

## 4. Effizienz von Algorithmen

Effizienz von Algorithmen, Random Access Machine Modell, Funktionenwachstum, Asymptotik [Cormen et al, Kap. 2.2,3,4.2-4.4 | Ottman/Widmayer, Kap. 1.1]

### Effizienz von Algorithmen

#### Ziele

- Laufzeitverhalten eines Algorithmus maschinenunabhängig quantifizieren.
- Effizienz von Algorithmen vergleichen.
- Abhängigkeit von der Eingabegrösse verstehen.

54

55

### Technologiemodell

#### *Random Access Machine (RAM)*

- Ausführungsmodell: Instruktionen werden der Reihe nach (auf einem Prozessorkern) ausgeführt.
- Speichermodell: Konstante Zugriffszeit.
- Elementare Operationen: Rechenoperation (+, -, ·, ...) , Vergleichsoperationen, Zuweisung / Kopieroperation, Flusskontrolle (Sprünge)
- Einheitskostenmodell: elementare Operation hat Kosten 1.
- Datentypen: Fundamentaltypen wie grössenbeschränkte Ganzzahl oder Fließkommazahl.

56

### Asymptotisches Verhalten

Genauere Laufzeit lässt sich selbst für kleine Eingabedaten kaum voraussagen.

- Betrachten das asymptotische Verhalten eines Algorithmus.
- Ignorieren alle konstanten Faktoren.

#### Beispiel

Eine Operation mit Kosten 20 ist genauso gut wie eine mit Kosten 1. Lineares Wachstum mit Steigung 5 ist genauso gut wie lineares Wachstum mit Steigung 1.

57

## Oberflächlich

### 4.1 Funktionenwachstum

$\mathcal{O}$ ,  $\Theta$ ,  $\Omega$  [Cormen et al, Kap. 3; Ottman/Widmayer, Kap. 1.1]

Verwende die asymptotische Notation zur Kennzeichnung der Laufzeit von Algorithmen

Wir schreiben  $\Theta(n^2)$  und meinen, dass der Algorithmus sich für grosse  $n$  wie  $n^2$  verhält: verdoppelt sich die Problemgrösse, so vervierfacht sich die Laufzeit.

58

59

## Genauer: Asymptotische obere Schranke

Gegeben: Funktion  $f : \mathbb{N} \rightarrow \mathbb{R}$ .

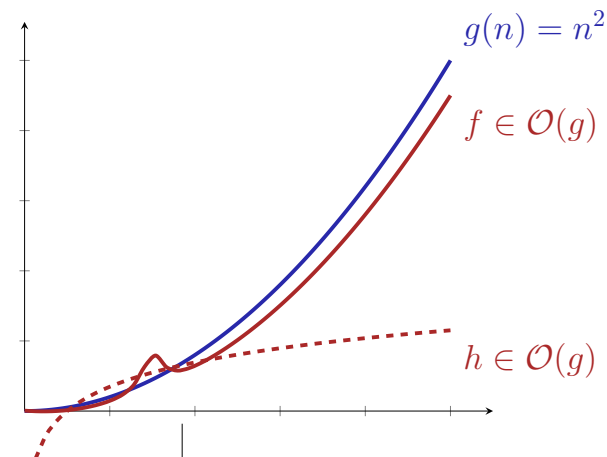
Definition:

$$\mathcal{O}(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$$

Schreibweise:

$$\mathcal{O}(g(n)) := \mathcal{O}(g(\cdot)) = \mathcal{O}(g).$$

## Anschauung



60

61

## Beispiele

$$\mathcal{O}(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} : 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$$

$f(n)$	$f \in \mathcal{O}(?)$	Beispiel
$3n + 4$	$\mathcal{O}(n)$	$c = 4, n_0 = 4$
$2n$	$\mathcal{O}(n)$	$c = 2, n_0 = 0$
$n^2 + 100n$	$\mathcal{O}(n^2)$	$c = 2, n_0 = 100$
$n + \sqrt{n}$	$\mathcal{O}(n)$	$c = 2, n_0 = 1$

$$f_1 \in \mathcal{O}(g), f_2 \in \mathcal{O}(g) \Rightarrow f_1 + f_2 \in \mathcal{O}(g)$$

62

63

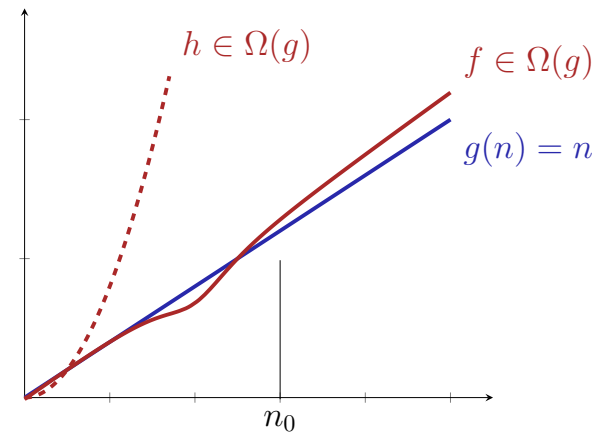
## Umkehrung: Asymptotische untere Schranke

Gegeben: Funktion  $f : \mathbb{N} \rightarrow \mathbb{R}$ .

