konstrukte anwenden können

Buch Mark Weiss „Data Structures & Problem Solving Using Java“ siehe Seiten 3-68, Chapter 1 & 2 (primitive java, strings, arrays, input and output)

Lernziele Kapitel 2 Elementares Java

- Aufbau eines Java-Programms
- Elementare Datentypen, Arrays, Strings; strenge Typisierung
- E/A- und API-Nutzung bei Java
- Hauptunterschiede zu C++; Plattformneutralität durch VM

Thema / Inhalt

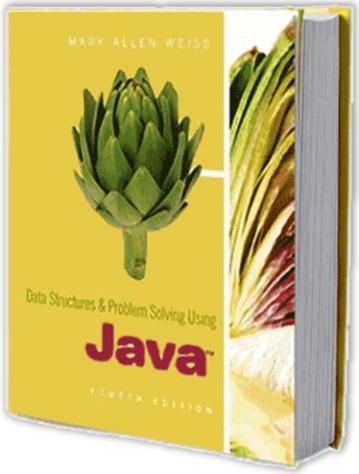
Die Programmiersprache **C++**, zumindest wesentliche Teile davon, kennen wir bereits aus „Informatik I". **Java** ist oberflächlich gesehen recht ähnlich, aber moderner, weniger fehleranfällig und bietet mehr Konzepte für neuere Anforderungen aus der Anwendungswelt, wie beispielsweise zur Parallelität. Beide Sprachen sind aber Mitglieder einer gemeinsamen Familie (zu der u.a. auch die Grossmutter C gehört), und um diese **Sprachfamilie** insgesamt geht es uns eigentlich.

Für den Anfang lernen wir in diesem Kapitel nur das Notwendigste, um einfache Java-Programme auf dem Niveau von C schreiben zu können: grundsätzlicher **Programmaufbau**, elementare **Datentypen** sowie zwei komplexere Datentypen, **Arrays** und **Strings**. Die strenge Typbindung („**Typisierung**") ist etwas gewöhnungsbedürftig, aber langfristig ein Segen. Mächtig wird Java vor allem durch die umfangreichen **Programmbibliotheken** („libraries"); wir lernen am Beispiel von Strings, wie man deren Funktionalität über die „**APIs**", also die angebotenen Schnittstellen zur Anwendungsprogrammierung, nutzt.

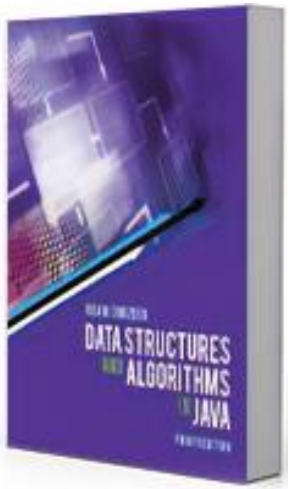
Java: Fokus Algorithmen

The only advice that one person can give another about reading is to take no advice, to use your own reason to come to your own conclusions. -- Virginia Woolf

- Buch von Mark Allen Weiss: *Data Structures & Problem Solving Using Java*, Addison Wesley, 4th edition, ISBN 0-321-54140-5
  - Zur Java-Sprache siehe insbes. Kapitel 1 – 4
  - Quellcode der Beispiele: http://users.cis.fiu.edu/~weiss/dsj4/code/
  - Schwerpunkt des Buches ist nicht Java selbst, sondern das Thema *Algorithmen und Datenstrukturen*



- Evtl. alternativ / als Ergänzung: Adam Drozdek: *Data Structures and Algorithms in Java*, Cengage Learning, 4th ed., ISBN 9814392782

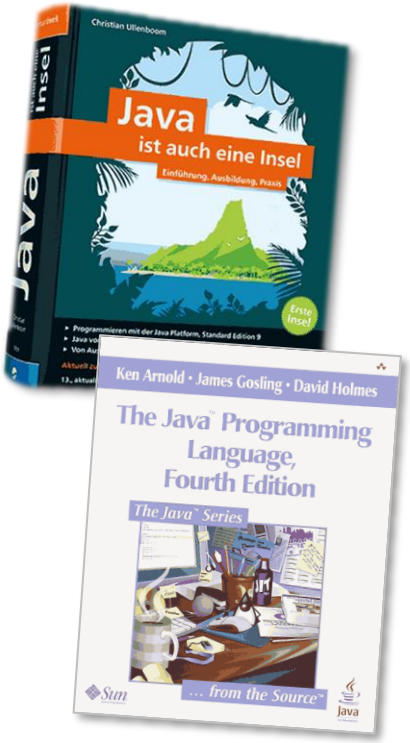


Java: Fokus Programmiersprache

Gewisse Bücher scheinen geschrieben zu sein, nicht damit man daraus lerne, sondern damit man wisse, dass der Verfasser etwas gewusst hat. – J.W. Goethe

- Zur Sprache Java gibt es sehr viele Bücher (diverser Qualität); empfehlenswert sind z.B.:
  - Christian Ullenboom: *Java ist auch eine Insel*, Rheinwerk-Verlag, 16. Auflage, 2021 (ca. 1300 Seiten; 12. Auflage frei im Web: http://openbook.rheinwerk-verlag.de/javainsel/)

Genügt uns!



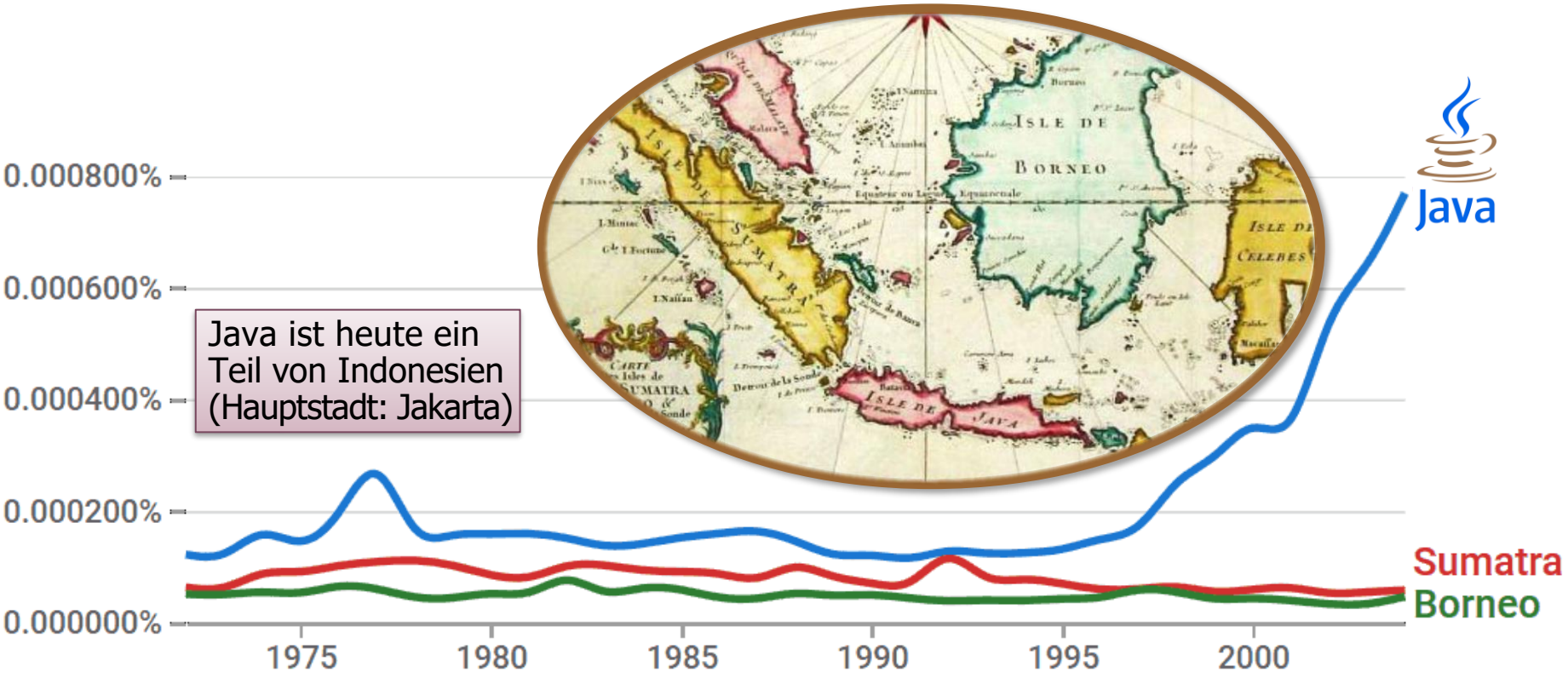
  - K. Arnold, J. Gosling, D. Holmes: *The Java Programming Language*, Addison-Wesley

Von den Java Entwicklern

- Interessant auch die themenspezifischen Java-Tutorials https://docs.oracle.com/javase/tutorial/
- Die Referenz für die Java-Klassenbibliotheken: *Java API Documentation / Specification* (und weitere Infos zu Java)
  https://docs.oracle.com/en/java/javase/ → API Documentation → java.base → java.lang

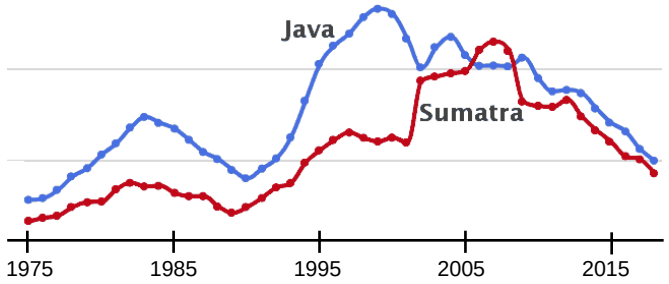
Java ist auch eine Insel…

The language was initially called Oak after an oak tree that stood outside Gosling's office. Later the project went by the name Green and was finally renamed Java, from Java coffee.



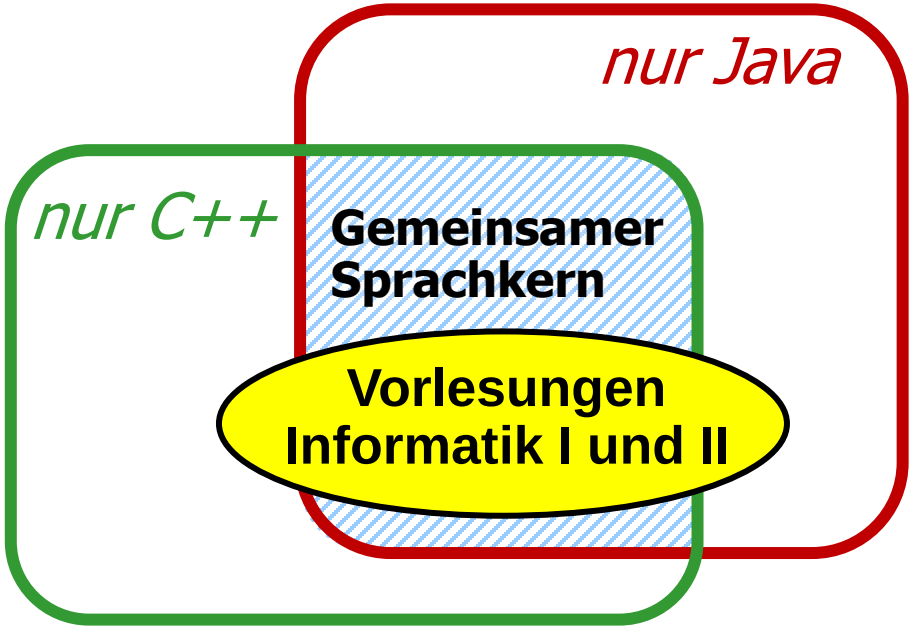
Oben: Relative Häufigkeit von Wörtern im deutschsprachigen Wortschatz von Büchern entsprechend Google.

