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

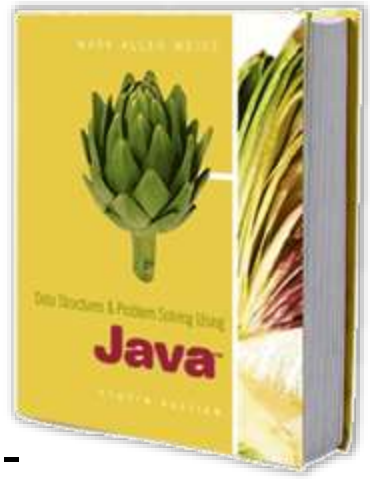
Thema / Inhalt

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.

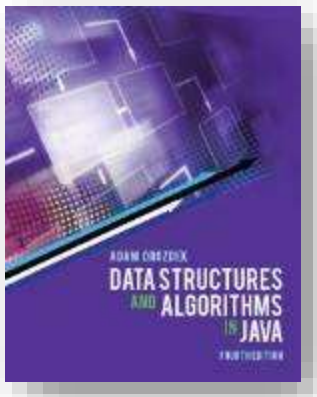
Für den Anfang lernen wir in diesem Kapitel nur das Notwendigste, um einfache Java-Programme auf dem Niveau von C schreiben zu können: Grundsätzlicher **Programmaufbau**, elementare **Datentypen** sowie zwei komplexere Datentypen, **Arrays** und **Strings**. Die strenge Typbindung („**Typisierung**") 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 Datenstrukturen und Algorithmen

- Buch von Mark Allen Weiss: *Data Structures & Problem Solving Using Java*, Addison Wesley, 4th edition, 2010, 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 liegt nicht auf Java selbst, sondern auf dem Thema *Algorithmen und Datenstrukturen*



- Evtl. alternativ / als Ergänzung: Adam Drozdek: *Data Structures and Algorithms in Java*, Cengage Learning, 4th edition, 2013, 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

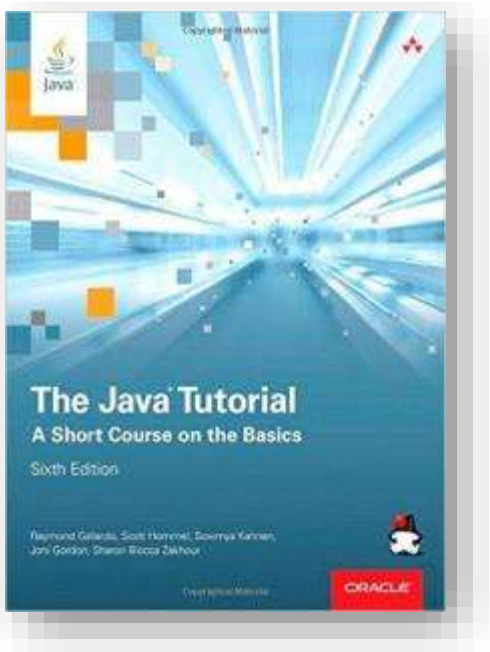
- Zur Sprache Java gibt es ausserordentlich viele Bücher (diverser Qualität), empfehlenswert sind z.B.:
  - Christian Ullenboom: *Java ist auch eine Insel*, Rheinwerk-Verlag, 14. Auflage, 2018, ISBN 978-3-8362-6721-2, (ca. 1300 Seiten; auch als e-book; 10. Auflage frei im Web)

    Genügt uns!
  - Ken Arnold, J. Gosling, D. Holmes: *The Java Programming Language*, Addison-Wesley, 4th ed., 2005 (auch als Online-Tutorial im Web)

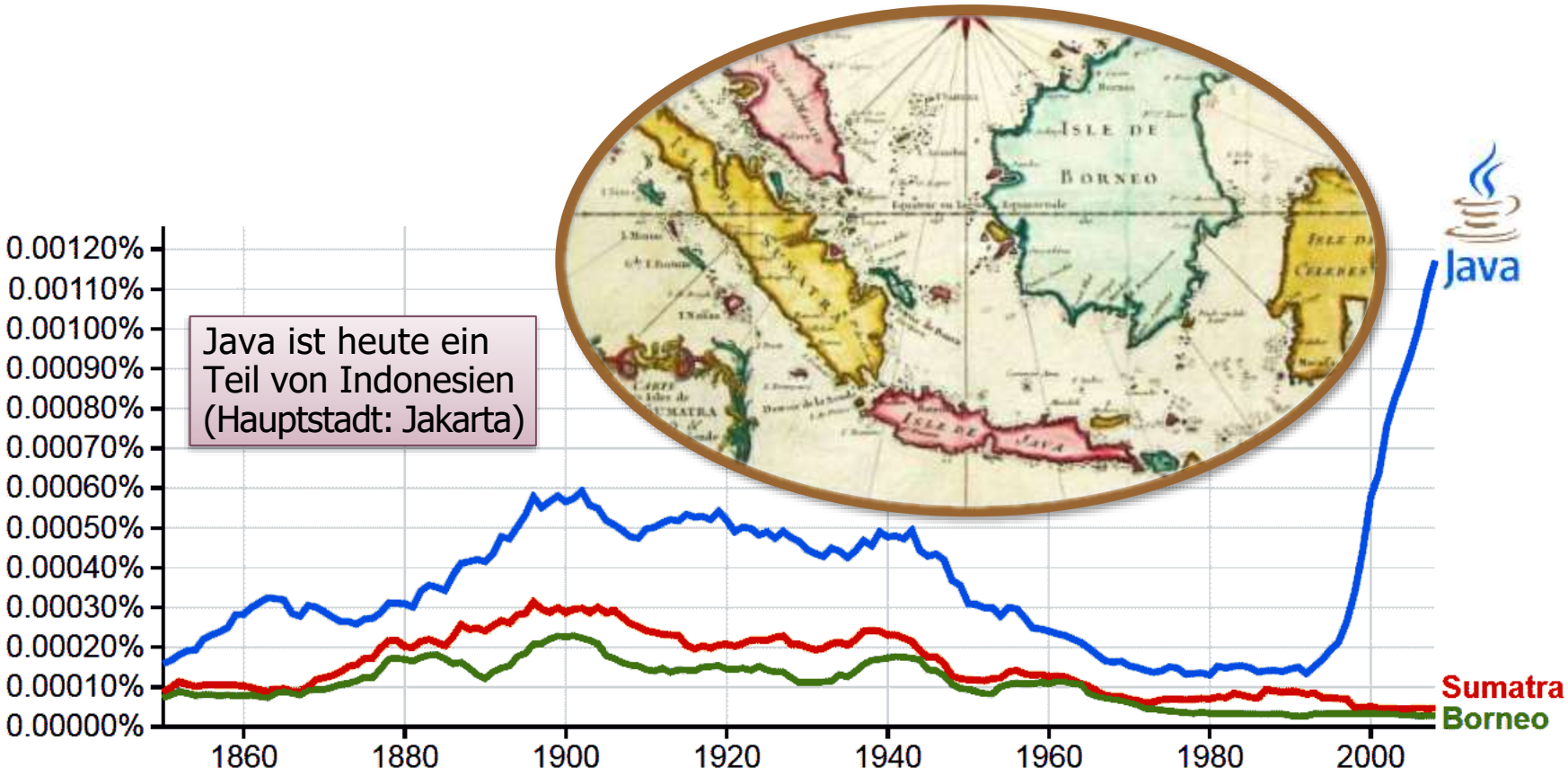
- Die Referenz für die Java-Klassenbibliotheken: *Java API Specifications* https://docs.oracle.com/en/java/javase/13/docs/api/index.html