Definition:

$$\Omega(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} : 0 \leq c \cdot g(n) \leq f(n) \forall n \geq n_0\}$$

## Beispiel



64

65

## Asymptotisch scharfe Schranke

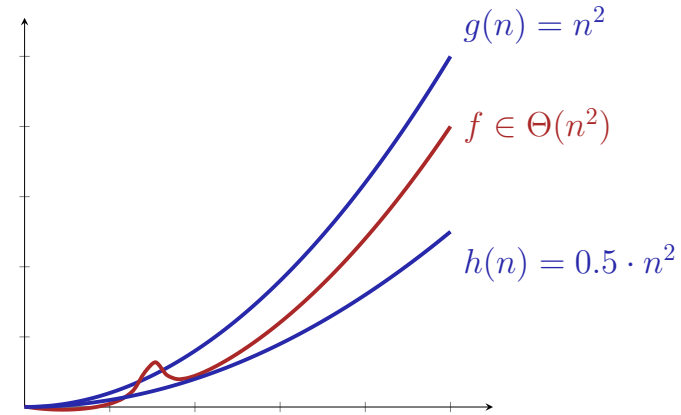
Gegeben Funktion  $f : \mathbb{N} \rightarrow \mathbb{R}$ .

Definition:

$$\Theta(g) := \Omega(g) \cap \mathcal{O}(g).$$

Einfache, geschlossene Form: Übung.

## Beispiel



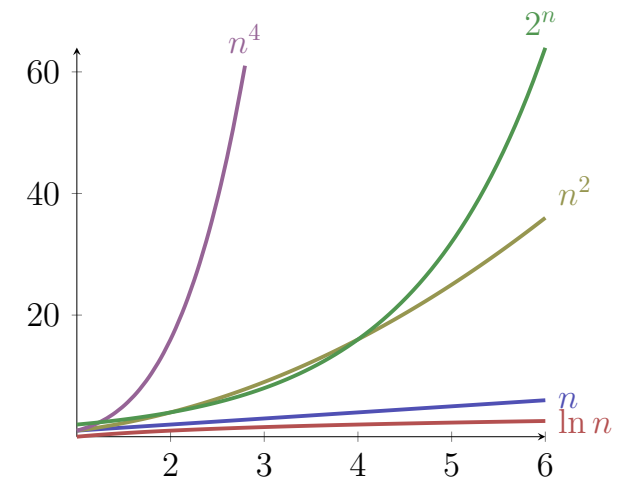
66

67

## Wachstumsbezeichnungen

$\mathcal{O}(1)$	beschränkt	Array-Zugriff
$\mathcal{O}(\log \log n)$	doppelt logarithmisch	Binäre sortierte Suche interpoliert
$\mathcal{O}(\log n)$	logarithmisch	Binäre sortierte Suche
$\mathcal{O}(\sqrt{n})$	wie die Wurzelfunktion	Primzahltest (naiv)
$\mathcal{O}(n)$	linear	Unsortierte naive Suche
$\mathcal{O}(n \log n)$	superlinear / loglinear	Gute Sortieralgorithmen
$\mathcal{O}(n^2)$	quadratisch	Einfache Sortieralgorithmen
$\mathcal{O}(n^c)$	polynomial	Matrixmultiplikation
$\mathcal{O}(2^n)$	exponentiell	Travelling Salesman Dynamic Programming
$\mathcal{O}(n!)$	faktoriell	Travelling Salesman naiv