Rechts: Im Vergleich dazu die Zeitungsdatenbank des DWDS. Man erkennt: In (Fach)büchern ist „Java" ab ca. 1995 stark vertreten; bei den Zeitungen ist hingegen weniger die Programmiersprache als die Insel gemeint.



Java und C++

- C++ und Java bilden eine gemeinsame Sprachfamilie
  - Einheitliche Syntax und analoge Semantik
  - C++: 1979, objektorientierte Erweiterung von C (ca. 1972, mit UNIX)
  - Java: Moderner, aber aufbauend auf C++ (sowie anderen Programmiersprachen wie Smalltalk); entwickelt ab 1991, öffentlich 1995



Beide Sprachen spielen in der Praxis eine grosse Rolle

Java: „Removed from C/C++“

- Zeigerarithmetik, malloc (aber: Arrays und new)
- Destruktoren, free, delete (aber: Garbage-Collector; finalize)
- Überladen von Operatoren

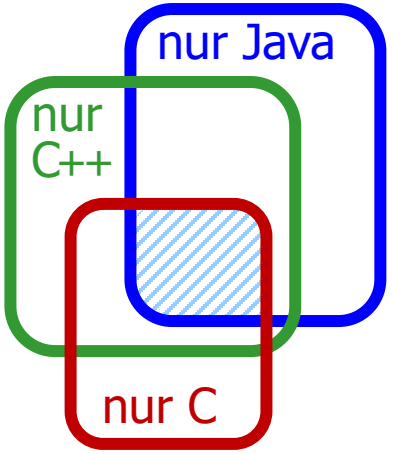
- Funktionen (stattdessen: Methoden)
- Implizite Typkonvertierung
- sizeof x (stattdessen: x.length)
- Strukturen, union (stattdessen: Klassen / Objekte)
- Templates
- Mehrfachvererbung (stattdessen: Interfaces)
- Friends (aber: „friendly access“ innerhalb von Paketen)
- Präprozessor: TYPEDEF, ...
- #DEFINE und const (stattdessen: final)
- goto (stattdessen: break/continue, Exceptions)
- using namespace, #include, .h-Dateien (aber: Pakete)

La perfection est atteinte non quand il ne reste rien à ajouter, mais quand il ne reste rien à enlever.
Antoine de Saint-Exupéry

Java: Neu gegenüber C++

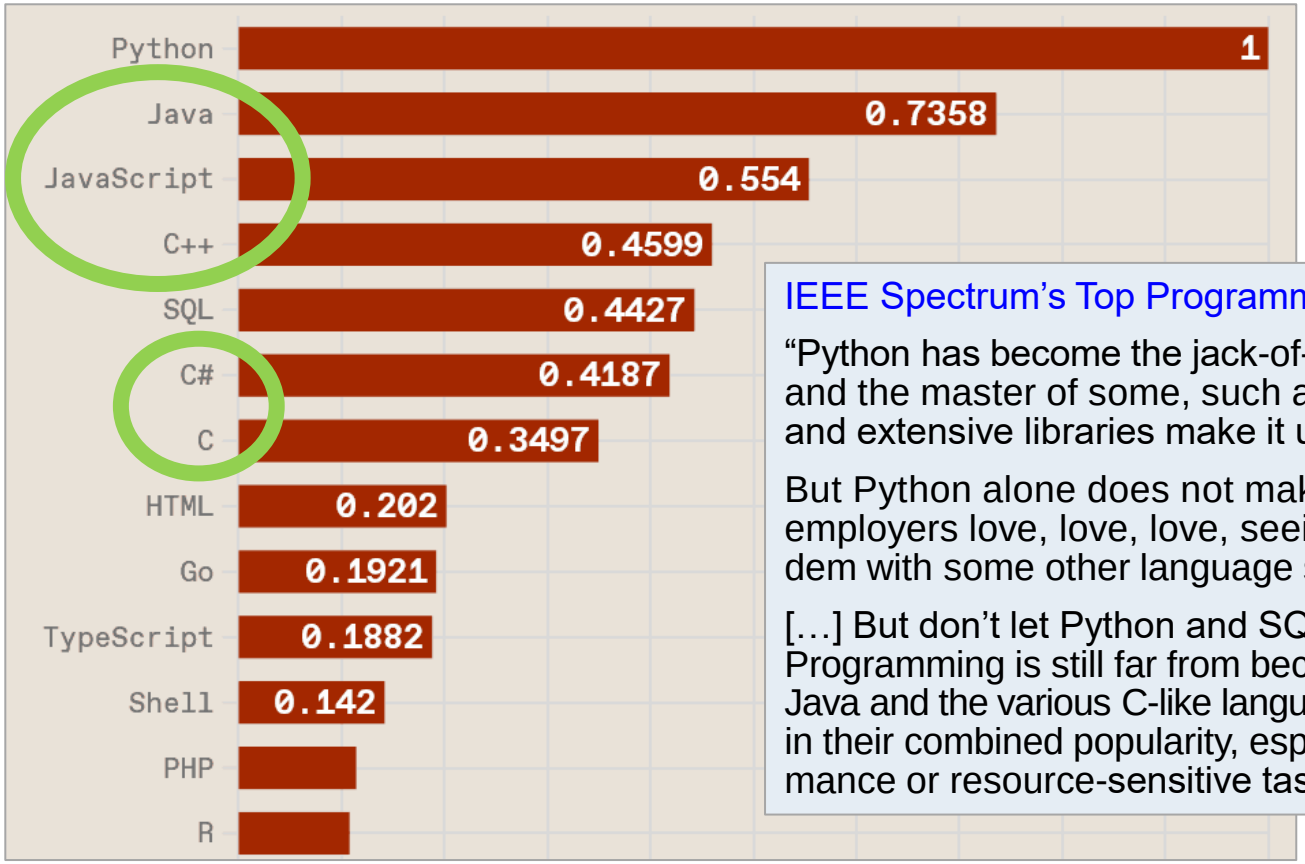
- Parallelverarbeitung in der Sprache selbst („Threads“)
- Viele vorgefertigte Pakete mit nützlichen Klassen
- …

- Generell: Java ist moderner, konsequenter, mehr „high-level“ und befreit von einigem historischen Ballast von C und C++
  - C++ hat allerdings in den letzten Jahren aufgeholt und auch einige gute Konzepte von Java „adoptiert“
  - C wird noch oft verwendet, wenn besonders effizient oder nah an der Hardware programmiert werden soll
  - C# („C sharp“) ist ebenfalls ein moderneres Element dieser Sprachfamilie, greift u.a. auch Konzepte der Sprachen Haskell und Delphi auf



The Top Programming Languages 2023

https://spectrum.ieee.org/the-top-programming-languages-2023



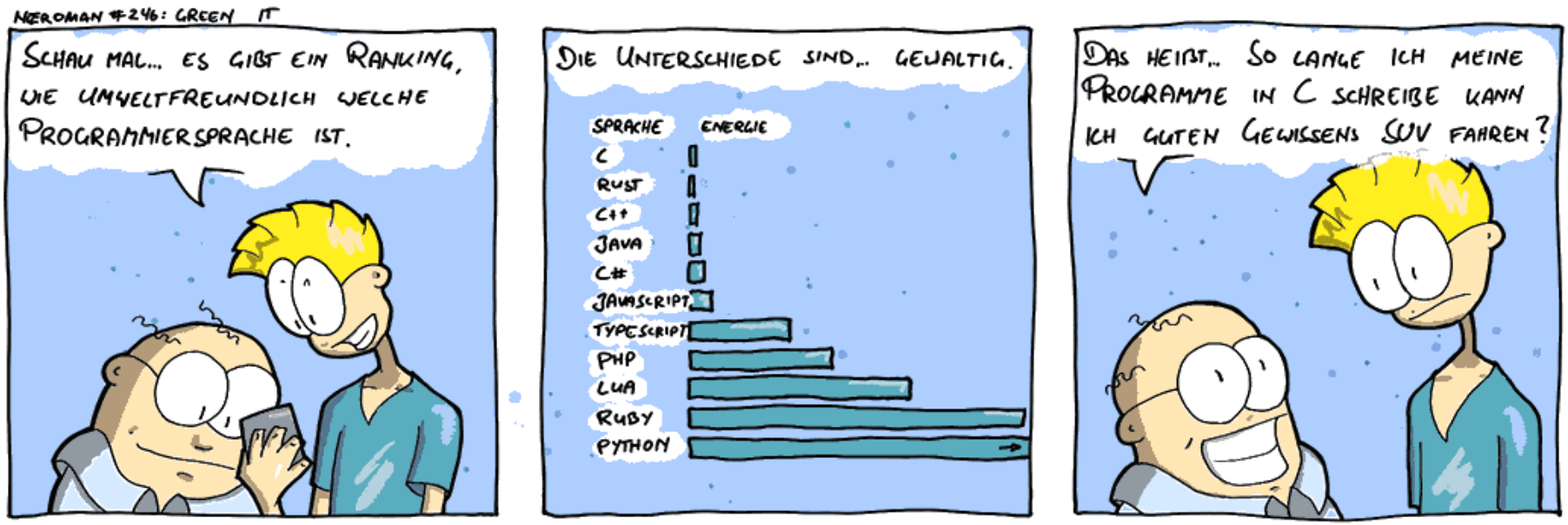
IEEE Spectrum's Top Programming Languages 2023.

"Python has become the jack-of-all-trades language—and the master of some, such as AI, where powerful and extensive libraries make it ubiquitous. […]

But Python alone does not make a career. Instead, employers love, love, love, seeing SQL skills in tandem with some other language such as Java or C++.

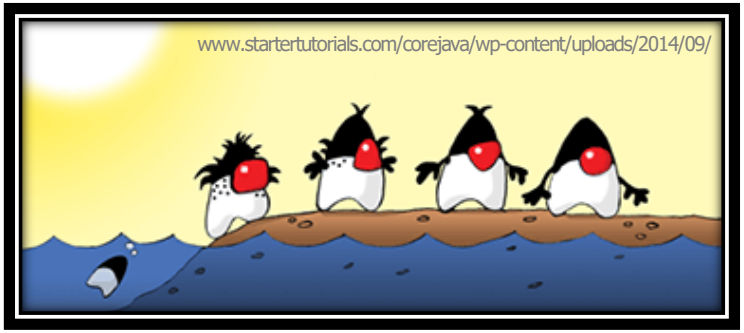
[…] But don't let Python and SQL's rankings fool you: Programming is still far from becoming a monoculture. Java and the various C-like languages outweigh Python in their combined popularity, especially for high-performance or resource-sensitive tasks."