Java: Tutorials

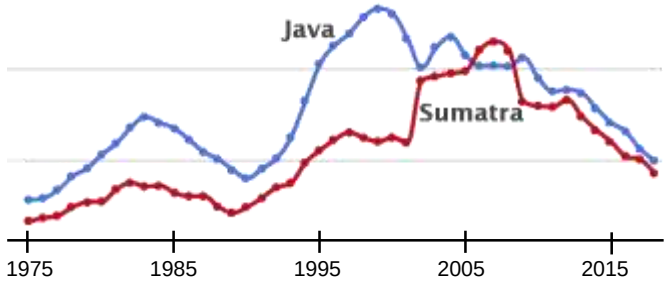
- Man konsultiere evtl. auch die Java-Online-Tutorials zu unterschiedlichen Teilthemen
  - https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html und https://docs.oracle.com/javase/tutorial/
  - Relevante Teile auch als gedrucktes Buch erhältlich:
    The Java Tutorial: A Short Course on the Basics (6th ed. 2014), ISBN 0134034082



Java ist auch eine Insel…

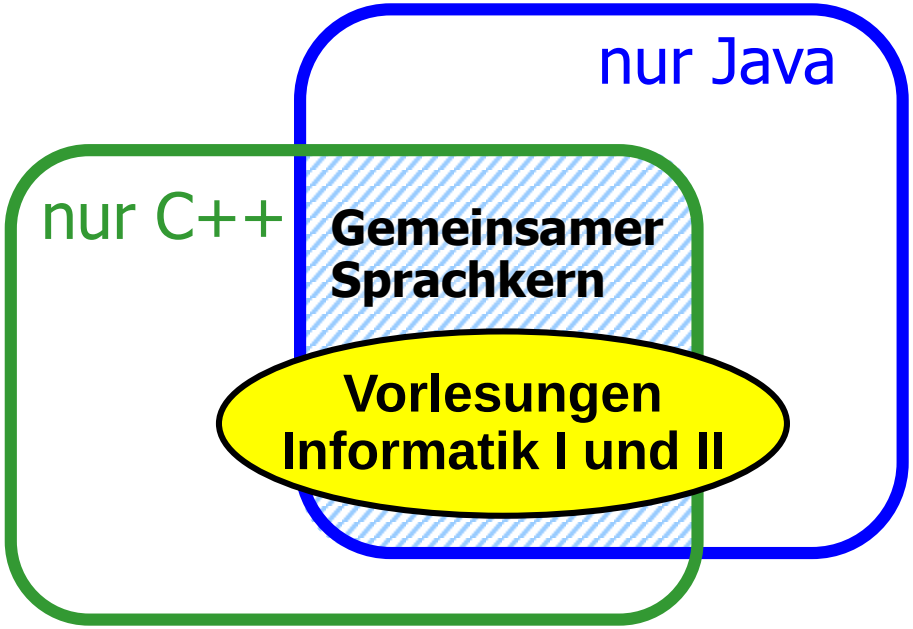


Oben: Relative Häufigkeit von Wörtern im deutschsprachigen Wortschatz von Büchern entsprechend Google. Rechts: Im Vergleich dazu die Zeitungsdatenbank des DWDS. Man erkennt: In Fachbüchern ist „Java" neuerdings stark vertreten; bei den Zeitungen ist hingegen weniger die Programmiersprache als die Insel 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
- Ebenso auch Varianten wie C#

Java: „Removed from C/C++“

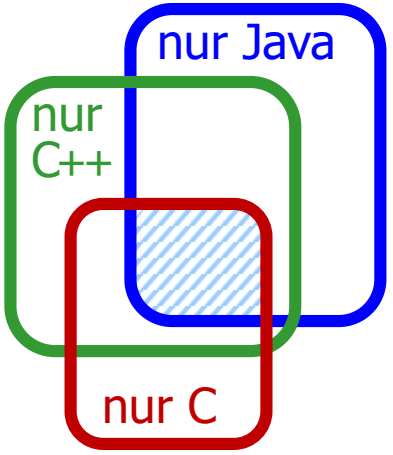
- Zeigerarithmetik, malloc (aber: Arrays und new)
- Destruktoren, free, delete (aber: Garbage-Collector; finalize)
- Funktionen (statt dessen: Methoden)
- Überladen von Operatoren
- Implizite Typkonvertierung
- sizeof x (statt dessen: x.length)
- Strukturen, union (statt dessen: Klassen / Objekte)
- Templates
- Mehrfachvererbung (statt dessen: Interfaces)
- Friends (aber: „friendly access“ innerhalb von Paketen)
- Präprozessor: TYPEDEF, ...
- #DEFINE und const (statt dessen: final)
- goto (statt dessen: 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