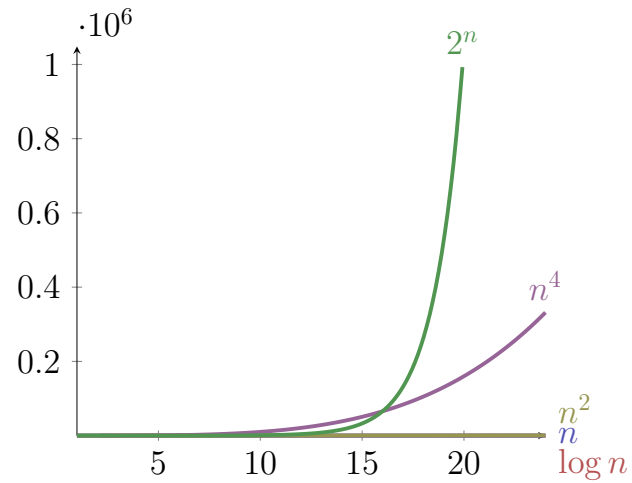
## Kleine $n$



68

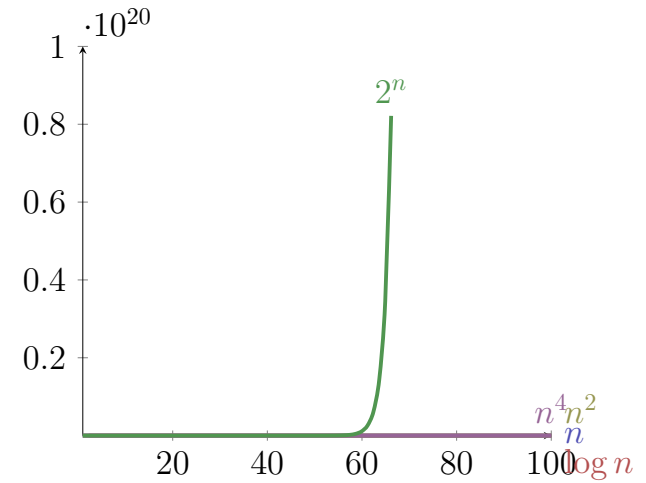
69

## Grössere $n$



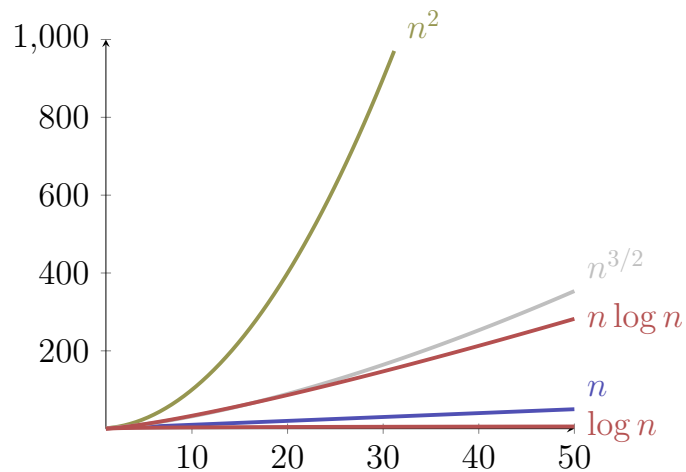
70

## “Grosse” $n$



71

## Logarithmen!



72

## Zeitbedarf

Annahme: 1 Operation =  $1\mu s$ .

Problemgrösse	1	100	10000	$10^6$	$10^9$
$\log_2 n$	$1\mu s$	$7\mu s$	$13\mu s$	$20\mu s$	$30\mu s$
$n$	$1\mu s$	$100\mu s$	$1/100s$	$1s$	17 Minuten
$n \log_2 n$	$1\mu s$	$700\mu s$	$13/100\mu s$	$20s$	8.5 Stunden
$n^2$	$1\mu s$	$1/100s$	1.7 Minuten	11.5 Tage	317 Jahrh.
$2^n$	$1\mu s$	$10^{14}$ Jahrh.	$\approx \infty$	$\approx \infty$	$\approx \infty$

73

## Eine gute Strategie?

... dann kaufe ich mir eben eine neue Maschine! Wenn ich heute ein Problem der Grösse  $n$  lösen kann, dann kann ich mit einer 10 oder 100 mal so schnellen Maschine...

Komplexität	(speed $\times 10$ )	(speed $\times 100$ )
$\log_2 n$	$n \rightarrow n^{10}$	$n \rightarrow n^{100}$
$n$	$n \rightarrow 10 \cdot n$	$n \rightarrow 100 \cdot n$
$n^2$	$n \rightarrow 3.16 \cdot n$	$n \rightarrow 10 \cdot n$
$2^n$	$n \rightarrow n + 3.32$	$n \rightarrow n + 6.64$

## 5. Java - Sprachkonstrukte I

Namen und Bezeichner, Variablen, Zuweisungen, Konstanten, Datentypen, Operationen, Auswerten von Ausdrücken, Konversionen, Kontrollfluss, Klassen, statische Methoden

## Beispiele

- $n \in \mathcal{O}(n^2)$  korrekt, aber ungenau:  
 $n \in \mathcal{O}(n)$  und sogar  $n \in \Theta(n)$ .
- $3n^2 \in \mathcal{O}(2n^2)$  korrekt, aber unüblich:  
Konstanten weglassen:  $3n^2 \in \mathcal{O}(n^2)$ .
- $2n^2 \in \mathcal{O}(n)$  ist falsch:  $\frac{2n^2}{cn} = \frac{2}{c}n \xrightarrow{n \rightarrow \infty} \infty$  !
- $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$  ist korrekt
- $\Theta(n) \subseteq \Theta(n^2)$  ist falsch:  $n \notin \Omega(n^2) \supset \Theta(n^2)$