The Top Programming Languages



NŒRDMAN Webcomic https://noerdman.de/

Java-Evolution und wichtige Sprachversionen



- 1996: Version 1.0
  (Sun Microsystems, u.a. James Gosling)
- 2000: Java 1.3
  - Häufig benutzte Codefragmente werden zur Laufzeit von Bytecode in Maschinencode übersetzt → Leistungssteigerung
- 2002: Java 1.4
  - U.a. assertions
- 2004: Java SE 5 (bzw. 1.5 oder „Java 2 Platform Standard Edition 5.0“)
  - Generische Typen und Aufzählungen (enum)
  - Implizite Umwandlung einfacher Datentypen in Objekte und zurück
- 2014: Java SE 8
  - Lambda-Ausdrücke als funktionale Sprachelemente; Default-Implementierung bei Interface-Methoden
- 2023: Java SE 20
  - Effizientere Implementierung von Threads („lightweight threads“ bzw. „virtual threads“)

Für unsere Zwecke sind die Unterschiede der div. Versionen weitgehend irrelevant!

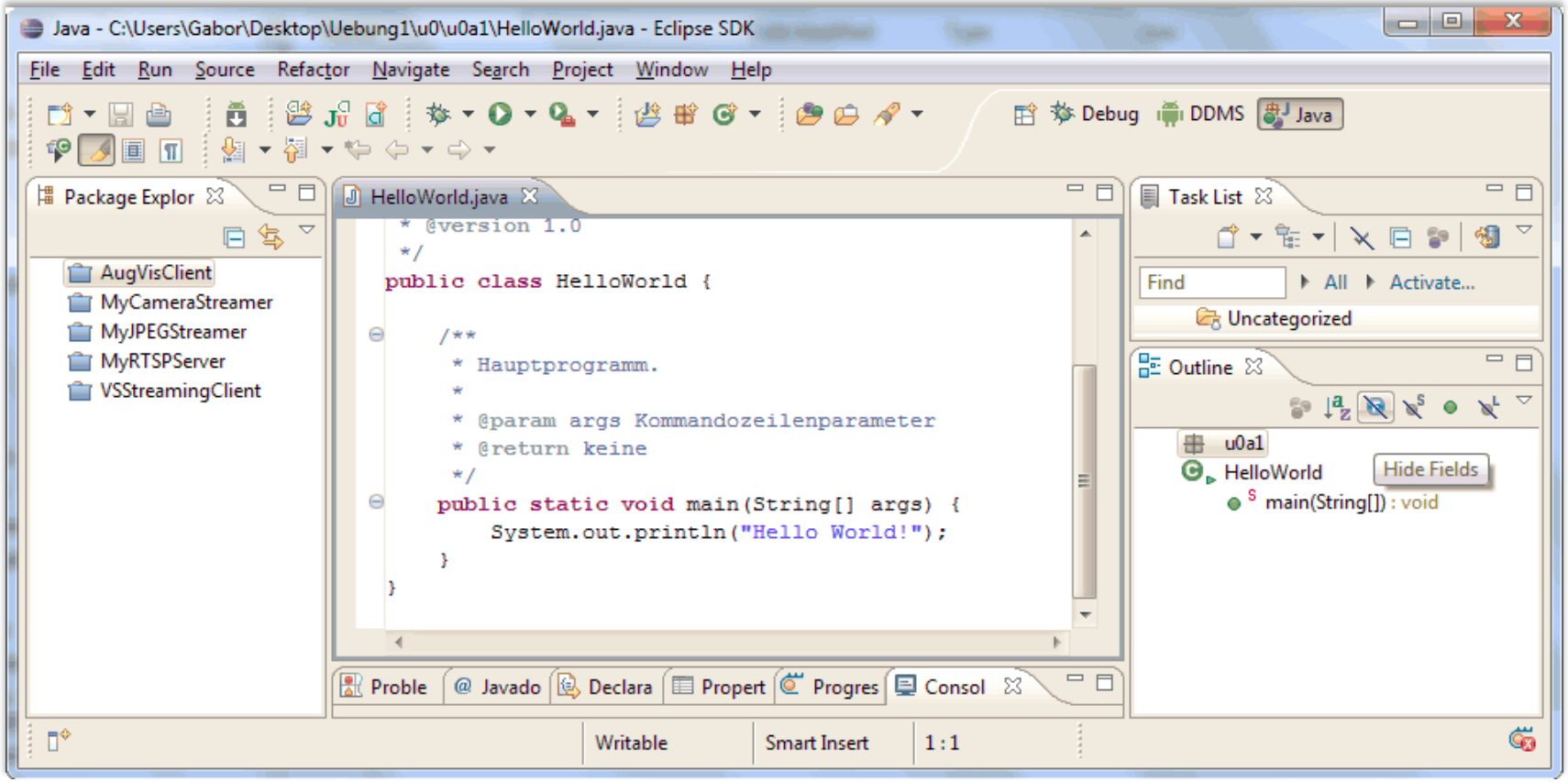
2010 wurde Sun Microsystems von Oracle übernommen

„Seit dem März 2018 veröffentlicht Oracle, man könnte sagen, pünktlich wie die SBB, alle 6 Monate eine neue Java-Version.“ -- www.inside-it.ch

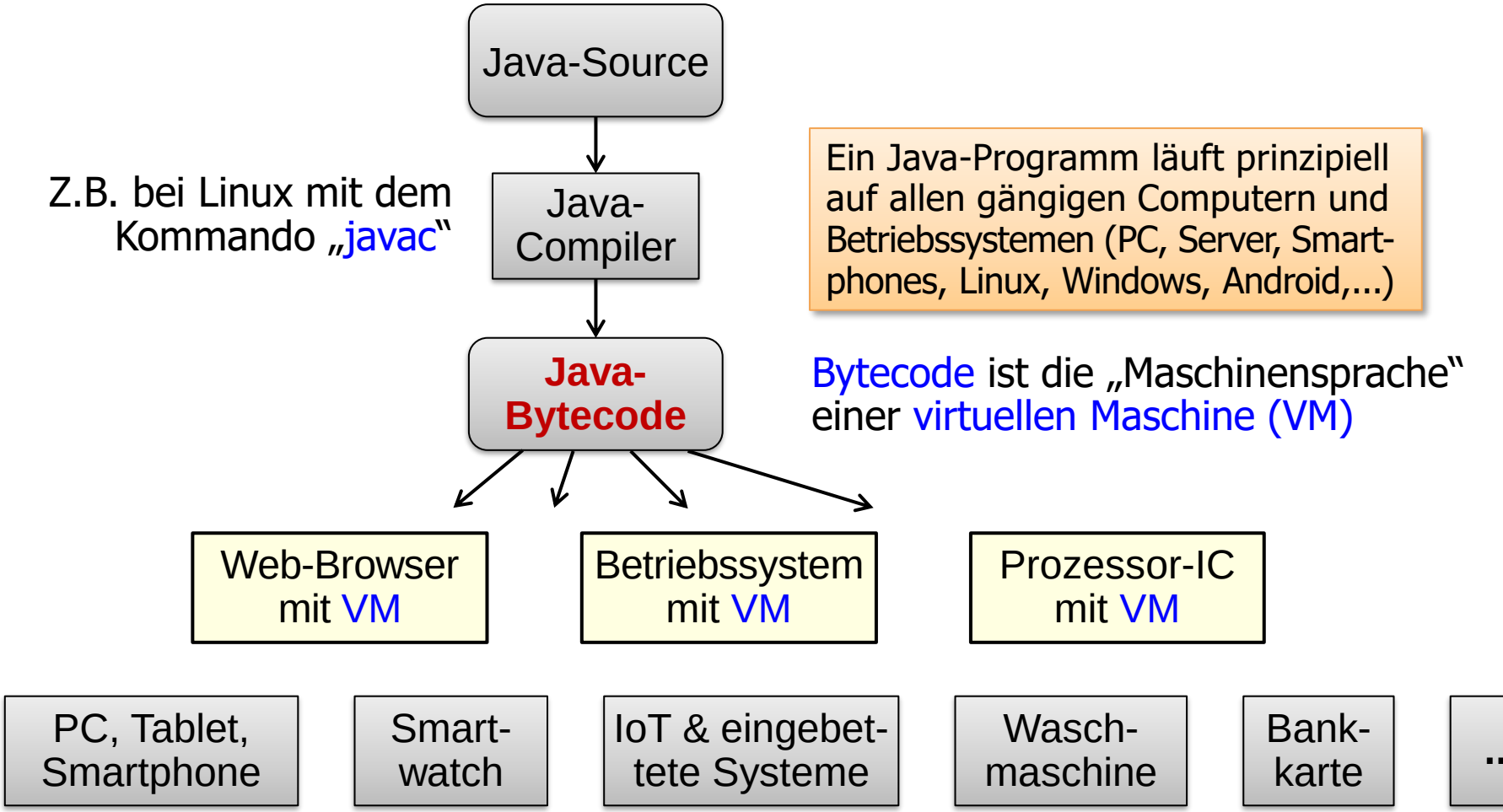
Programmieren mit Java

Integrated Development Environment

- Man benutzt meist eine Programmierumgebung („IDE“)
  - Zum Beispiel „Eclipse“ für Java: Diverse Programmierwerkzeuge integriert in eine graphische Oberfläche → Mehr in den Tutorien



Bytecode-Interpretation ermöglicht Plattformunabhängigkeit von Java



Z.B. bei Linux mit dem Kommando „javac“

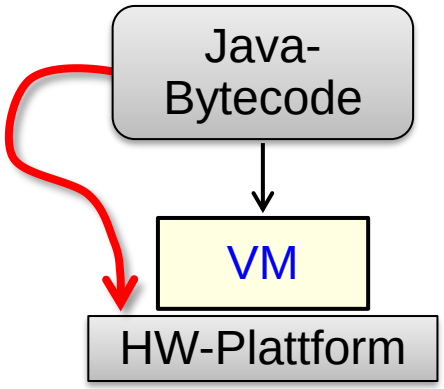
Ein Java-Programm läuft prinzipiell auf allen gängigen Computern und Betriebssystemen (PC, Server, Smart-phones, Linux, Windows, Android,...)