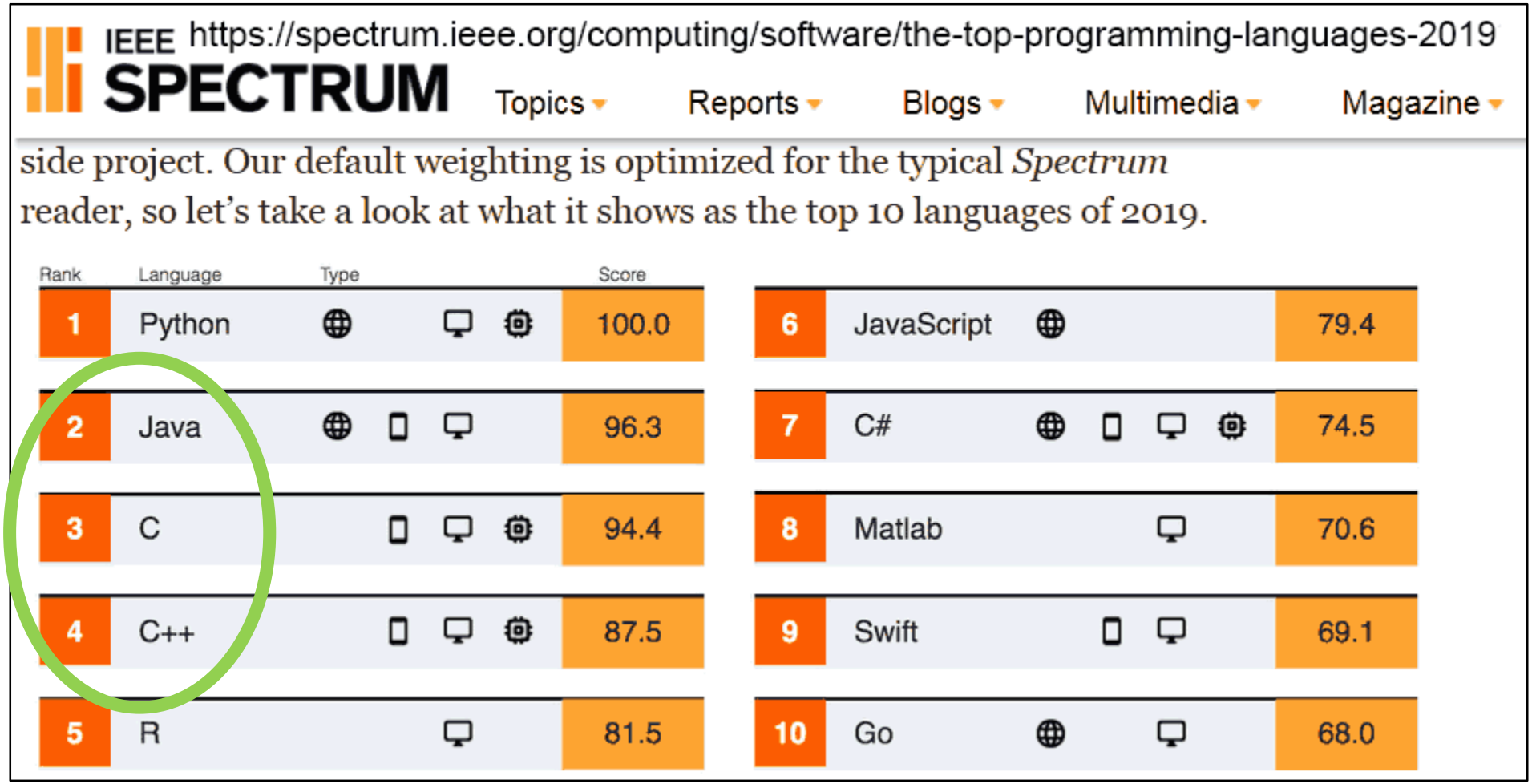
Java: Neu gegenüber C++

- 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

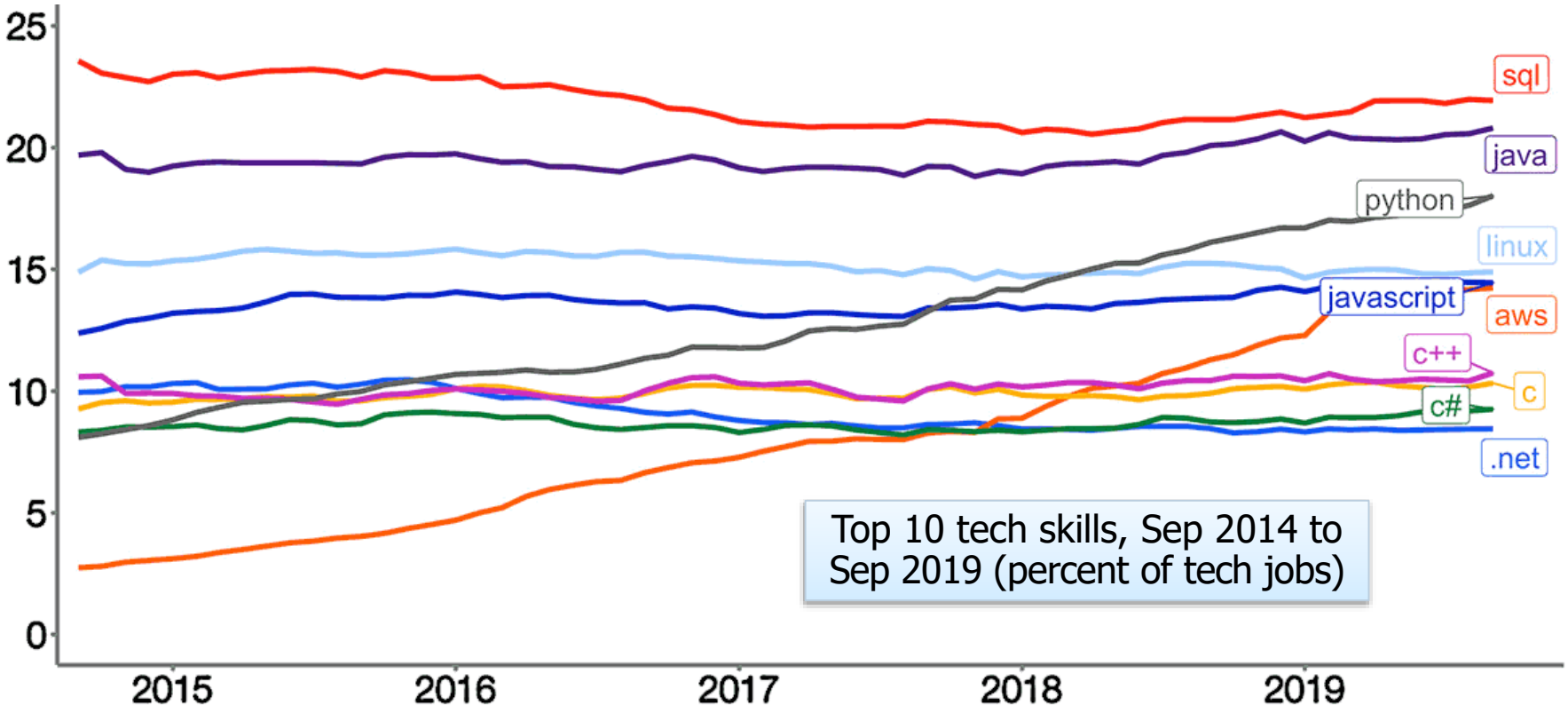


The Top Programming Languages 2019



SQL, Java Top List of Most In-Demand Tech Skills

IEEE Spectrum, 19 Nov. 2019



What tech skills do U.S. employers want? Researchers at job search site Indeed took a deep dive into its database to answer that question. Indeed's team considered U.S. English-language jobs posted on the site between September 2014 and September 2019; those postings encompassed 571 tech skills. Indeed's researchers note that the big jumps in demand for engineers skilled in Python stems from the boom in data scientist and engineer jobs, which disproportionately use Python.

Java-Entwicklung und wichtige Sprachversionen

- 1995: Version 1.0 (Sun Microsystems: James Gosling und Andere)
- 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 5.0 (bzw. 1.5 oder „Java 2 Platform Standard Edition 5.0“)
  - U.a. generische Typen und Aufzählungen (enum)
  - Implizite Umwandlung einfacher Datentypen in Objekte und zurück
- 2014: Java 8
  - Lambda-Ausdrücke als funktionale Sprachelemente; Default-Implementierung bei Interface-Methoden
- 2019: Java 12

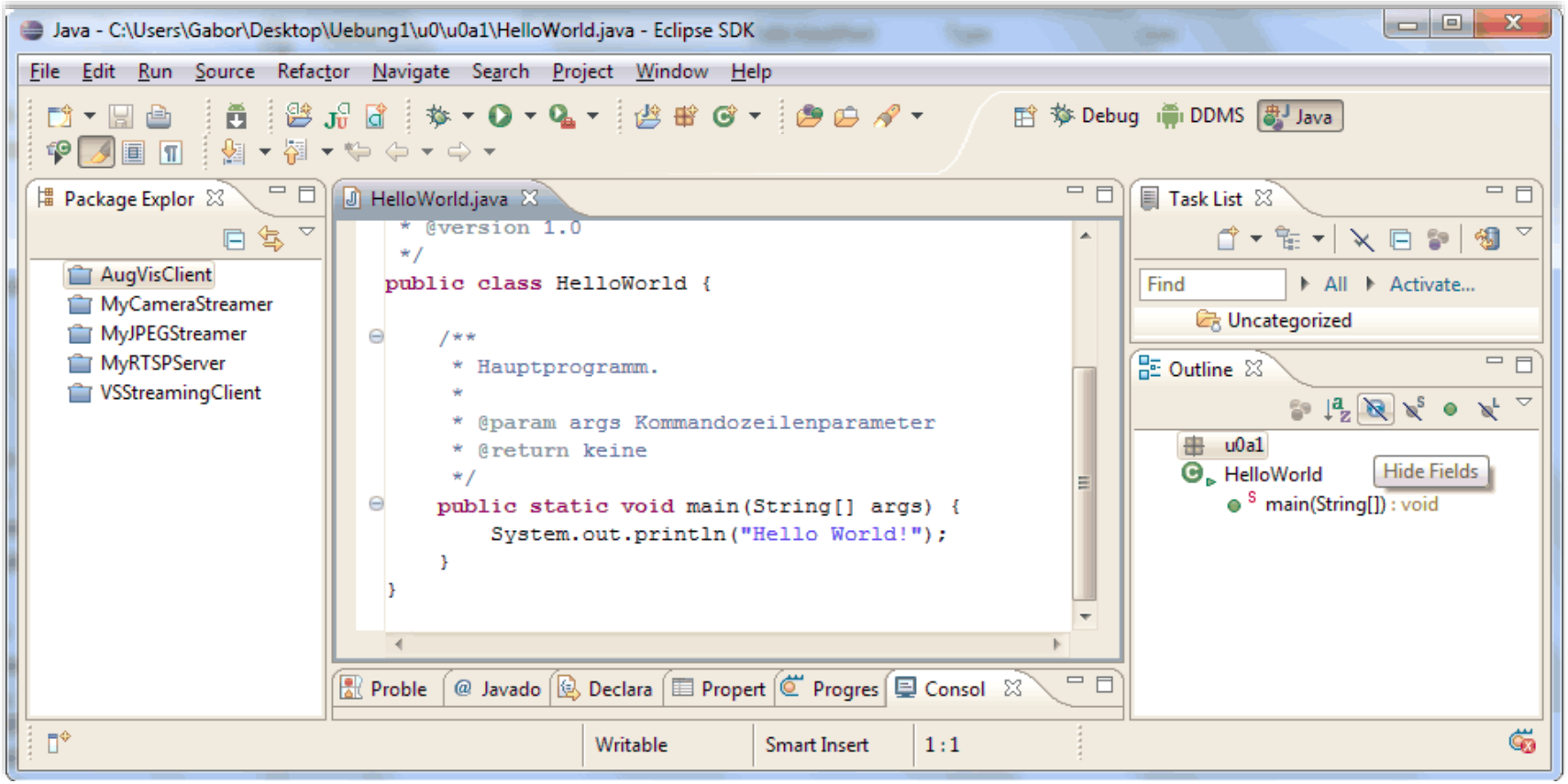
2010 wurde Sun Microsystems von Oracle übernommen