74

75

## Namen und Bezeichner

Ein Programm (also Klasse) braucht einen Namen

```
public class SudokuSolver { ...
```

- Konvention für Klassennamen: *CamelCase* benutzen  $\rightarrow$  Wörter sind zusammengeschrieben mit jeweils einem Grossbuchstaben

Erlaubte Namen für "Entitäten" im Programm:

- Namen beginnen mit einem *Buchstaben* oder `_` oder `$`
- Danach Folge von *Buchstaben*, *Zahlen* oder `_` oder `$`

76

77

## Namen - was ist erlaubt

Valid identifiers (green background):

- \_myName
- TheCure
- \_\_AN\$WE4\_1S\_42\_\_
- \$bling\$

Invalid identifiers (red background):

- me@home
- strictfp ?!
- 49ers
- side-swipe
- Ph.D's

78

## Schlüsselwörter

Folgende Wörter werden von der Sprache bereits benützt und können deshalb nicht als Namen benützt werden:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

79

## Variablen und Konstanten

- *Behälter* für einen Wert.
- Haben einen *Datentyp* und einen *Namen*
- Der Datentyp bestimmt, welche Art von Werten in der Variable erlaubt sind

<code>int x</code>	<code>int y</code>	<code>float f</code>	<code>char c</code>
23	42	0.0f	'a'

Deklaration in Java:

```
int x = 23, y = 42;  
float f;  
char c = 'a';
```

↑  
Initialisierung

80

## Konstanten

- Schlüsselwort `final`
- Der Wert der Variable kann genau einmal gesetzt werden.

### Beispiel

```
final int maxSize = 100;
```

**Tip:** Benützen Sie `final` immer, es sei denn der Wert muss tatsächlich veränderlich sein.

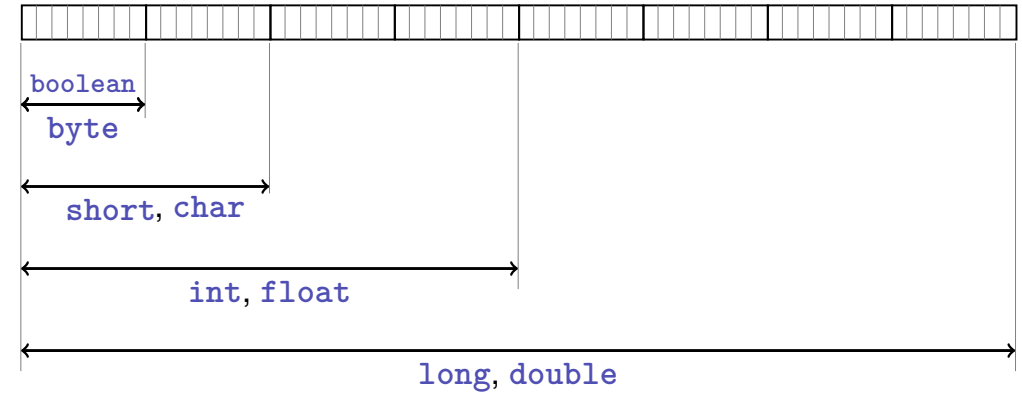
81

## Standardtypen

Datentyp	Definition	Wertebereich	Initialwert
byte	8-bit Ganzzahl	$-128, \dots, 127$	0
short	16-bit Ganzzahl	$-32'768, \dots, 32'767$	0
int	32-bit Ganzzahl	$-2^{31}, \dots, 2^{31} - 1$	0
long	64-bit Ganzzahl	$-2^{63}, \dots, 2^{63} - 1$	0L
float	32-bit Fließkommazahl	$\pm 1.4E^{-45}, \dots, \pm 3.4E^{+38}$	0.0f
double	64-bit Fließkommazahl	$\pm 4.9E^{-324}, \dots, \pm 1.7E^{+308}$	0.0d
boolean	Wahrheitswert	true, false	false
char	Unicode-16 Zeichen	'\u0000', ..., 'a', 'b', ..., '\uFFFF'	'\u0000'
String	Zeichenkette	$\infty$	null

## Typen und Speicherbelegung

Zur Erinnerung: Speicherzellen enthalten 1 Byte = 8 bit



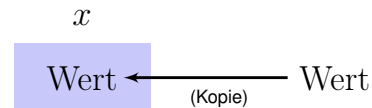
82

83

## Wertzuweisungen

Kopiert einen Wert in die Variable  $x$

- In Pseudocode:  $x \leftarrow \text{Wert}$
- In Java:  $x = \text{Wert}$



"=" ist der Zuweisungsoperator *und nicht ein Vergleich!*

`int y = 42` ist also Deklaration + Wertzuweisung in einem.