Bytecode ist die „Maschinensprache“ einer virtuellen Maschine (VM)

Die Virtuelle Maschine (VM)

- Die VM ist ein Bytecode-Interpreter
  - Programmierter Simulator eines abstrakten Prozessors
  - Relativ einfach für verschiedene Plattformen realisierbar
  - Unter Linux: Start der VM mit dem Kommando „java“

- Effizienzverlust durch Interpretation?
  - Evtl. stattdessen den Bytecode in Zielsprache (weiter-)übersetzen
  - Zumindest wiederholt gebrauchte Programmteile „just in time“



Java ganz kurzgefasst im „Cheat Sheet“

Java CHEAT SHEET

Basic code structure

text file named HelloWorld.java

name

main() method

```
public class HellWorld
{
   public static void main(String[] args)
   {
      System.out.print("Hello, World");
      System.out.println();
   }
}
```

statements

body

http://introcs.cs.princeton.edu/java/11cheatsheet/
http://visual.ly/java-cheat-sheet
www.lifehacker.com.au/2014/11/keep-this-java-cheat-sheet-on-hand-while-youre-learning-to-code/

Editing, compiling, and executing

use any editor to create your program

type javac HelloWorld.java *to compile your program*

type java HelloW... *to compile your program*

editor → HelloWorld.java → *compiler* → HelloWorld.class → *JVM* → "Hello, World"

your program (a text file)

computer-language version of your program

output

Date types

type	*set of values*	*common operators*	*sample literal values*
int	intergers	+ - * / %	99 -12 2147483647
double	floating-point numbers	+ - * /	3.14 -2.5 6.022e23
boolean	boolean values	&& || !	true false
char	characters		'A' '1' '%' '\n'
String	sequence of characters	+	"AB" "Hello" "2.5"

Assignment

declaration statement

variable name

literal

```
int a, b;
a = 1234
b = 99;
int c = a + b;
```

assignment statement

combined declaration and assignment statement

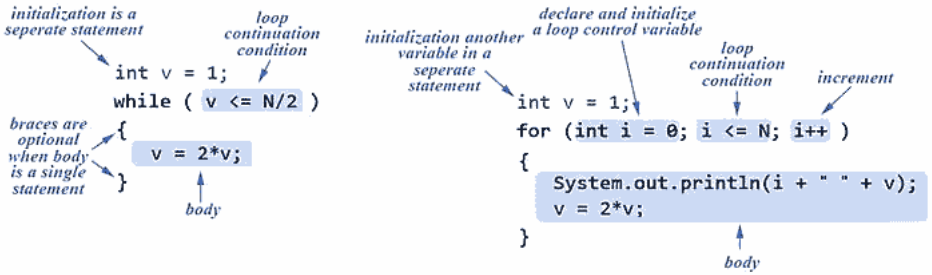
Booleans

values	true or false		
literals	true false		
operations	and	or	not
operators	&&	||	!

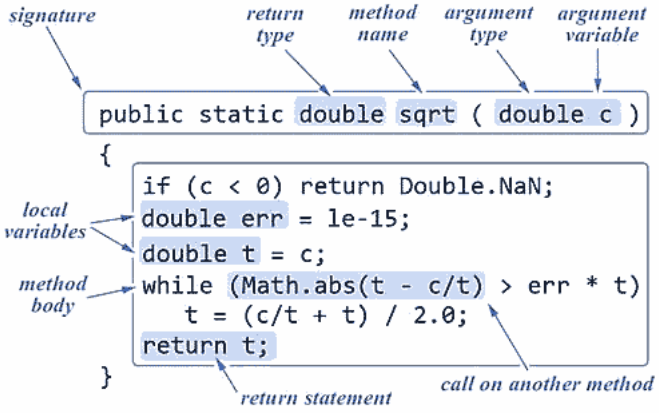
Comparison

op	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

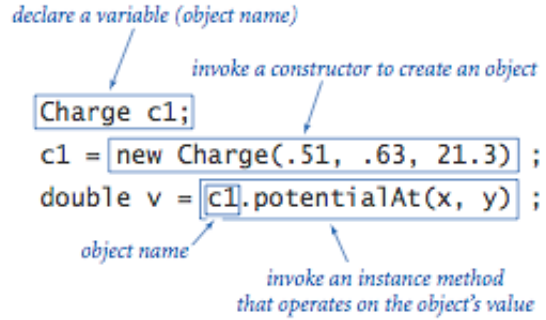
Loops



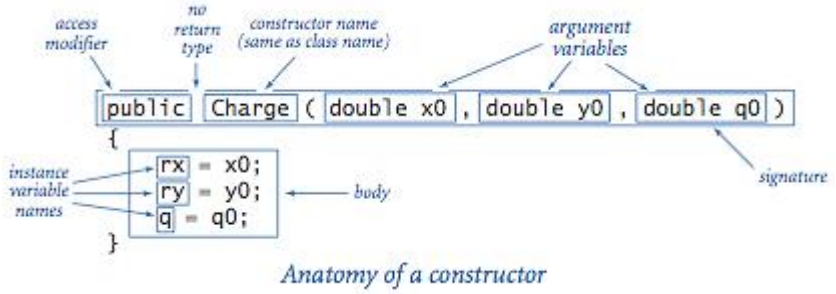
Methods



Using an object

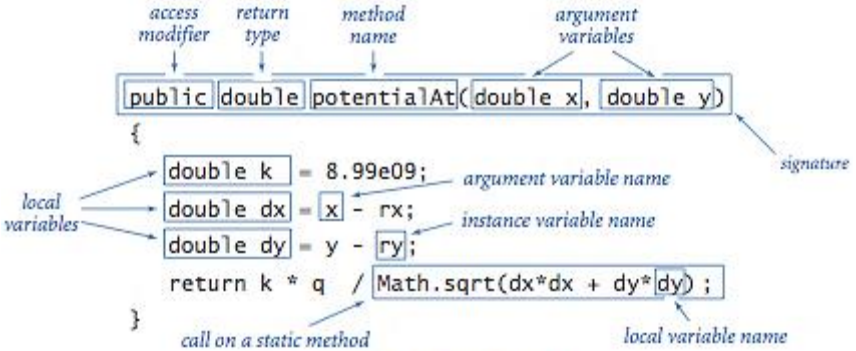


Constructors



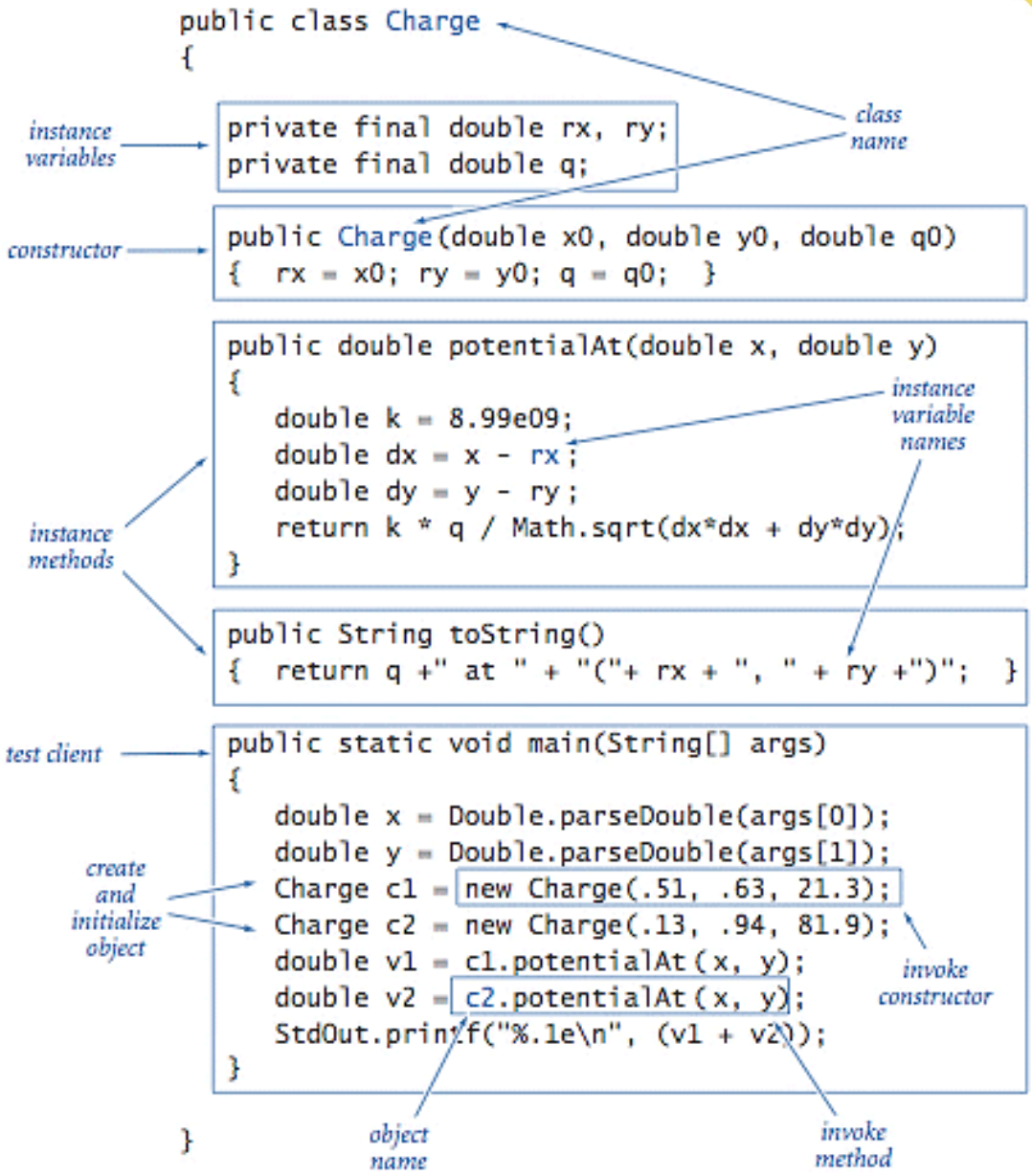
Anatomy of a constructor

Instance methods



Anatomy of an instance method

Classes

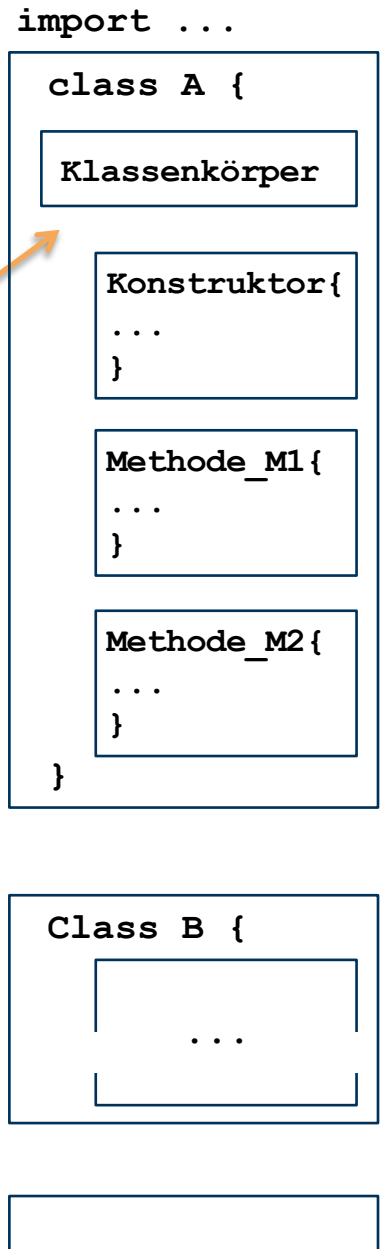

public class Charge
{
private final double rx, ry;
private final double q;
public Charge(double x0, double y0, double q0)
{ rx = x0; ry = y0; q = q0; }
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy);
}
public String toString()
{ return q +" at " + "("+ rx + ", " + ry +")"; }
public static void main(String[] args)
{
double x = Double.parseDouble(args[0]);
double y = Double.parseDouble(args[1]);
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
StdOut.printf("%.1e\n", (v1 + v2));
}
}
instance variables
class name
constructor
instance variable names
instance methods
test client
create and initialize object
invoke constructor
object name
invoke method

Java-Programmstruktur

- Mit import werden evtl. anderweitig vorhandene Pakete von Klassen verfügbar gemacht

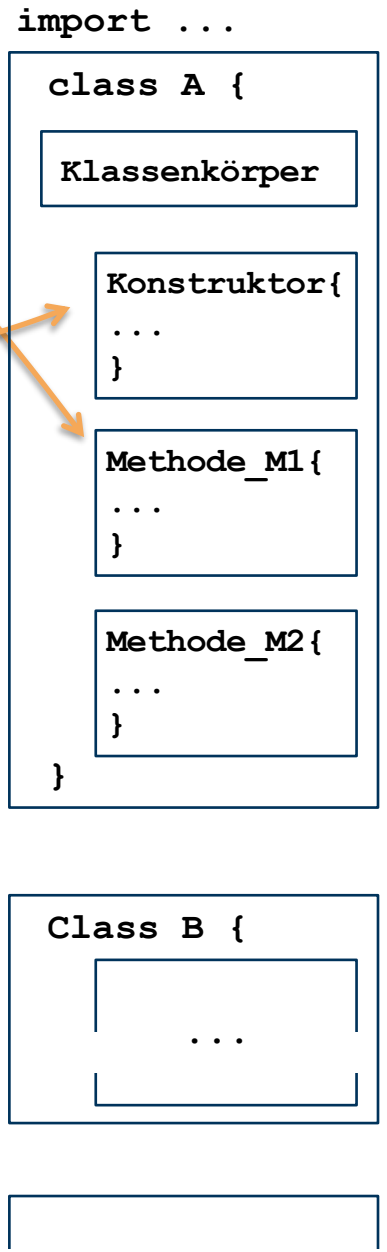
- Der Klassenkörper enthält
  - Instanzen- und Klassenvariablen („Attribute“)
  - Benannte Konstanten
  - Klassenbezogene („static“) Methoden

Manchmal auch „Felder“ genannt (Vorsicht: Gelegentlich wird als deutsche Bezeichnung für „Array“ auch der Begriff „Feld“ verwendet!)