Für unsere Zwecke sind die Unterschiede der Versionen weitgehend irrelevant

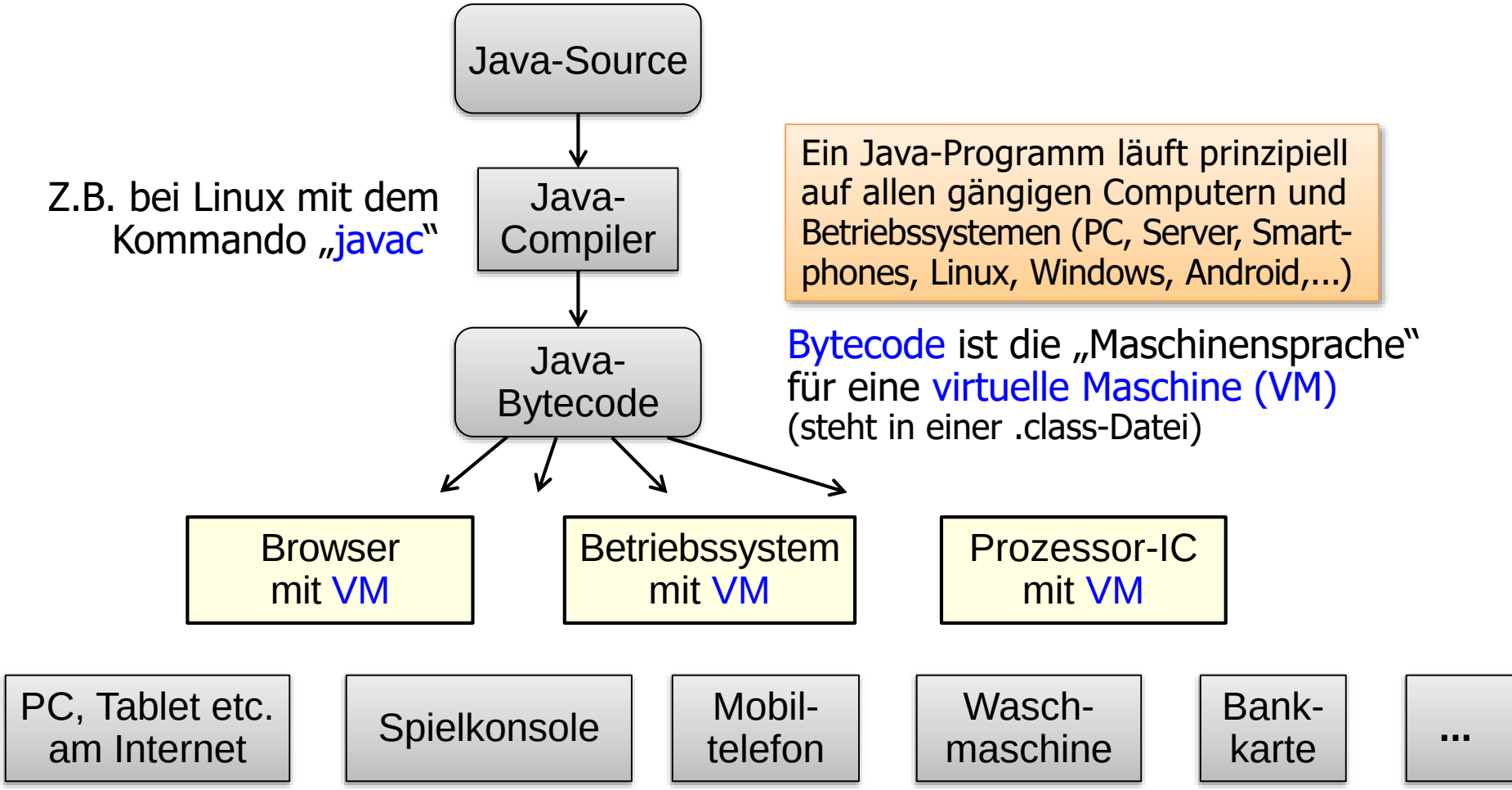
Programmieren mit Java

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



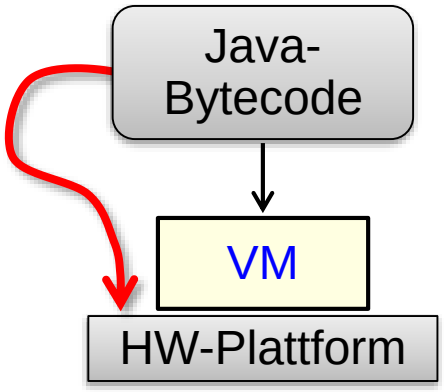
Java: Plattformunabhängigkeit durch Bytecode-Interpretation