## Wertzuweisungen

### Beispiele

```
int a = 3;
double b;

b = 3.141;

int c = a = 0;

String name = "Inf";
```

Eine *verschachtelte* Zuweisung:  
Der Ausdruck `a = 0` speichert den Wert 0 in Variable a. *und gibt den Wert danach zurück*

84

85



## Arithmetische Binäre Operatoren

Infix Notation:  $x \text{ op } y$  mit folgenden Operatoren

op: + - \* / %

↑  
Modulo

Division  $x / y$ : Ganzzahldivision falls  $x$  und  $y$  Ganzzahlen sind.

### Beispiele

Ganzzahldivision und Modulo

- $5 / 3$  ergibt 1      $-5 / 3$  ergibt -1
- $5 \% 3$  ergibt 2      $-5 \% 3$  ergibt -2

86

## Arithmetische Zuweisung

$x = x + y$

⇕

$x += y$

### Beispiele:

```
x -= 3;      // x = x - 3
name += "x" // name = name + "x"
num *= 2;   // num = num * 2
```

Analog für -, \*, /, %

87

## Arithmetische Unäre Operatoren

Prefix Notation:  $+ x$  oder  $- x$

Unäre Operatoren binden stärker als binäre.

### Beispiele

Angenommen  $x$  ist 3

- $2 * -x$  ergibt -6
- $-x - +1$  ergibt -4

88

## Inkrement/Dekrement Operatoren

Inkrement Operatoren  $++x$  und  $x++$  haben den gleichen *Effekt*:

$$x \leftarrow x + 1.$$

Aber unterschiedliche Rückgabewerte:

- *Präfixoperator*  $++x$  gibt *neuen* Wert zurück:

$a = ++x; \iff x = x + 1; a = x;$

- *Postfixoperator*  $x++$  gibt den *alten* Wert zurück:

$a = x++; \iff temp = x; x = x + 1; a = temp;$

Analog für  $x--$  und  $--x$ .

89

## Inkrement/Dekrement Operatoren

### Beispiele

Angenommen x ist initial 2

- `y = ++x * 3` ergibt: x ist 3 und y ist 9
- `y = x++ * 3` ergibt: x ist 3 und y ist 6

## Ausdrücke (Expressions)

- repräsentieren *Berechnungen*
- sind entweder *primär*
- oder *zusammengesetzt* ...
- ... aus anderen Ausdrücken, mit Hilfe von *Operatoren*
- sind statisch typisiert

Analogie: Baukasten

90

91

## Ausdrücke (Expressions)

### Beispiele

primär: `"-4.1d"` oder `"x"` oder `"Hi"`

zusammengesetzt: `"x + y"` oder `"f * 2.1f"`

Der Typ von `"12 * 2.1f"` ist `float`

## Celsius zu Fahrenheit

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        System.out.print("Celsius: ");
        Scanner input = new Scanner(System.in);
        int celsius = input.nextInt();
        float fahrenheit = 9 * celsius / 5 + 32;
        System.out.println("Fahrenheit: " + fahrenheit);
    }
}
```

**Beispiel:** 15° Celsius sind 59° Fahrenheit

92

93

## Celsius zu Fahrenheit - Analyse

`9 * celsius / 5 + 32`

- Arithmetischer Ausdruck,
- enthält drei Literale, eine Variable, drei Operatorsymbole

Wie ist der Ausdruck geklammert?

## Regel 1: Präzedenz

Multiplikative Operatoren (`*`, `/`, `%`) haben höhere Präzedenz ("binden stärker") als additive Operatoren (`+`, `-`)

### Beispiel

`9 * celsius / 5 + 32`

bedeutet

`(9 * celsius / 5) + 32`

94

95

## Regel 2: Assoziativität

Arithmetische Operatoren (`*`, `/`, `%`, `+`, `-`) sind linksassoziativ: bei gleicher Präzedenz erfolgt Auswertung von links nach rechts

### Beispiel

`9 * celsius / 5 + 32`

bedeutet

`((9 * celsius) / 5) + 32`

## Regel 3: Stelligkeit

Unäre Operatoren `+`, `-` vor binären `+`, `-`.

### Beispiel

`9 * celsius / + 5 + 32`

bedeutet

`9 * celsius / (+5) + 32`

96

97

## Klammerung

Jeder Ausdruck kann mit Hilfe der

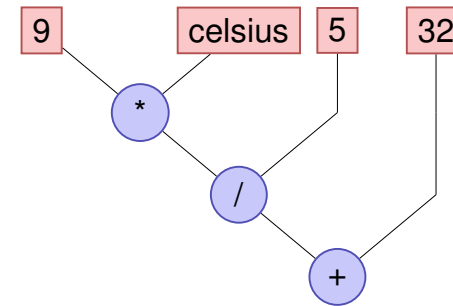
- Assoziativitäten
- Präzedenzen
- Stelligkeiten (Anzahl Operanden)

der beteiligten Operatoren eindeutig geklammert werden.

## Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

`((9 * celsius) / 5) + 32)`



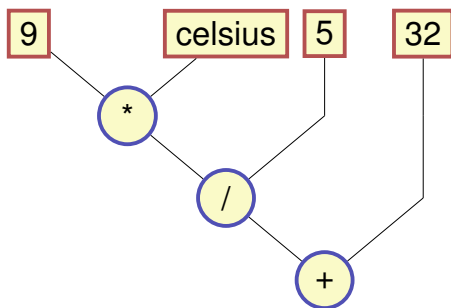
98

99

## Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

`9 * celsius / 5 + 32`

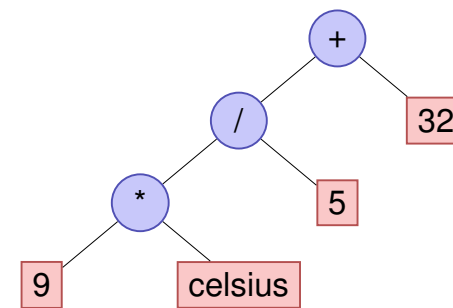


100

## Ausdrucksbäume – Notation

Üblichere Notation: Wurzel oben

`9 * celsius / 5 + 32`



101

# Typsystem

Java hat ein *statisches* Typsystem:

- Alle Typen müssen deklariert werden
- Falls möglich wird Typisierung vom Compiler geprüft ...
- ... ansonsten zur Laufzeit

Vorteil eines statischen Typsystems

- *Fail-fast* Fehler im Programm werden oft schon vom Compiler gefunden
- Verständlicher Code

102

# Typfehler

## Beispiel