Java-Programmstruktur (2)

- Methoden übernehmen die Rolle von Funktionen bzw. Prozeduren anderer Sprachen
  - Konstruktoren sind spezielle Methoden (bei Erzeugen der Klasse automatisch aufgerufen)
- Methoden haben einen Namen und bestehen aus
  - Parametern
  - Lokalen Variablen
  - Anweisungen
- Parameterübergabe erfolgt „by value“
  - Wertübergabesemantik bedeutet, dass Änderungen des „formalen“ Parameters in der Methode keine Änderung des „aktuellen“ Parameters beim Aufrufer bewirkt
  - Auch für Referenzen auf Objekte gilt die Wertübergabe! (Attribute des referenzierten Objekts können allerdings über den übergebenen Parameter verändert werden)

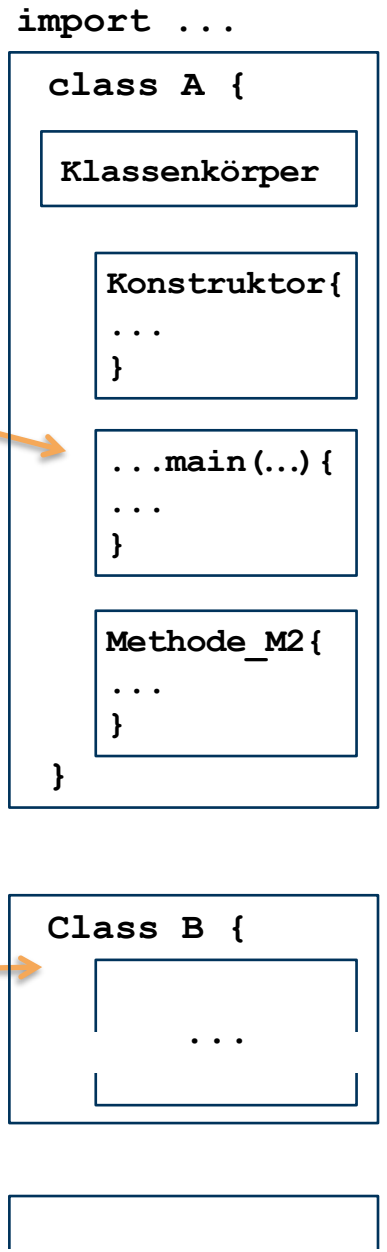


Java-Programmstruktur (3)

- Bei eigenständigen Programmen muss es eine „main"- Methode geben:

```
public static void main(String[] args) {
    ...
    ...
}
```

- Jede Klasse kann eine solche main-Methode enthalten; sie wird bei „Aufruf" der Klasse ausgeführt
  - Z.B. Linux: wenn der entsprechende Klassenname beim „java"-Kommando genannt wird
- Klassen können getrennt übersetzt werden
- Variablen im Klassenkörper ausserhalb von Methoden sind global zu allen Methoden der Klasse



Einfache Datentypen in Java

Bereits von C++ bekannt

- Integer (ganze Zahlen im 2er-Komplement):
  - **int** (32 Bits)
  - **long** (64 Bits)
  - **short** (16 Bits)
  - **byte** (8 Bits)
- Gleitkommazahlen
  - **float** (32 Bits)
  - **double** (64 Bits)
- Zeichen („Unicode“)
  - **char** (16 Bits, UCS-2)
- Wahrheitswerte
  - **boolean**

Bei C++ heisst das kürzer nur „bool“

Bildquelle: Christian Ullenboom: *Java ist auch eine Insel*, Galileo Computing, 8. Auflage, 2009, ISBN 3-8362-1371-4

Einfache Datentypen in Java

Bereits von C++ bekannt

- Integer (ganze Zahlen im 2er-Komplement):
  - **int** (32 Bits)
  - **long** (64 Bits)
  - **short** (16 Bits)
  - **byte** (8 Bits)

Bsp. für Werte: 17 , -3914
Bereich: -2147483648 ... 2147483647

Evtl. auch „Gleitpunkt", aber besser nicht „Fliesspunkt" oder „-komma"!

- Gleitkommazahlen
  - **float** (32 Bits)
  - **double** (64 Bits)

Bsp. für Werte: 18.0 , -0.18e2 , .341E-2

Integer-Zahl **1**, Gleitkommazahl **1.0**, Char-Zeichen **'1'** und String **"1"** werden im Speicher eines Computers vollkommen unterschiedlich repräsentiert!

- Zeichen („Unicode")
  - **char** (16 Bits, UCS-2)
- Wahrheitswerte
  - **boolean**

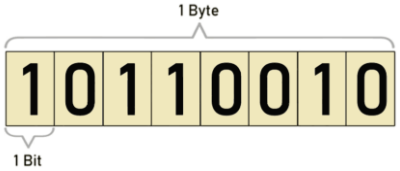
Werte: **true** , **false**
Operatoren: &&, ||, !

A priori sind diese Datentypen nicht kompatibel!

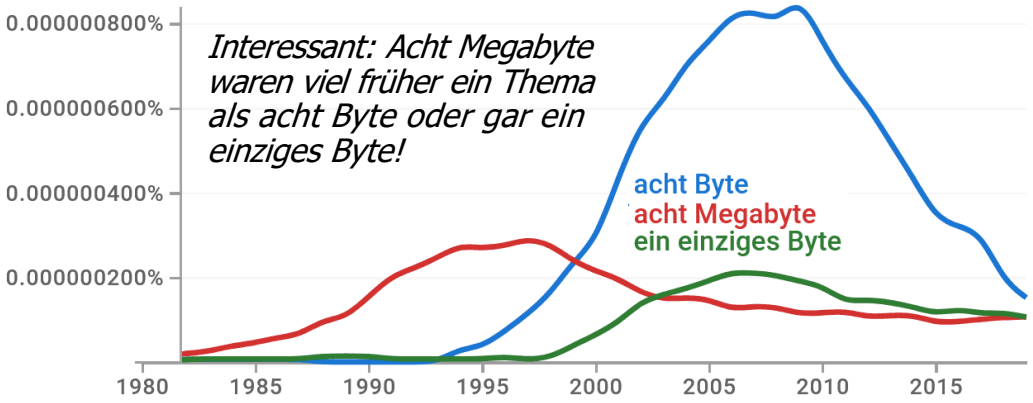
Bei C++ heisst das kürzer nur „bool"

- **int** (32 Bits)
- **long** (64 Bits)
- **short** (16 Bits)
- **byte** (8 Bits)

Das Kunstwort „Byte“ wurde erst 1956 kreiert – von Werner Buchholz (1922 – 2019), der damals als Hardwarearchitekt im Entwicklungslabor der Firma IBM in Poughkeepsie, USA arbeitete. Zunächst waren damit nicht exakt 8 Bits („Oktett“) gemeint, sondern eine Gruppe von Bits, die als kleinste adressierbare Einheit ein Zeichen codieren. Je nach Computerarchitektur konnten dies z.B. auch nur 6 Bits sein, beim Telex waren es 5 Bits. Das Wort „Bit“ (für binary digit, Binärziffer) wurde bereits 1946 von JohnTukey kreiert.



Buchholz wurde als Sohn einer jüdischen Kaufmannsfamilie in Deutschland geboren. Nach der Machtergreifung der Nationalsozialisten 1933 wollte die Familie ursprünglich nach Palästina auswandern, was jedoch nicht gelang. Im Alter von 16 schickten ihn seine Eltern aufgrund des zunehmenden Antisemitismus noch kurz vor Ausbruch des Zweiten Weltkriegs nach England. Dort wurde er nach Kriegsbeginn interniert und 1940 nach Kanada gebracht. 1941 wurde er aus der Internierung entlassen und konnte die University of Toronto besuchen, später machte er seinen PhD in Elektrotechnik am Caltech in Kalifornien. Seine Eltern wurden 1941 in das Ghetto Łódź deportiert. Der Vater kam dort 1942 um, die Mutter wurde 1944 weiter in das Vernichtungslager Kulmhof deportiert und dort ermordet.



Einfache Datentypen in Java (2)

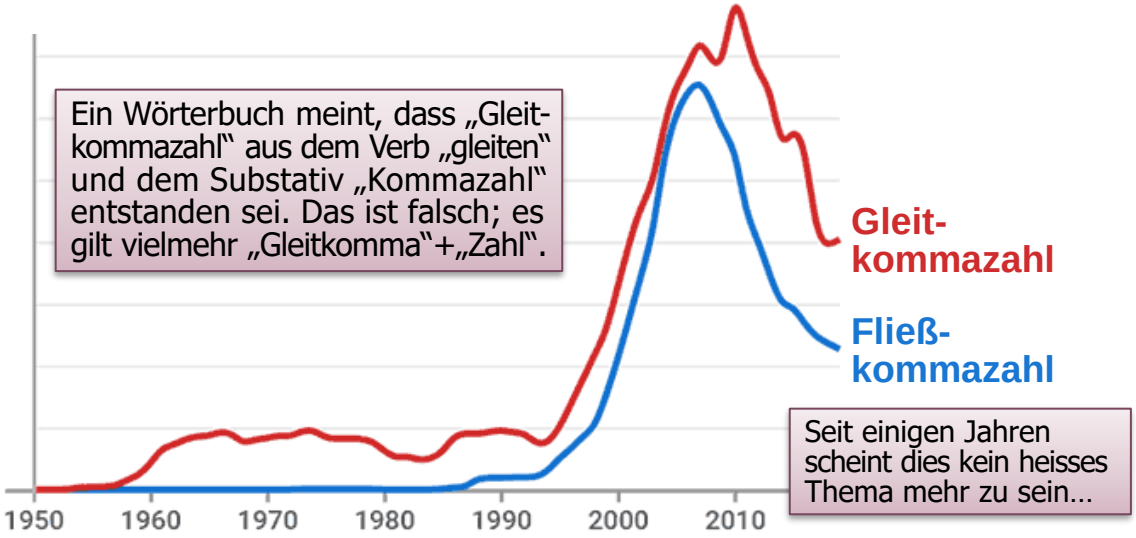
Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.
-- Leopold Kronecker