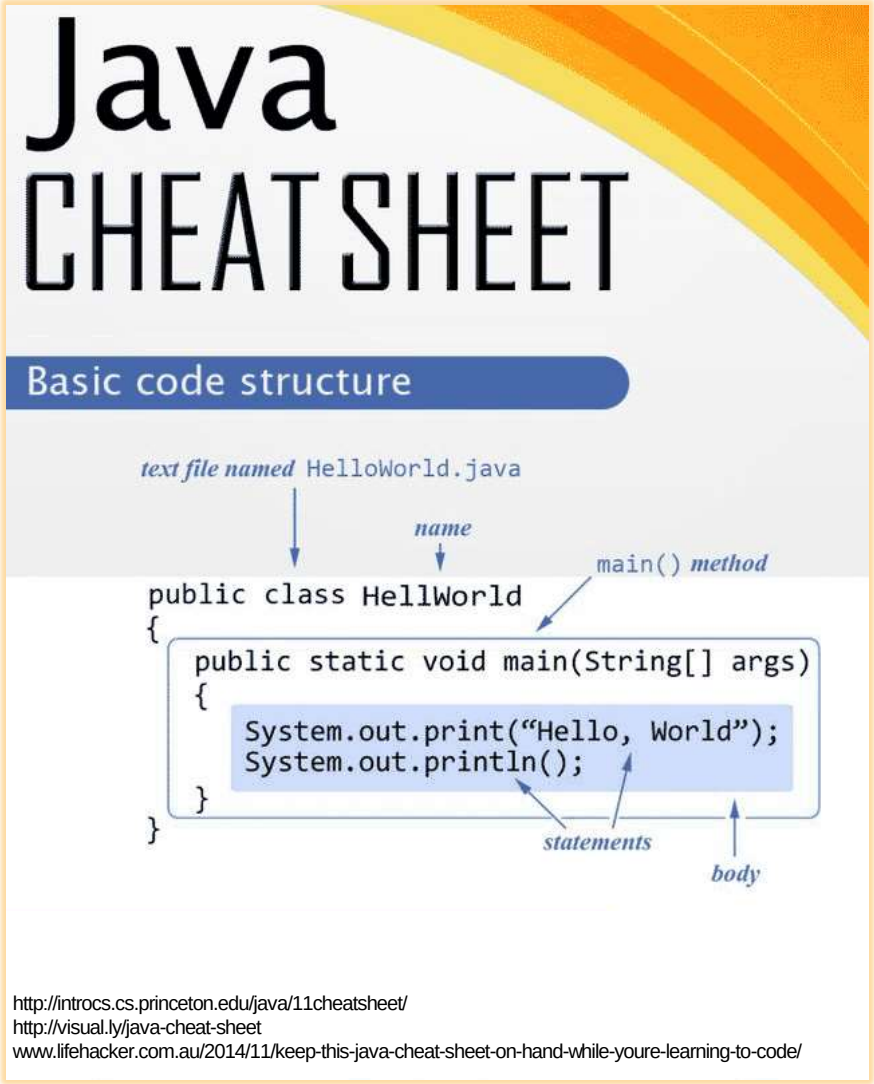
Die Virtuelle Maschine (VM)

- 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“

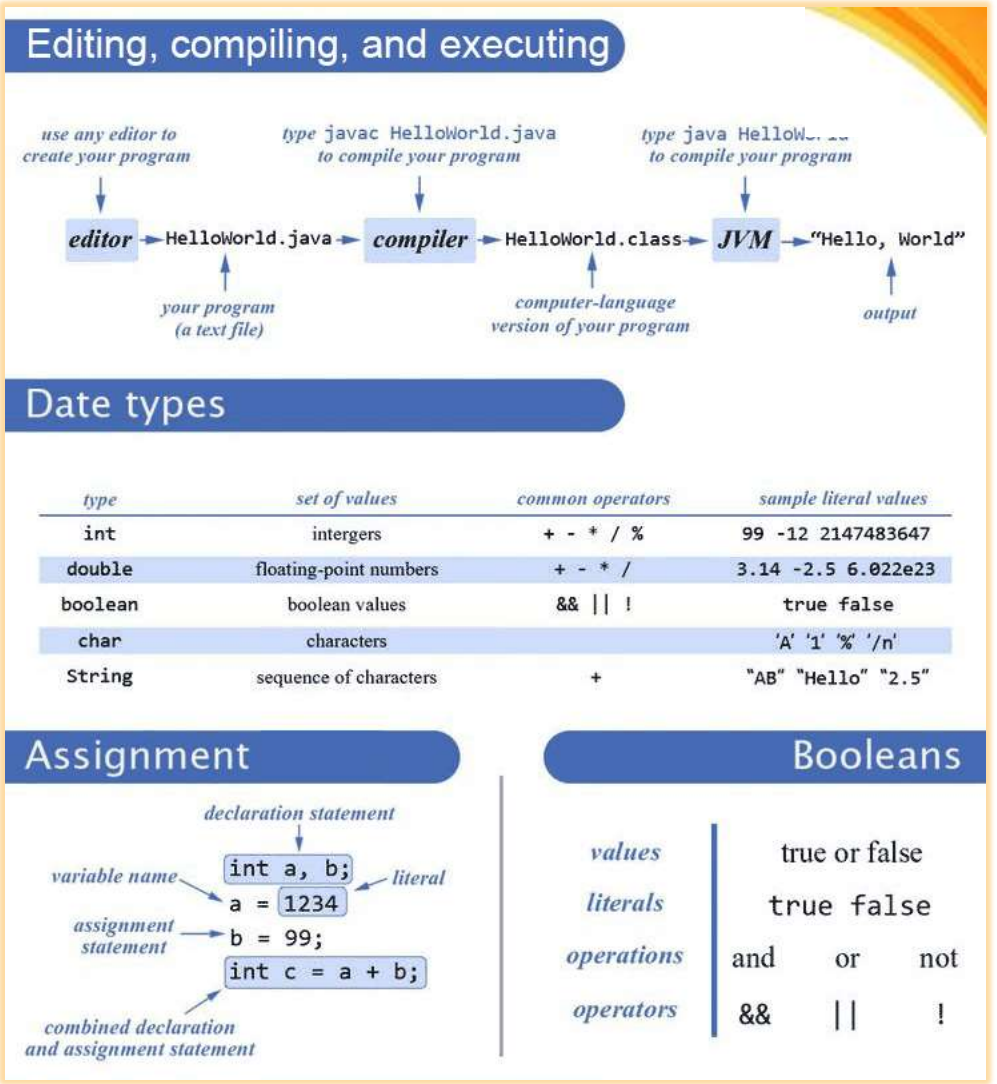
- Effizienzverlust durch Interpretation?
  - Evtl. statt dessen den Bytecode in Zielsprache (weiter-)übersetzen
  - Zumindest wiederholt gebrauchte Programmteile „just in time“



Java ganz kurzgefasst im „Cheat Sheet“



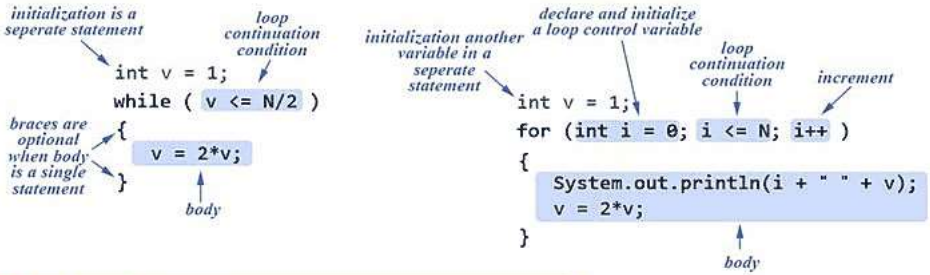
http://introcs.cs.princeton.edu/java/11cheatsheet/
http://visual.ly/java-cheat-sheet
www.lifehacker.com.au/2014/11/keep-this-java-cheat-sheet-on-hand-while-youre-learning-to-code/