```
int pi_ungenau;  
float pi = 3.14f;  
  
pi_ungenau = pi;
```

Compiler Fehler:

```
./Root/Main.java:12: error: incompatible types: possible lossy conversion from float to int  
    pi_ungenau = pi;  
                  ^
```

103

# Explizite Typkonvertierung

## Beispiel

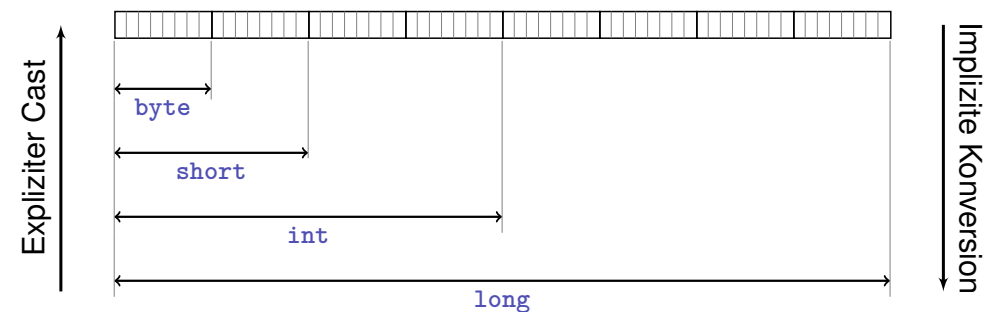
```
int pi_ungenau;  
float pi = 3.14f;  
  
pi_ungenau = (int) pi;
```

Explizite Typkonvertierung mit Typecasting: (typ)

- Statisch typkorrekt, Compiler happy
- Laufzeitverhalten: Je nach Situation  
*Hier: Genauigkeitsverlust: 3.14 ⇒ 3*
- Kann das Programm zur Laufzeit zum Absturz bringen!

104

# Typ Konvertierung - Anschaulich für Ganzzahlen



Potentieller Informationsverlust bei explizitem Cast, da weniger Speicher zur Verfügung.

105

## Typkonversion bei binären Operationen

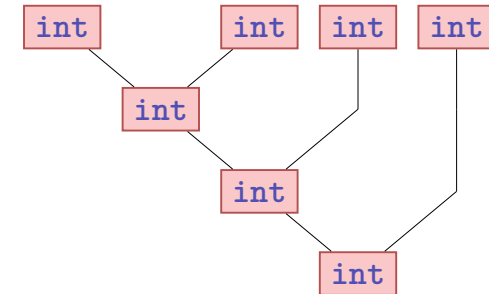
Bei einer binären Operation mit numerischen Operanden von verschiedenem Typ werden die Operanden nach folgenden Regeln konvertiert

- Haben beide Operanden denselben Typ, findet keine Konversion statt
- Ist einer der Operanden `double`, so wird der andere nach `double` konvertiert
- Ist einer der Operanden `float`, so wird der andere nach `float` konvertiert
- Ist einer der Operanden `long`, so wird der andere nach `long` konvertiert
- Ansonsten: Beide Operanden werden nach `int` konvertiert

## Celsius zu Fahrenheit: Typanalyse

Welcher Typ hat das Ergebnis?