Datentyp	Grösse	Wrapper-Klasse	Wertebereich	Beschreibung
int	32 Bit	java.lang.Integer	−2 147 483 648 … +2 147 483 647	Zweierkomplement-Wert
long	64 Bit	java.lang.Long	−9 223 372 036 854 775 808 … +9 223 372 036 854 775 807	Zweierkomplement-Wert
short	16 Bit	java.lang.Short	−32 768 … +32 767	Zweierkomplement-Wert
byte	8 Bit	java.lang.Byte	−128 … +127	Zweierkomplement-Wert
float	32 Bit	java.lang.Float	±1.4E−45 … ±3.4E+38	Gleitkommazahl (IEEE 754)
double	64 Bit	java.lang.Double	±4.9E−324 … ±1.7E+308	Gleitkommazahl doppelter Genauigkeit (IEEE 754)
char	16 Bit	java.lang.Character	U+0000 … U+FFFF	Unicode-Zeichen (= Symbol) (z.B. 'A' oder '\uC3A4')
boolean	1 Bit	java.lang.Boolean	true / false	Boolescher Wahrheitswert

„Die mathematische Konstruktion der unendlichen Menge **Z** ist etwas radikal anderes als das, was in der Informatik der oft **int** genannte „Typ" ist – die endliche Menge jener ganzen Zahlen, die der Datentyp integer in grandioser Verknappung von **Z** enthält. Mathematik befasst sich mit Unendlichkeiten, Informatik hingegen mit Endlichkeiten. Vieles holt die Informatik sich aus der Mathematik, das meiste aber lässt sie dort." – Frieder Nake

Gleitkommazahlen

-- gleiten nicht und fliessen schon überhaupt gar nicht!



Im Englischen wurde der Begriff „floating-point number" mindestens seit 1947 benutzt und anfangs auch sinngemäss korrekt als „Gleitkommazahl" im Deutschen verwendet – es heisst ja nicht „flowing-point number" und das Dezimalkomma fliesst nicht, sondern schwebt, schwimmt, gleitet…

Die früheren (elektro)mechanischen Rechenmaschinen kannten kein Gleitkomma; man musste sich die richtige Position des Kommas bzw. den zugehörigen Exponenten einer Mantisse getrennt notieren (oder sich dazudenken und im Kopf nachführen) – eine häufige Fehlerquelle! Bei Computern war Zuse der erste, der Gleitkommazahlen und eine zugehörige Arithmetik (für seine Z1, Z3 und Z4) implementierte. Die ersten elektronischen US-Computer hatten dagegen noch kein Gleitkomma; der Hardware-Mehraufwand (auch an Bits für die Zahlendarstellung) schien Mitte der 1940er-Jahre den Nutzen nicht zu rechtfertigen.

https://dict.leo.org/:

to **float** | floated, floated |	schwimmen | schwamm, geschwommen |
to **float** | floated, floated |	treiben | trieb, getrieben |
to **float** | floated, floated |	flößen | flößte, geflößt |
to **float** | floated, floated |	schweben | schwebte, geschwebt |
to **float** | floated, floated |	gleiten | glitt, geglitten |
to **float** | floated, floated |	umlaufen | lief um, umgelaufen |
to **float** | floated, floated |	flottmachen | machte flott, flottgemacht |
to **float** | floated, floated |	im Umlauf sein
to **float** | floated, floated |	in Umlauf setzen
to **float** away	forttreiben | trieb fort, fortgetrieben |
to **float** off	abschwemmen | schwemmte ab, abgeschwemmt |
to **float** sth. | floated, floated | *[fig.] - an idea, option, etc.*	etw.[Akk.] in den Raum stellen *- eine Idee, Möglichkeit*
to **float** sth. | floated, floated | *[fig.] - an idea, option, etc.*	etw.[Akk.] zur Sprache bringen *- eine Idee, Möglichkeit*
to **float** | floated, floated | [COMP.]	relozieren | relozierte, reloziert |
floating bridge	die Schwimmbrücke *Pl.:* die Schwimmbrücken
floating bridge	die Schiffsbrücke *Pl.:* die Schiffsbrücken
floating capital	das Umlaufskapital
floating capital	das Umlaufvermögen *Pl.:* die Umlaufvermögen
floating file	die Umlaufmappe *Pl.:* die Umlaufmappen
floating population	nicht sesshafte Bevölkerung
floating head	fliegender Magnetkopf
floating mine	die Treibmine *Pl.:* die Treibminen
floating axle	die Schwebeachse *Pl.:* die Schwebeachsen
floating holiday	beweglicher Feiertag
floating money	verfügbares Geld
floating balance	der Schwebeanker *Pl.:* die Schwebeanker
floating decimal	das Gleitkomma *Pl.:* die Gleitkommas/die Gleitkommata
floating load	schwebende Last
floating stocks *Pl.*	variable Warenvorräte *Pl.*
floating tire[AE]	der Schwimmreifen *Pl.:* die Schwimmreifen
floating tyre[BE]	der Schwimmreifen *Pl.:* die Schwimmreifen
floating assets *Pl.* [FINAN.]	das Umlaufvermögen *Pl.:* die Umlaufvermögen
floating bearing [TECH.]	das Loslager *Pl.:* die Loslager
floating bearing [TECH.]	das Pendelkugellager *Pl.:* die Pendelkugellager
floating bearing [TECH.]	das Pendelrollenlager *Pl.:* die Pendelrollenlager
floating bearing [TECH.]	schwimmendes Lager
floating bearing [TECH.]	hydraulisch entlastetes Lager
floating bearing [TECH.]	das Gleitlager *Pl.:* die Gleitlager
floating bridge [BAU.][TECH.]	die Pontonbrücke *Pl.:* die Pontonbrücken
floating crane [TECH.]	der Schwimmkran *Pl.:* die Schwimmkräne
floating dock [NAUT.]	das Schwimmdock *Pl.:* die Schwimmdocks
floating ice [GEOL.]	das Treibeis *kein Pl.*
floating reamer [TECH.]	die Pendelreibahle *Pl.:* die Pendelreibahlen
floating voter *(Brit.)* [POL.]	der Wechselwähler *Pl.:* die Wechselwähler
floating floor [BAU.]	schwimmender Estrich

Gleitkommazahlen (2)

Die Argumentation gegen eine „eingebaute" Gleitzahlarithmetik ist aus heutiger Sicht interessant; wir zitieren dazu aus dem Bericht „Report on the mathematical and logical aspects of an electronic computing instrument" von 1946 / 47 der drei prominenten Autoren Arthur Burks, Herman Goldstine und John von Neumann. In dem Zusammenhang ist auch interessant, dass John von Neumann keinerlei Verständnis dafür hatte, dass Wissenschaftler selbst programmierten oder an die Verwendung höherer Programmiersprachen (also über Assembler hinaus) dachten.

"One basic question which must be decided before a computer is built is whether the machine is to have a so-called floating binary (or decimal) point. While a floating binary point is undoubtedly very convenient in coding problems, building it into the computer adds greatly to its complexity and hence a choice in this matter should receive very careful attention. […] Building a floating binary point into the computer will not only complicate the control but will also increase the length of a number and hence increase the size of the memory and the arithmetic unit. Every number is effectively increased in size, even though the floating binary point is not needed in many instances. Furthermore, there is considerable redundancy in a floating binary point type of notation, for each number carries with it a scale factor, while generally speaking a single scale factor will suffice for a possibly extensive set of numbers. By means of the operations already described in the report a floating binary point can be programmed. While additional memory capacity is needed for this, it is probably less than that required by a built-in floating binary point since a different scale factor does not need to be remembered foreach number. […]

Under suitable conditions it may be possible to avoid the mathematical analysis that is required to control the sizes […], by instructing the machine to rearrange the calculation, either continuously (i.e. for every number that is produced) or at suitable selected critical moments, so that all scales are appropriately readjusted and all numbers kept to the permissible size (i.e. between -1 and 1). The continuous readjusting is, of course, wasteful in time and memory capacity, while the critical (occasional) readjusting requires more or less mathematical insight. The latter is preferable, since it is almost always inadvisable to neglect a mathematical analysis of the problem under consideration.

The process of continuous readjustment can be automatized and built into a machine, as a *floating binary point*. We referred to it already in the first part of this report. […] The above discussion reemphasizes this point: The floating binary point provides continuous size readjusting, while we prefer critical readjusting, and that only to the extent to which it is really needed, i.e. inasmuch as the sizes are not foreseeable without undue effort. Besides the floating binary point represents an effort to render a thorough mathematical understanding of at least a part of the problem unnecessary, and we feel that this is a step in a doubtful direction."

Konventionen bei der Deklaration von Namen

- Variablen und Methoden beginnen mit einem Kleinbuchstaben
  - `int i, j, meinZaehler;`
  - `public aegypt_mult(…)`
- Klassennamen beginnen mit einem Grossbuchstaben
  - `class Person {…}`
- Benannte Konstanten ganz mit Grossbuchstaben
  - `MAX_SIZE`

Wir halten uns im Folgenden aber nicht immer an diese Konventionen…

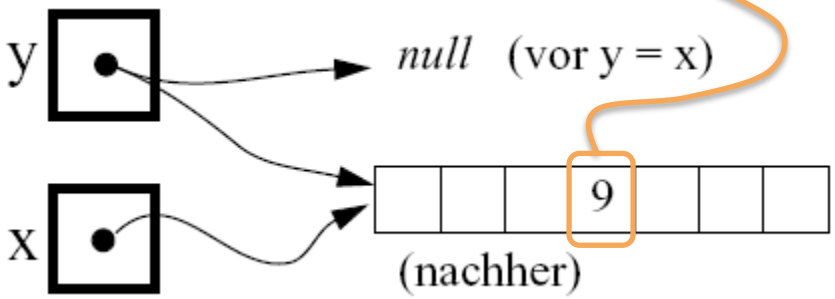
Arrays

Arrays werden auf Deutsch auch als „Felder“ oder „Reihungen“ bezeichnet

- Arrays sind („mathematisch betrachtet“) endliche Folgen

```
int [] x; // array of int
x = new int[7]; // (Indexbereich 0..6)
int [] x = new int[7]; // so ginge Obiges auch
for (int i=0; i < x.length; i++) x[i] = 1+2*i;
int [] y;
y = x;      // y zeigt auf das gleiche Objekt
y[3] = 9;   // x[3] ist daher jetzt auch 9
```

Länge (d.h. Anzahl der Elemente) = 7



Arrayvariablen enthalten Referenzen auf (Speicher)-Objekte: Vorsicht bzgl. der Kopiersemantik („Aliaseffekt“) und beim Vergleich zweier Arrayvariablen!