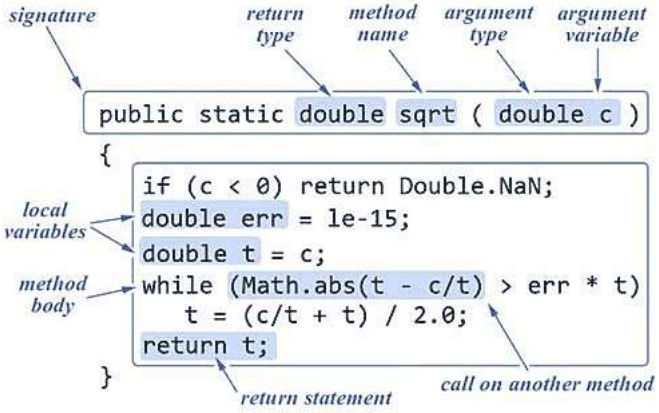
Comparison

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

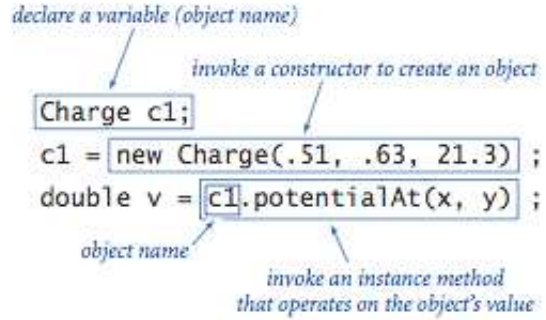
Loops



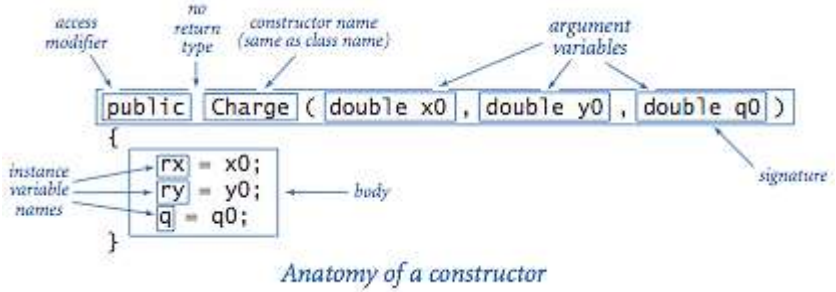
Methods



Using an object

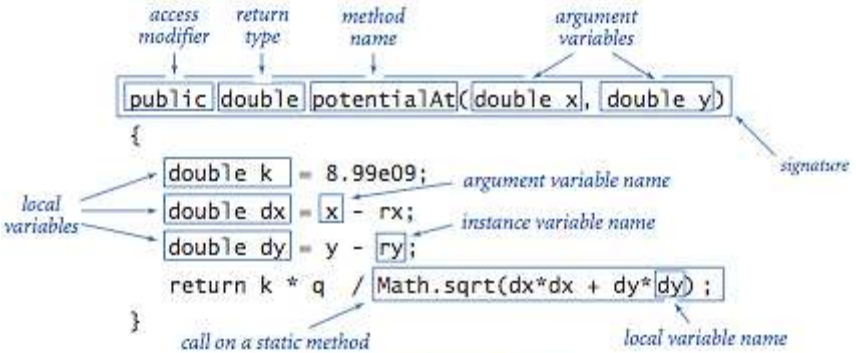


Constructors



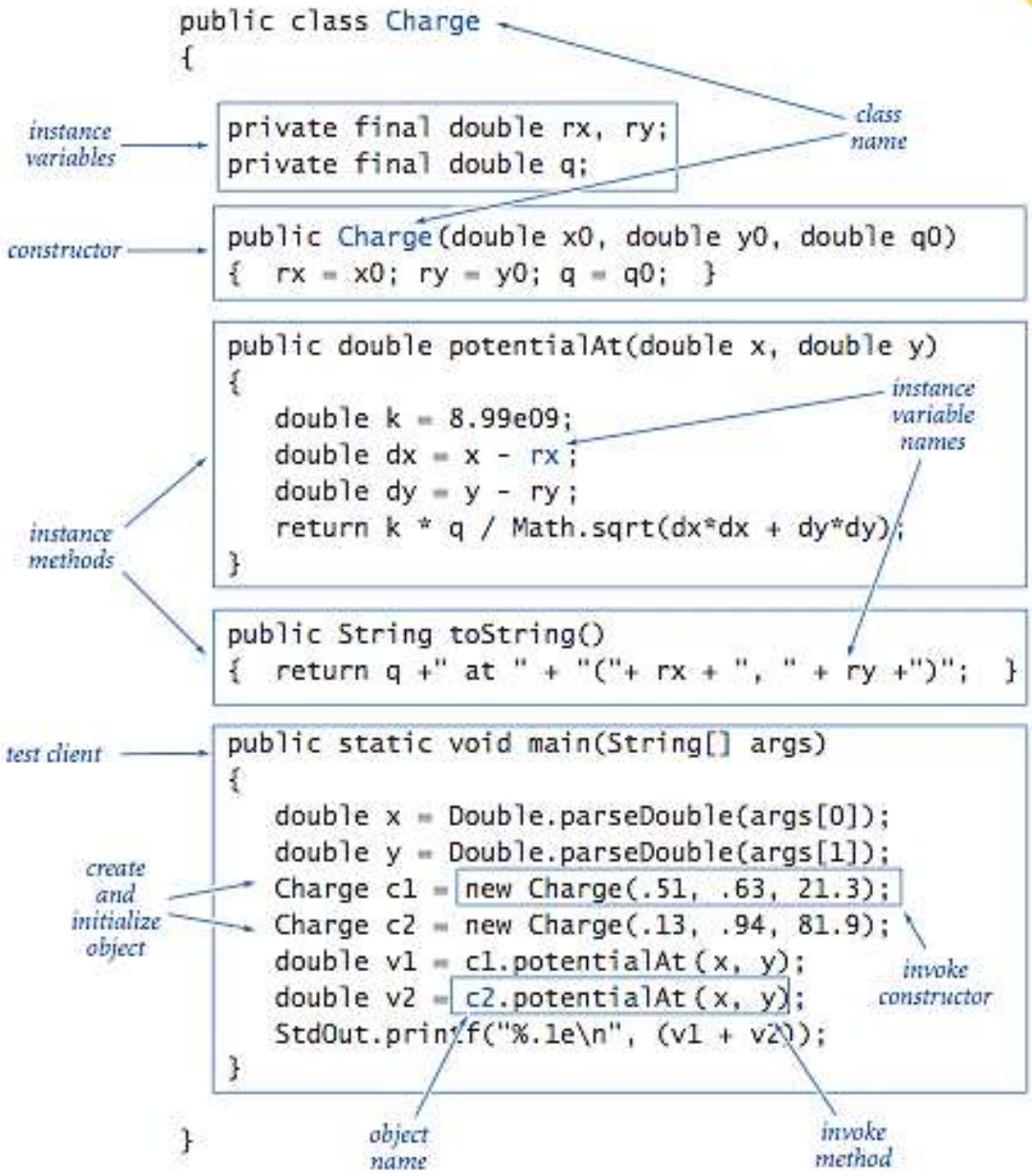
Anatomy of a constructor

Instance methods



Anatomy of an instance method

Classes

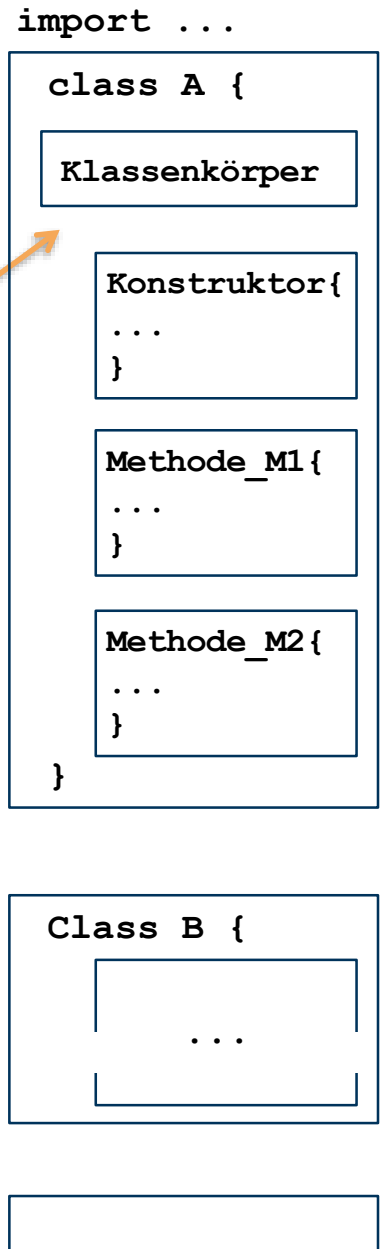

public class Charge
{
private final double rx, ry;
private final double q;
public Charge(double x0, double y0, double q0)
{ rx = x0; ry = y0; q = q0; }
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy);
}
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