```
9 * celsius / 5 + 32
```



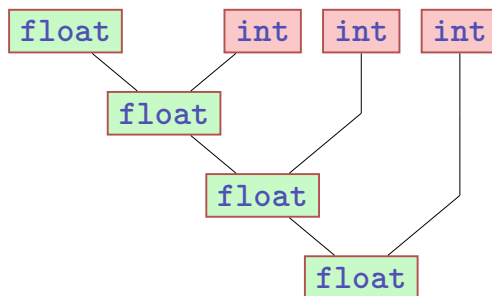
106

107

## Celsius zu Fahrenheit: Typanalyse

Besser: Berechnung und Ergebnis als `float`

```
9f * celsius / 5 + 32
```



## Anweisungen (Statements)

Eine Anweisung ist ...

- vergleichbar mit einem Satz in der natürlichen Sprache.
- eine komplette Ausführungseinheit.
- immer mit einem *Semikolon* abgeschlossen.

### Beispiel

```
f = 9f * celsius / 5 + 32 ;
```

108

109

## Anweisungenarten

Gültige Anweisungen sind:

- Wertzuweisungen
- Inkrement / Dekrement Ausdrücke
- Methodenaufrufe
- Objekterzeugungs-Ausdrücke (noch nicht gesehen)

110

## Anweisungenarten

### Beispiele

```
aValue = 8933.234;           // Wertzuweisung
aValue++;                    // Inkrement
System.out.println("Awesome"); // Methodenaufruf
```

111

## Blöcke

Ein Block ist ...

- eine Gruppe von Anweisungen
- überall erlaubt wo Anweisungen erlaubt sind
- durch geschweiften Klammern markiert

```
{
  Anweisung
  Anweisung
  :
}
```

112

## Kontrollstrukturen

Die Abfolge der Ausführung eines Programms wird durch Anweisungen gesteuert.

Im einfachsten Fall:

**Sequenz:** Lineare Abfolge von Anweisungen:

```
Anweisung
Anweisung
```

113

## Kontrollstrukturen - Sequenz

### Beispiel

```
int fahrenheit = 9f * celsius / 5 + 32;  
System.out.println("Fahrenheit: " + fahrenheit);
```

114

## Kontrollstrukturen - If Anweisung

```
if ( Bedingung )  
    Anweisung  
optional || else  
    Anweisung
```

115

## Bedingung: Boolsche Ausdrücke

- Ein Ausdruck der zu `true` oder `false` ausgewertet wird.
- Verwendet boolschen Operatoren in zusammengesetzten Ausdrücken.

### Beispiele

```
scanner.hasNext() // Hat es noch ein Zeichen?  
x > 3 || y > 3    // Ist x oder y groesser 3?
```

116

## Vergleichsoperatoren

Infix Notation:  $x \text{ op } y$  mit folgenden Operatoren

op: < > <= >= == !=

↑  
Nicht zu verwechseln mit Zuweisungsoperator "="

- Angewandt auf zwei vergleichbare Operanden
- Resultat ist vom Typ `boolean`
- <, >, <= und >= binden stärker als == und !=

117



## Vergleichsoperatoren

### Beispiele

```
x >= 5    // ergibt true, falls x groesser oder gleich
          // 5 ist, ansonsten false

a < b == b > a    // Kommutativgesetz fuer Vergleiche
```

118

## Boolsche Binäre Operatoren

Infix Notation:  $x \text{ op } y$  mit folgenden Operatoren

- `&&` : Konjunktion (logisches “und”)
- `||` : Disjunktion (logisches “oder”)

Präzedenz:

- `&&` vor `||`
- Beide binden schwächer als Vergleichsoperatoren

119

## Shortcut Evaluation

Wie wird dieser Ausdruck ausgewertet? ...

```
d > 0 && n/d > 1
```

... im Fall von `d == 0` ?

120

## Shortcut Evaluation

**Annahme:** `d == 0`

**Problem:** Die Auswertung des rechten Operandes würde zu einer Division durch 0 führen.

⇒ *Solche Programmfehler führen zum Abbruch der Auswertung!*

121

## Shortcut Evaluation $\Rightarrow$ Lazy Evaluation

Annahme: `d == 0`

Lösung: Faule Auswertung! Nur Auswerten was wirklich nötig ist.

