Schleifeninvarianten, Zahlendarstellung und Typanpassungen

CoMa-Übung V

TU Berlin

14.11.2012

Themen der Übung

1. Organisatorisches
2. Schleifeninvarianten
3. Widerspruchsbeweise
4. Zahlendarstellung in Rechnern
5. Typanpassungen
6. Operatoren
7. Variablen-Gültigkeit

- Neues Tutorium: Do, 10 – 12, MA 645 bei Daniel Kuske
- Zu wenig Platz in eurem Tutorium? Do, 10 – 12 passt euch besser? Geht hin!

Schleifeninvarianten

Eine **Schleifeninvariante** ist eine Eigenschaft einer Schleife in einem Algorithmus, die zu einem *bestimmten Zeitpunkt* in jedem Schleifendurchlauf gültig ist.

Nutzen von Schleifeninvarianten

- Korrektheit von Schleifen / Algorithmen beweisen

Korrektheit von Algorithmen

- Ein Algorithmus ist **partiell korrekt**, wenn er bei Terminierung das korrekte Ergebnis liefert.
- Vorsicht: Jeder nie terminierende Algorithmus ist partiell korrekt!
- Ein Algorithmus ist **total korrekt**, wenn er partiell korrekt ist und immer terminiert.

Schleifeninvarianten (2)

Beispiel: Maximale Komponente eines Vektors

```
maximum(x):
Input:  Vektor x ∈ ℝ^n
Output:  max{x_i | i = 1,...,n}
max := −∞
FOR i := 1 TO n DO
     IF x_i > max THEN
          max := x_i
     ENDIF
ENDFOR
RETURN max
```

Korrektheitsbeweis

- Zu zeigen: Variable `max` hat am Ende den Wert $\max\{x_i \mid i = 1, \ldots, n\}$
- Idee: Betrachte den Wert von `max` nach jedem Durchlauf

Schleifeninvarianten (3)

Beispiel: Maximale Komponente eines Vektors

```
max := −∞
FOR i := 1 TO n DO
      IF x_i > max THEN
           max := x_i
      ENDIF
ENDFOR
RETURN max
```

Invarianten

Definiere `max(i)` := Wert von `max` **nach** i-tem Schleifendurchlauf.

- Invariante: $\max(i) = \{x_1, \ldots, x_i\}$, $i = 1, \ldots, n$

Nach Terminierung der Schleife gilt $i = n$, d.h:

$$\texttt{max} = \texttt{max(n)} = \max\{x_1, ..., x_n\}.$$

Schleifeninvarianten (4)

Beispiel: Maximale Komponente eines Vektors

```
max := −∞
FOR i := 1 TO n DO
      IF x_i > max THEN
          max := x_i
      ENDIF
ENDFOR
RETURN max
```

Beweis der Invarianten $\max(i) = \{x_1, \ldots, x_i\}$, $i = 1, \ldots, n$

Beweis durch vollständige Induktion über die Anzahl der Iterationen i:

- *Induktionsanfang*, $i = 1$. Wegen $x_i \in \mathbb{R} > -\infty$ ist nach der ersten Iteration $\max(1) = x_1 = \max\{x_1\}$.
- *Induktionsvoraussetzung*, die Behauptung gelte für $i = 1, \ldots, k$ und ist zu zeigen für $i = k + 1$.
- *Induktionsschluss*.

Schleifeninvarianten (5)

Beispiel: Maximale Komponente eines Vektors

```
FOR i := 1 TO n DO
      IF x_i > max THEN max := x_i ENDIF
ENDFOR
```

Beweis der Invarianten $\max(i) = \{x_1, \ldots, x_i\}$, $i = 1, \ldots, n$

Induktionsvoraussetzung, die Behauptung gelte für $i = 1, \ldots, k$ und ist zu zeigen für $i = k + 1$.

Induktionsschluss. Nach Definition des Algorithmus gilt:

$$\max(k+1) = \begin{cases} \max(k) = \max\{x_1, \ldots, x_k\} & \text{falls } \max(k) \geq x_{k+1} \\ x_{k+1} & \text{falls } \max(k) < x_{k+1} \end{cases}$$

Damit folgt $\max(k+1) = \max\{\max(k), x_{k+1}\}$ und nach IV ist $\max\{\max(k), x_{k+1}\} = \max\{\max\{x_1, \ldots, x_k\}, x_{k+1}\} = \max\{x_1, \ldots, x_{k+1}\}$.