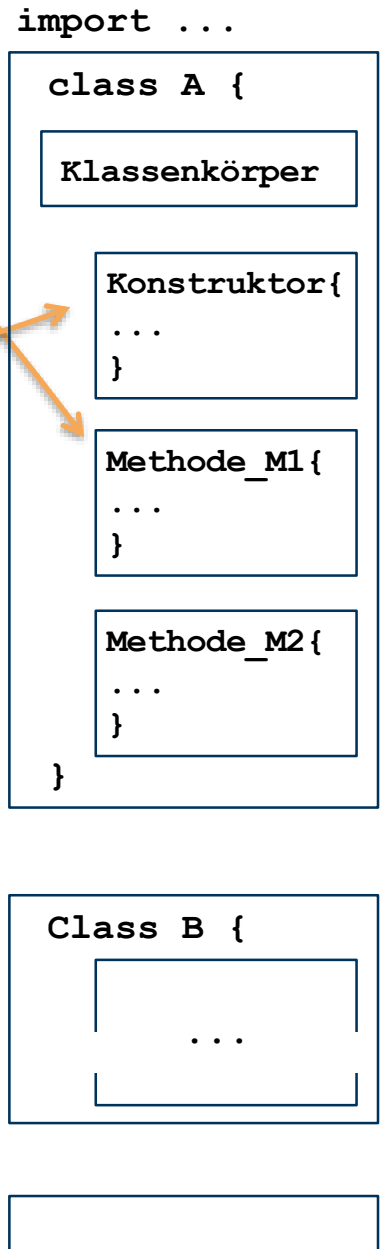
- Der Klassenkörper enthält
  - Instanzen- und Klassenvariablen („Attribute“)
  - Benannte Konstanten
  - Klassenbezogene („static“) Methoden

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)

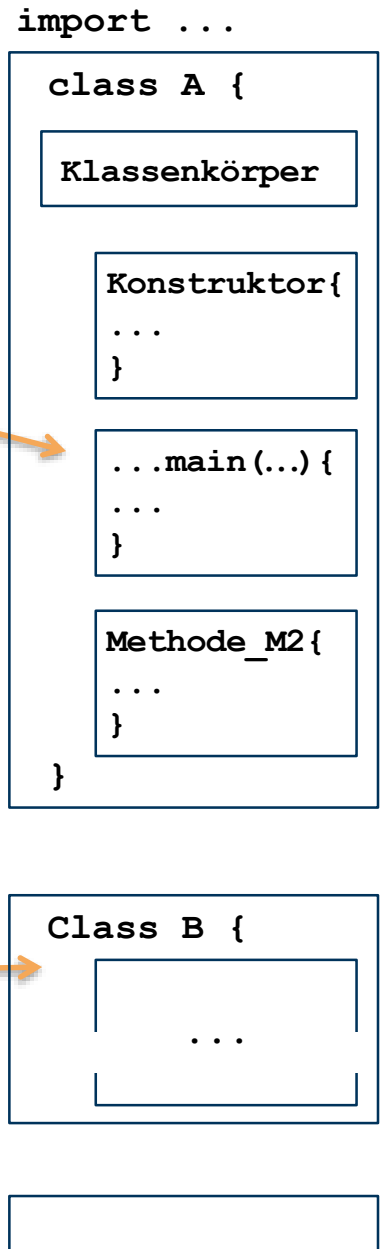


Java-Programmstruktur (3)

- Bei eigenständigen Programmen (d.h. keinen Applets) 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

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

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

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

- Zeichen („Unicode“)
  - **char** (16 Bits, UCS-2)

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

- Wahrheitswerte
  - **boolean**

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

Unmittelbar sind diese Datentypen nicht kompatibel!

Bei C++ heisst das kürzer nur „bool“

Einfache Datentypen in Java (2)

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

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`

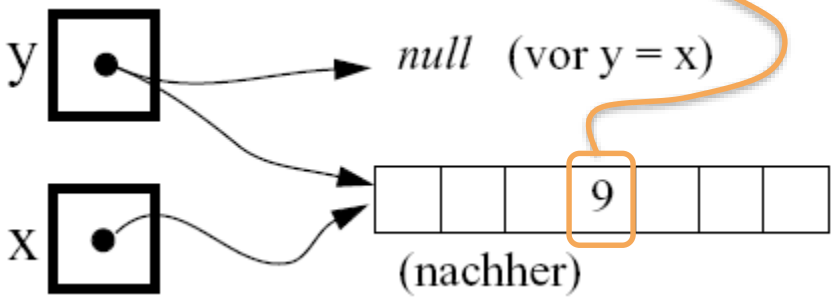
Wir halten uns im Folgenden aber nicht immer an diese Konventionen…

Arrays

- Arrays 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 einlesen
int [] tabelle = new int [n];
```

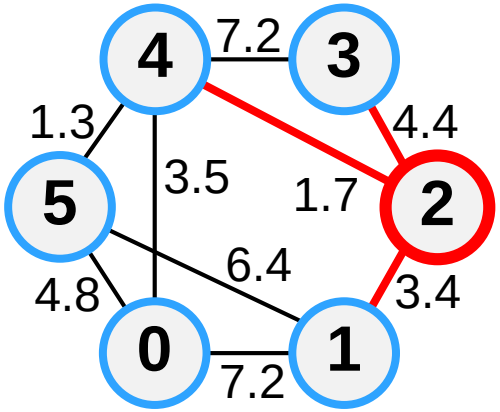
  - Einmal angelegt, kann sich die Länge aber nicht mehr ändern!
    Flexible Arrays (Datentyp „ArrayList“; entspr. „Vector“ in C++), besprechen wir später

- Mehrdimensionale Arrays:

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

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

2D-Array-Beispiel: Adjazenzmatrix

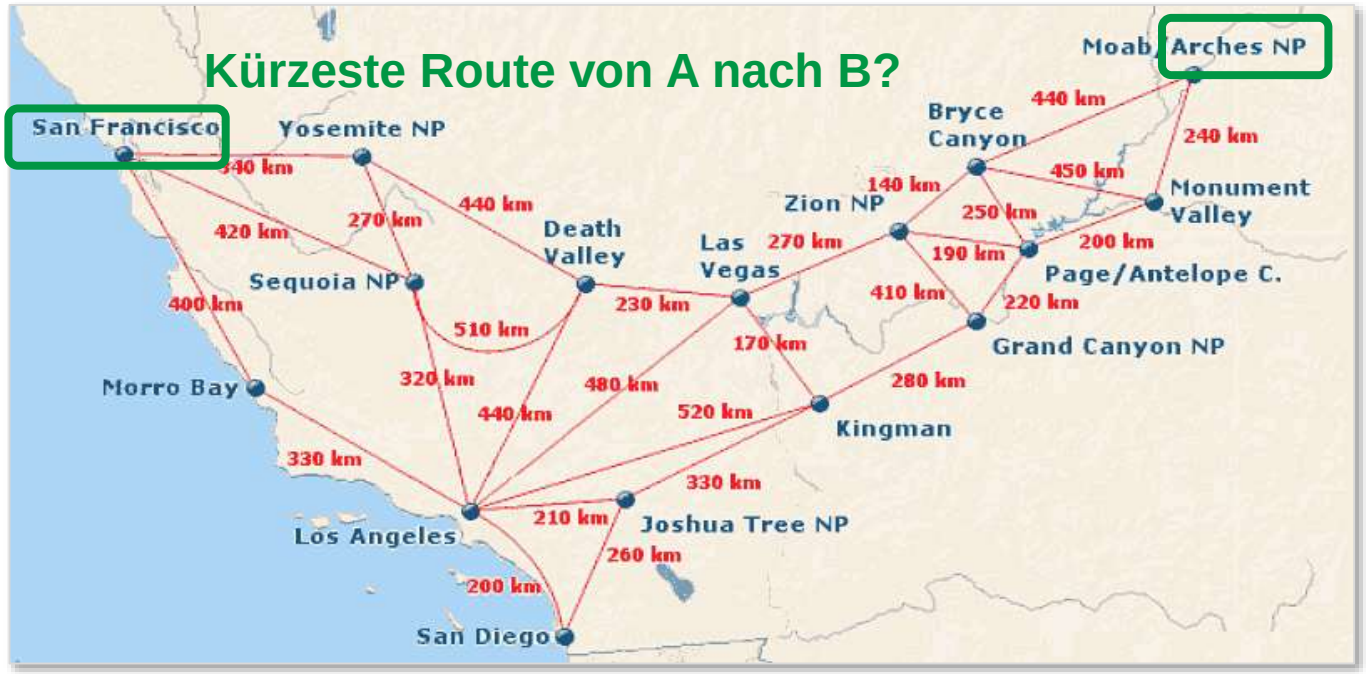


Entfernungstabelle

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 symmet-risch; 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);
```