```
d > 0 && irgend_etwas
```

$\Rightarrow$

```
false && irgend_etwas
```

$\Rightarrow$

```
false
```

**Shortcut-Evaluation:** Berechnung des Ausdrucks abbrechen sobald Ergebnis feststeht.

122

## Boolscher unärer Operator: Negation

Prefix Notation: `! b`

Die Negation binden stärker als alle anderen boolschen Operatoren, inklusive Vergleichsoperatoren

123

## Boolscher unärer Operator: Negation

Beispiel

```
!(d > 0) || n/d == 0
```

124

## Kontrollstrukturen - Verwendung von Blöcken

In Kontrollstrukturen sollten immer Blöcke verwendet werden.<sup>5</sup>

```
if ( Bedingung ) {  
    Anweisung  
} else if ( Bedingung ){  
    Anweisung  
} else {  
    Anweisung  
}
```

<sup>5</sup>Wir weichen von dieser Regel sporadisch aus Gründen des Platzmangels auf übersichtlichen Folien ab.

125

## Kontrollstrukturen - Verwendung von Blöcken

### Beispiele

```
if (anzahl > 0)
    mean = sum / anzahl; ssq = sumsq / anzahl;
```

```
System.out.println(ssq);
```

126

## Kontrollstrukturen - Verwendung von Blöcken

### Beispiele

```
if (anzahl > 0) {
    mean = sum / anzahl;
}
ssq = sumsq / anzahl;
System.out.println(ssq);
```

127

## Kontrollstrukturen: Schleifen

Es gibt mehrere Schleifen-Anweisungen

- `while`
- `do ...while`
- `for`
- und noch weitere Konstrukte (kommen später)

128

## Kontrollstrukturen - While Schleife

```
while ( Bedingung )
    Anweisung
```

129

## Kontrollstrukturen - While Schleife

### Beispiele

```
int num = 143;
int factor = 2;

while (num % factor != 0){
    factor++;
}

System.out.println(factor + "|" + num);
```

130

## Kontrollstrukturen - Do ... While Schleife

```
do
    Anweisung
while ( Bedingung )
```

131

## Kontrollstrukturen - Do ... While Schleife

### Beispiele

```
Scanner scanner = new Scanner(System.in);
String password;
do{
    System.out.print("enter magic word:");
    password = scanner.next();
} while (!password.equals("geheim"));
System.out.println("Door open.");
scanner.close();
```

132

## Kontrollstrukturen - For Schleife

```
for ( Initialisierung ; Bedingung ; Fortschritt )
    Anweisung
```

⇔

```
Initialisierung
while ( Bedingung ) {
    Anweisung
    Fortschritt
}
```

133

## Kontrollstrukturen - Do ... While Schleife

### Beispiele

```
for (int x = 1; x <= 128; x *= 2) {  
    System.out.println("x= " + x);  
}
```

⇔

```
int x = 1;  
while (x <= 128){  
    System.out.println("x= " + x);  
    x *= 2;  
}
```

134

## Ein Beispiel: Monte Carlo Simulation

*Monte Carlo Simulation:* Verwendung des Zufalls zum Lösen von Problemen. Breites Anwendungsfeld in der angewandten Mathematik und sämtlichen Natur- und Ingenieurwissenschaften.

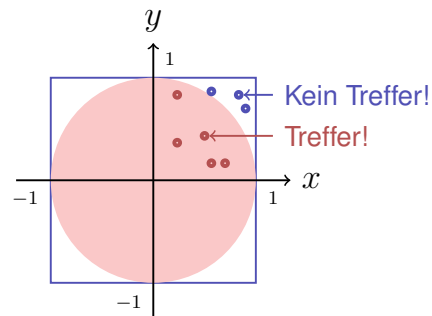
135

## $\pi$ schätzen mit Monte Carlo Simulation

$$\frac{\text{Kreisfläche}}{\text{Fläche des Quadrates}} = \frac{\pi}{4}$$

Idee: Simuliere Zufallsvariable mit Gleichverteilung auf dem Einheitsquadrat  $[0, 1] \times [0, 1]$ .

$$\frac{\text{Anzahl Treffer}}{\text{Anzahl Versuche}} \cdot 4 \approx \pi.$$



136

## Aufgabe

- Experiment: Schätze  $\pi$  mit obigem Monte-Carlo Verfahren
- Führe das Experiment durch für Anzahl Versuche 1, 2, 4, ... , 4096
- Jeweils Ausgabe der Anzahl Versuche und der Schätzung von  $\pi$ .

Hinweis: eine auf  $[0, 1)$  uniformverteilte Pseudo-Zufallszahl bekommt man mit `Math.random()`.

137

## $\pi$ schätzen mit Monte Carlo Simulation

```
public class Pi {
    public static void main(String[] args){
        for (int trials = 1; trials <= 4096; trials*=2){
            int hits = 0;
            for (int i = 0; i<trials; ++i){
                double x = Math.random();
                double y = Math.random();
                if (x * x + y * y <= 1){
                    hits++; }
            }
            double pi = (double)hits / trials * 4;
            System.out.println("trials=" + trials + ", pi=" + pi);
        }
    }
}
```