Arrays (2)

- Da Arrays mit „new“ dynamisch erzeugt werden, kann die Länge eines Arrays zur Laufzeit festgelegt werden:

```
int n;
...
n = … // Wert berechnen oder eingeben
int [] tabelle = new int [n];
```

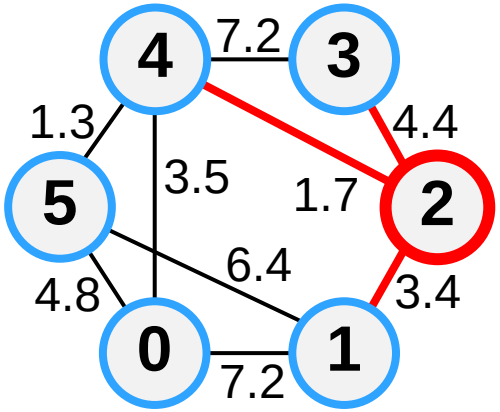
  - Einmal angelegt, kann sich die Länge aber nicht mehr ändern!
    Flexible Arrays (Datentyp „ArrayList“; entspr. „Vector“ in C++), besprechen wir später

- Mehrdimensionale Arrays:

```
float [][] matrix = new float [4][4];

matrix[0][3] = 2.71;
```

2D-Array-Beispiel: Adjazenzmatrix

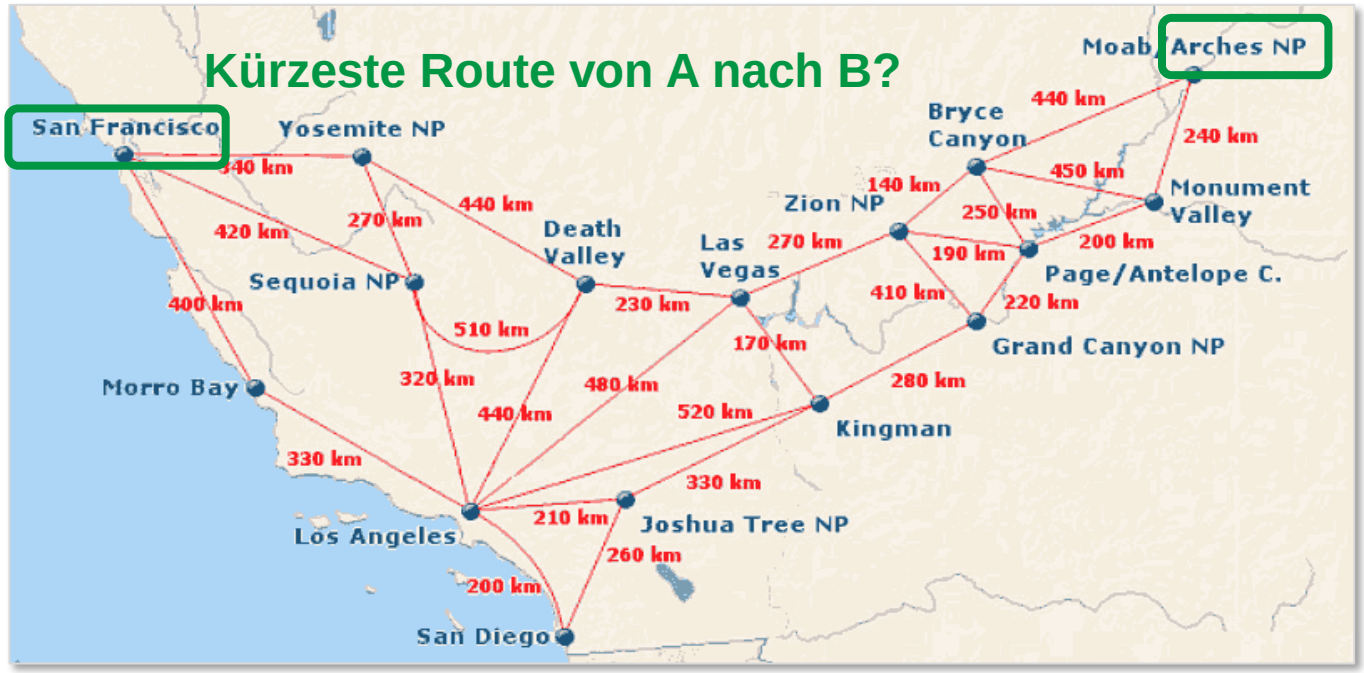


Entfernungstabelle

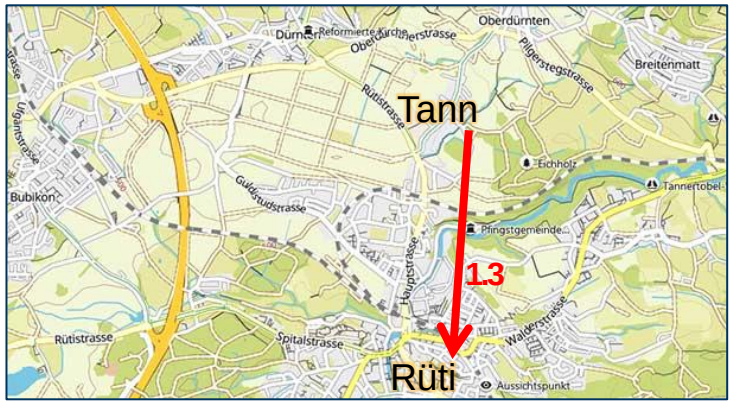
Von\nach		0	1	2	3	4	5
0	→	-	7.2	-	-	3.5	4.8
1	→	7.2	-	3.4	-	-	6.4
2	→	-	3.4	-	4.4	1.7	-
3	→	-	-	4.4	-	7.2	-
4	→	3.5	-	1.7	7.2	-	1.3
5	→	4.8	6.4	-	-	1.3	-

In diesem Beispiel symmetrisch; alle Werte > 0

Entfernung „unendlich“ beachten!



2D-Array-Beispiel: Adjazenzmatrix



Entfernungstabelle

Von\nach		0	1	2	3	4	5
0	→	-	7.2	-	-	3.5	4.8
1	→	7.2	-	3.4	-	-	6.4
2	→	-	3.4	-	4.4	1.7	-
3	→	-	-	4.4	-	7.2	-
4	→	3.5	-	1.7	7.2	-	1.3
5	→	4.8	6.4	-	-	1.3	-

In diesem Beispiel symmetrisch; alle Werte > 0

Entfernung „unendlich“ beachten!

Daten benachbarter Orte mit a-priori bekannten Distanzen erfassen:

```
float [][] entfernungstab; int ortszahl = 6;
entfernungstab = new float [ortszahl][ortszahl];

public void setzeEntf (int von, int nach, float km)
  {
    entfernungstab [von][nach] = km;
  }
public float kuerzesteRoute (int von, int nach)
  {
    // Genialer Algorithmus
    // (Routenplaner) hier!
  }
```

```
int TANN = 5; int RUETI = 4;
...;
setzeEntf (TANN,RUETI,1.3);
```

Grundidee des sogen. „Dijkstra-Algorithmus“ besprechen wir später

Statt der Entfernung könnte man auch typische oder aktuelle Reisezeiten nehmen!

Berechnet aus der Entfernungstabelle direkt benachbarter Orte die (kürzeste) Entfernung (und die Route!) der indirekt verbundenen Orte.

Typkonversion

Keine automatische Typ-konversion wie bei C++

- Java ist eine streng typisierte Sprache
  - → Bereits der Compiler kann viele Typfehler entdecken (was sonst zur Laufzeit zum Systemabsturz führen kann)
- Gelegentlich muss dies jedoch durchbrochen werden
- So geht es nicht (→ Fehlermeldung durch Compiler):

```
int myInt;
float myFloat = -3.14159;

myInt = myFloat;
```

"Type mismatch: cannot convert from float to int"

- Stattdessen explizite Typumwandlung („type cast"):

```
int myInt;
float myFloat = -3.14159;

myInt = (int)myFloat;
```

Typkonversion (2)

- Umwandlung hin zu einem grösseren Wertebereich (z.B. int → float) geht allerdings auch implizit

```
float d = 5 + 3.2;
```

- Typumwandlung ist gelegentlich sinnvoll bei Referenzen:

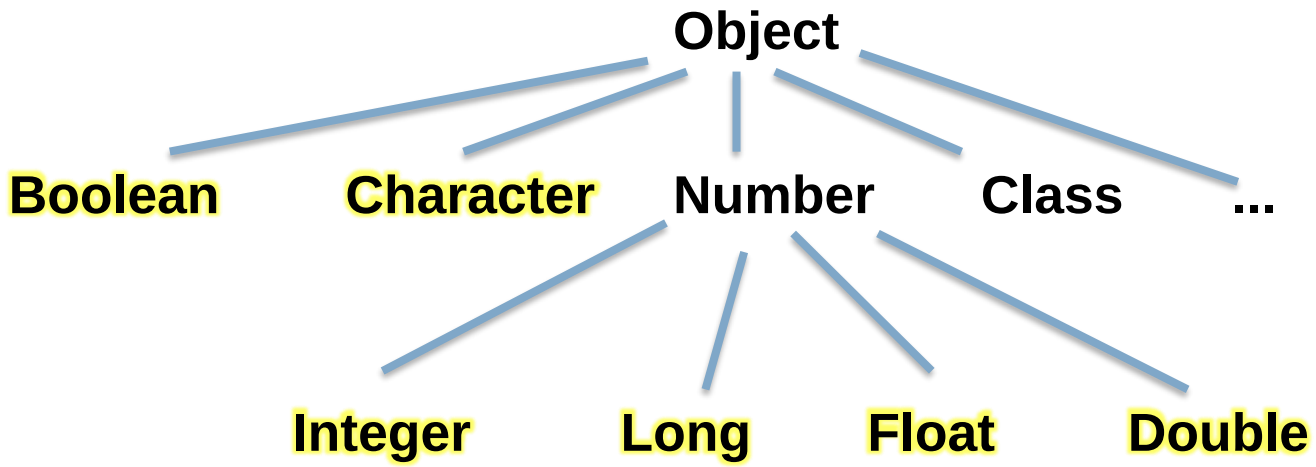
```
Hund h; Tier fiffi;
...
if (fiffi instanceof Hund)
   h = (Hund)fiffi;
```

Wenn das Tier mit dem Namen „fiffi" hier ein Hund ist, dann betrachte fiffi als einen Hund

Später mehr dazu (type cast bei Polymorphie)

Hüllenklassen

- Einfache Datentypen (int, float,...) sind a priori keine echten Objekte (zu grosser Aufwand → ineffizient!)
- Für diese gibt es bei Bedarf sogenannte Hüllenklassen



Hüllenklassen (2)

- Beispiel

```
int x = 5; // normaler "int"
Integer iob = new Integer (x); // Instanz der Klasse "Integer"
Integer iob = x; // Das gleiche in abgekürzter Form
if (iob == 5) then ... // sind typkompatibel
```

- Hüllenklassen bieten einige nützliche Methoden und Attribute
  - Vgl. dazu Sprachdokumentationen im Web oder geeignete Java-Bücher
  - Z.B. bei Integer:

```
floatValue ()
toString ()
```

  - Beispiele: `float f; … f = iob.floatValue();`

Ausgabe von Daten

```
System.out.println("Hallo, hallo!");
```

- System.out: Standard-Ausgabestrom
  - print gibt das übergebene Argument (auf einem Display) aus
  - println erzeugt zusätzlich danach noch einen Zeilenumbruch („newline“)
  - Es können int, float, string, boolean,... ausgegeben werden

```
int nummer = 007;
String bond = "007"
System.out.print("Hallo " + nummer);
System.out.println(" Hallo " + bond);
```

Konkatenation von strings

Denkübung: Bei `int nummer = 070;` wird 56 statt 70 ausgegeben. Wieso?