Die Grundidee des sogen. „Dijkstra-Algorithmus" besprechen wir später

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

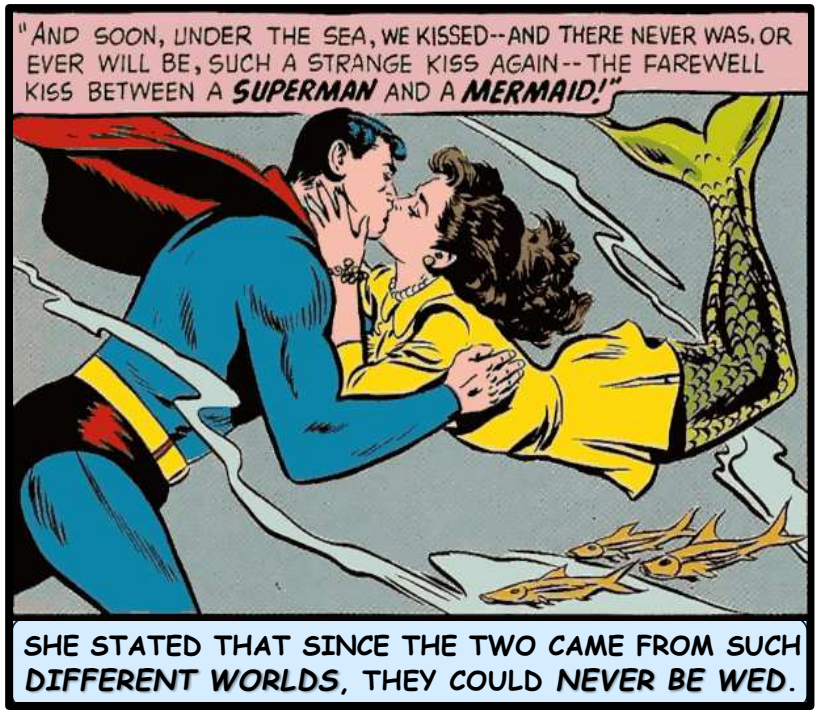
```
float d = 5 + 3.2;
```

- 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

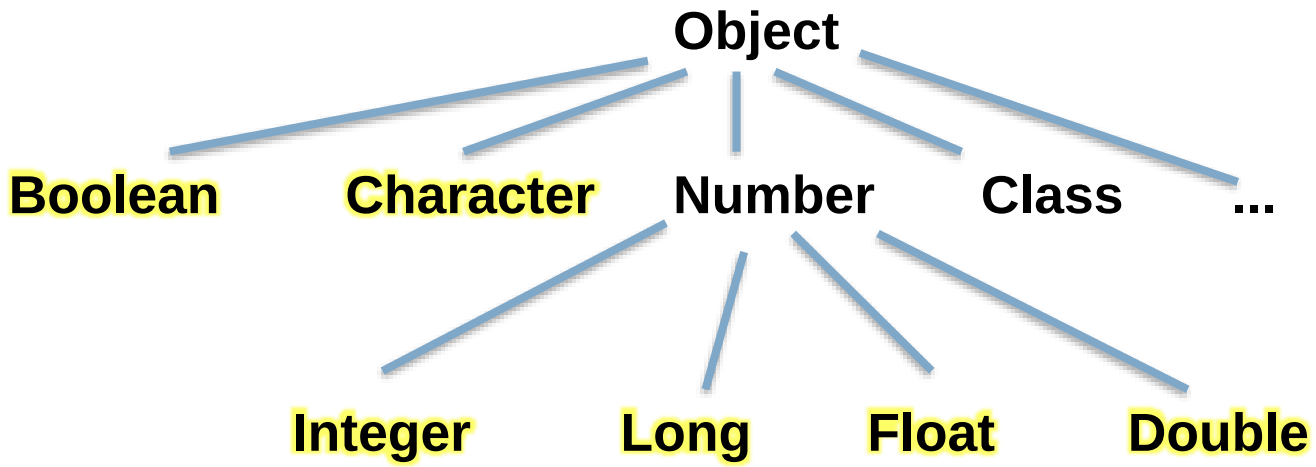
Später mehr dazu (type cast bei Polymorphie)



Das Problem inkompatibler Typen

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

```
int count = 0;
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

Eingabe von Daten (einzelne Zeichen)

Konkatenation von strings

```
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);
```

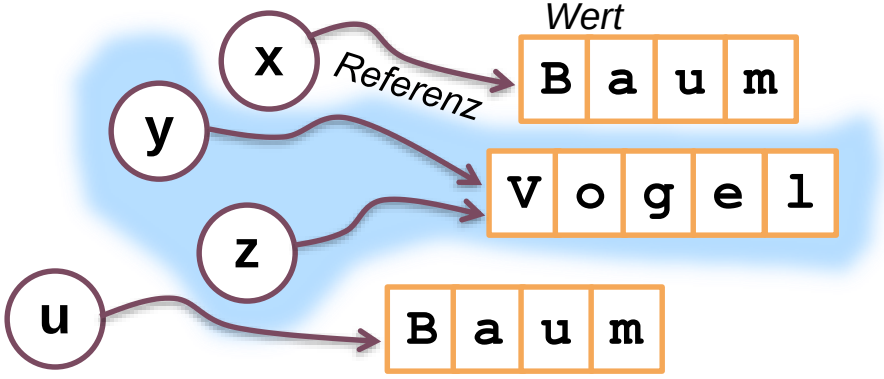
Alternativ zur ersten Zeile geht es auch so:

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

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/13/docs/api/java.base/java/lang/String.html

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 statt dessen 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!

Resümee des Kapitels

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

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