Eingabe von Daten (einzelne Zeichen)

```
int count = 0;
while (System.in.read() != -1) count++;
System.out.println("Die Eingabe hat " +
                   count + "Zeichen.");
```

System.in: Standard-Eingabestrom

- System ist eine Klasse mit Schnittstellenmethoden zum ausführenden System (Betriebssystem, Computer)
- System.in ist der Standard-Eingabestrom (vom Typ InputStream)
- read liest ein einzelnes Zeichen; liefert -1 bei Dateiende
- Es gibt noch einige weitere Methoden (skip, close,...)
- Erst abgeleitete Typen von InputStream enthalten Methoden, um ganze Zeilen etc. zu lesen (z.B. Klasse DataInputStream)

Eingabe von Zahlen – ein Beispiel

Dieses Paket enthält die Ein-Ausgabe-Methoden

Die auftretbaren Exceptions müssen nach **throws** am Anfang einer Methode genannt werden

Als Parameter beim Aufruf des Konstruktors den Eingabestrom angeben

```
import java.io.*;
class X {
  public static void main(String args[])
  throws java.io.IOException {
    int i=0; String zeile;
    DataInputStream herein = new DataInputStream(System.in);

while(true) {
      zeile = herein.readLine();
      i = i + Integer.parseInt(zeile);
      System.out.println(i);
    }
  ...
  }
  ...
}
```

Die Klasse DataInputStream enthält die Methode readLine, welche alle Zeichen bis Zeilenende liest und daraus einen string konstruiert

Beachte: Die Methode readLine kann eine IOException auslösen!

parseInt ist eine Methode der Klasse „Integer", die einen string in einen int-Wert konvertiert (analog kann man z.B. auch Gleitpunktzahlen einlesen)

Man könnte hier auch `herein.readLine()` für `zeile` substituieren

Eingabe von Zahlen – mittels BufferedReader

```
import java.io.*;
class X {
  public static void main(String args[])
  throws java.io.IOException {
    int i=0; String zeile;
    BufferedReader reader = new BufferedReader(new
                            InputStreamReader(System.in));
while(true) {
      zeile = reader.readLine();
      i = i + Integer.parseInt(zeile);
      System.out.println(i);
    }
  ...
  }
  ...
}
```

Zeichenketten (strings)

- Zeichenketten werden mit der Standardklasse String realisiert
  - Achtung: Strings sind im Unterschied zu C++ keine char-Arrays!

```
String msg = "Die"; // String-Objekt wird
int i = 7;          // automatisch erzeugt

msg = msg + " " + i;   // Konkatenation
msg = msg + " Zwerge";

System.out.println(msg); // Die 7 Zwerge
System.out.println(msg.length()); // 12

String b = msg;
msg = null;
System.out.println(b);
```

Alternativ zur ersten Zeile geht es auch so:

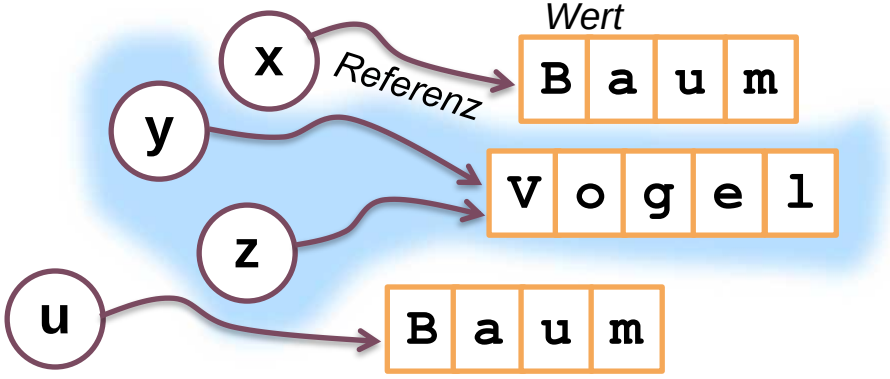
```
String msg
  = new String
    ("Die");
```

Jeder Referenzvariablen darf man den Wert der Nullreferenz zuweisen

Denkübung: Was wird hier ausgegeben?
a) Nichts (bzw. nur „newline“)
b) Fehlermeldung „NullPointerException“
c) „Die 7 Zwerge“

Strings: Vergleich

- Vergleich mit `==` (Referenzvergleich) ist meist nicht sinnvoll
  - Stringvariablen sind Referenzen auf Objekte!
- Stattdessen Wertevergleich: `s1.equals(s2)`



x und u referenzieren nicht das gleiche Objekt

- x == u → „false"
- x.equals(u) → „true"
- u.equals(x) → „true"
- y==z → „true"
- y.equals(z) → „true"

- Lexikographischer Vergleich mit `s1.compareTo(s2)`
  (liefert einen int-Wert < 0, = 0, oder > 0)

„Lexikographisch": Wie im Lexikon – also alphabetisch; bei gleichen Präfixen kommt es auf das erste unterschiedliche Zeichen an („Zucker" < „Zug"); ganze Wörter kommen vor Präfixen anderer Wörter („Kuh" < „Kuhle"); wo „ü" angeordnet wird (bei / vor / nach „u", oder erst nach „z"), wie Gross-/Kleinbuchstaben, Ziffern, Sonderzeichen angeordnet werden, ist durch das Alphabet bestimmt (hier: Unicode).

Strings: nützliche Methoden

- Es gibt eine Vielzahl von Methoden und Konstruktoren
  - Länge („length“)
  - Teilstrings („substring“)
  - Umwandlung von Zeichen (z.B. Gross- / Kleinschreibung)
  - Umwandlung von char- und byte-Arrays in strings
  - Umwandlung von anderen Datentypen in strings (und umgekehrt)
  - …
- Mehr dazu in der „Java Platform API Specification“:
  https://docs.oracle.com/en/java/javase/ → API Documentation → java.base → java.lang

Auszug aus der API-Beschreibung für String
(API = „Application Programming Interface")

compareTo

public int compareTo(String anotherString)
Compares two strings lexicographically.

Parameter:
anotherString - the String to be compared.

Returns:
The value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

Die einzelnen Zeichen werden entsprechend ihrer Reihenfolge im Unicode-Zeichensatz verglichen. Möchte man Gross- / Kleinbuchstaben gleich behandeln, dann verwende man stattdessen compareToIgnoreCase.

Beispiel:

```
if ((q[i].name).compareTo(q[i+1].name) < 0)
{  System.out.println("OK");
  else
    System.out.println("nicht sortiert!");
}
```

Auszug aus der API-Beschreibung für String (2)

concat

public String concat(String str)
Concatenates the string argument to the end of this string.

Parameter:
str - the String which is concatenated to the end of this String

Returns:
A string that represents the concatenation of this object's characters followed by the string argument's characters.

Beispiele:

```
String s = "Zahlen Sie $ 3";
s.concat(" Millionen"); //Hack
"Zeit".concat("geist");
```

Auszug aus der API-Beschreibung für String (3)

copyValueOf

public static String copyValueOf(**char** [] data)

Parameter:

data - the character array

Returns:

A String that contains the characters of the array.

Beispiel:

```
char[] c = {'h','e','l','l','o',',',' ','w','o','r','l','d'};
String s;
s = String.copyValueOf(c);
System.out.println("Der String s lautet: " + s);
```

Auszug aus dem API-Index für String

int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(StringBuffer sb) Compares this string to the specified StringBuffer.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(String anotherString) Compares this String to another String, ignoring case considerations

Es gibt noch viel mehr!

„Hard Skills wie Programmieren…

…sind gar nicht so entscheidend. Durch die Tools und Kurse, die es heute gibt, kann man in zwei Monaten lernen, wie man ein selbstfahrendes Auto programmiert – in Zukunft werden sich die meisten Programme ohnehin von selbst schreiben."

So Gerhard de Haan (Jahrg. 1951), Erziehungswissenschaftler und Professor für Zukunfts- und Bildungsforschung an der FU Berlin, am 7.9.2021 in Spiegel Online. Das Leserecho darauf war enorm. Aus den hunderten von Kommentaren hier einige nette und typische Ausschnitte:

Selten so gelacht. In dem Auto möchte ich nicht sitzen! --- Was meint er damit? Das Ziel ins Navi eingeben? Dafür wären 2 Monate etwas lang. --- Autonomes Fahren, das neue „Hello World" in der Programmierpädagogik. --- Und aus diesem Grund sind die Straßen voller selbstfahrender Autos. Weil man es ohne Ingenieursstudium in 2 Monaten programmieren kann. So als Anfänger quasi. --- Er hat nicht von „unfallfrei" gesprochen... --- Dann kann man sich ja gar nicht mehr auf die Straße wagen! --- Ob sich diese „Experten" auch von Menschen (am Kopf?) operieren lassen, die sich ihr Wissen über die Anatomie des Menschen und gängige Operationsmethoden nebenbei, in zwei Monaten, angeeignet haben? --- Sie können auch nicht Geige spielen, wenn sie Noten lesen können. --- Ich empfehle „Teach Yourself Programming in Ten Years" von Peter Norvig. Frei und kostenlos im Internet. --- Die Computer werden halt immer besser, und wenn wir sie mit Kernfusionsstrom betreiben auch leistungsstark genug, um sich selbst zu programmieren! --- Der Mensch, der sowas schreibt, kann wohl einen Toaster nicht von einem Computer unterscheiden. --- Wissen die Leute denn nicht, dass sie keine Ahnung haben oder denken sie, alle anderen sind nur zu doof? --- Dieser Spiegelbeitrag versüßt mir den ohnehin schon schönen Tag. Ich rege mich ganz gerne mal auf, spotte mit Herz und Leidenschaft – und ist es nicht stets ein gutes Gefühl, wenn einer blöder ist als man selbst? --- Der Professor zeigt sehr eindeutig, woran es in Deutschland mangelt: An Menschen, die tatsächlich etwas von Digitalisierung und Softwareentwicklung verstehen. --- Und ich frage mich unwillkürlich, ob seine Kollegen vom Institut für Informatik wissen, dass der gute Mann sie in den Medien für obsolet erklärt. --- Immerhin kann man mit dieser Wissensbasis Professor werden... --- Gilt das an der FU Berlin als Wissenschaft oder experimentelles Theater? --- Wenn jemand, der 8 Jahre für seinen Master gebraucht hat, einen Soziologie-Professor zum Thema Karriere in der Informatik interviewt, ist das ähnlich erkenntnisreich, wie wenn ein Blinder über Farben referiert. --- Man fragt sich, wieso selbst ein mittelmäßiger Journalist nicht auf die Idee kommt, einen Prof. bei solchen Aussagen vielleicht mal zu fragen, ob er sich – angesichts völliger Abwesenheit inhaltlicher Kompetenz – wirklich sicher sei, das ein derartiger Satz veröffentlicht werden soll. --- Zukünftig schreiben sich die Artikel selbst – dann gibt es vlt. wieder vernünftigen Journalismus.

Resümee des Kapitels

- Java
  - Virtuelle Maschine (VM), Bytecode
  - Globale Programmstruktur

- Basics der Java-Sprache
  - Einfache Datentypen
  - Typkonversion
  - Hüllenklassen
  - Arrays
  - Ein- und Ausgabe
  - Strings