Schleifeninvarianten (6)

Beweis der Invarianten $\max(i) = \{x_1, \ldots, x_i\}$, $i = 1, \ldots, n$

Beweis durch vollständige Induktion über die Anzahl der Iterationen i:

- *Induktionsanfang*, $i = 1$. Wegen $x_i \in \mathbb{R} > -\infty$ ist nach der ersten Iteration $\max(1) = x_1 = \max\{x_1\}$.
- *Induktionsvoraussetzung*, die Behauptung gelte für $i = 1, \ldots, k$ und ist zu zeigen für $i = k + 1$.
- *Induktionsschluss*. Nach Definition des Algorithmus gilt:

$$\max(k+1) = \begin{cases} \max(k) = \max\{x_1, \ldots, x_k\} & \text{falls } \max(k) \geq x_{k+1} \\ x_{k+1} & \text{falls } \max(k) < x_{k+1} \end{cases}$$

Damit folgt:
$$\begin{aligned} \max(k+1) &= \max\{\max(k), x_{k+1}\} \\ &= \max\{\max\{x_1, \ldots, x_k\}, x_{k+1}\} \\ &= \max\{x_1, \ldots, x_{k+1}\} \end{aligned}$$

Es gilt damit: $\max = \max(n) = \max\{x_1, \ldots, x_n\}$. □

Widerspruchsbeweise

Ein **Widerspruchsbeweis** (auch *indirekter Beweis* oder *reductio ad absurdum* genannt) folgt einer Beweistechnik, die nach folgendem Schema vorgeht:

- Nimm das logische Gegenteil dessen an, was bewiesen werden soll.
- Leite aus dieser Annahme einen Widerspruch her.
- Ist die Annahme wahr, wäre also auch der Widerspruch wahr.
- Da ein Widerspruch niemals wahr sein kann, muss die Annahme falsch sein.
- Somit ist das logische Gegenteil der Annahme wahr.

Widerspruchsbeweise

Ein **Widerspruchsbeweis** (auch *indirekter Beweis* oder *reductio ad absurdum* genannt) folgt einer Beweistechnik, die nach folgendem Schema vorgeht:

- Nimm das logische Gegenteil dessen an, was bewiesen werden soll.
- Leite aus dieser Annahme einen Widerspruch her.
- Ist die Annahme wahr, wäre also auch der Widerspruch wahr.
- Da ein Widerspruch niemals wahr sein kann, muss die Annahme falsch sein.
- Somit ist das logische Gegenteil der Annahme wahr.

Beispiel

Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.

Widerspruchsbeweise

Beispiel

Behauptung: Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.

Widerspruchsbeweise

Beispiel

Behauptung: Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.
- n ungerade $\Rightarrow \exists k \in \mathbb{N} : n = 2k + 1$

Widerspruchsbeweise

Beispiel

Behauptung: Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.
- n ungerade $\Rightarrow \exists k \in \mathbb{N} : n = 2k + 1$
- $\Rightarrow n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$

Widerspruchsbeweise

Beispiel

Behauptung: Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.
- n ungerade $\Rightarrow \exists k \in \mathbb{N} : n = 2k + 1$
- $\Rightarrow n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$
- n^2 ist ungerade $\rightarrow$ Widerspruch zu n^2 gerade! $\square$

Widerspruchsbeweise

Beispiel

Behauptung: Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.

- Annahme für den Widerspruchsbeweis: Sei n^2 für ein $n \in \mathbb{N}$ gerade und n ungerade.
- n ungerade $\Rightarrow \exists k \in \mathbb{N} : n = 2k + 1$
- $\Rightarrow n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$
- n^2 ist ungerade $\rightarrow$ Widerspruch zu n^2 gerade! $\square$

Zusammenfassung

- Widerspruchsannahme treffen.
- Mit Widerspruchsannahme und den Voraussetzungen der Behauptung schlussfolgern.
- Widerspruch herleiten.
- Fertig. $\square$

Widerspruchsbeweise

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$
- $2m^2 = n^2 \Rightarrow n^2$ ist gerade

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$
- $2m^2 = n^2 \Rightarrow n^2$ ist gerade
- n^2 ist gerade $\Rightarrow n$ ist gerade $\Rightarrow \frac{n}{2} \in \mathbb{N}$

  (Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.)

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$
- $2m^2 = n^2 \Rightarrow n^2$ ist gerade
- n^2 ist gerade $\Rightarrow n$ ist gerade $\Rightarrow \frac{n}{2} \in \mathbb{N}$

  (Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.)
- $2 = \frac{n^2}{m^2} \Rightarrow m^2 = \frac{n^2}{2} = 2 \cdot \left(\frac{n}{2}\right)^2$

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$
- $2m^2 = n^2 \Rightarrow n^2$ ist gerade
- n^2 ist gerade $\Rightarrow n$ ist gerade $\Rightarrow \frac{n}{2} \in \mathbb{N}$

  (Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.)
- $2 = \frac{n^2}{m^2} \Rightarrow m^2 = \frac{n^2}{2} = 2 \cdot \left(\frac{n}{2}\right)^2$
- $\Rightarrow m^2$ ist gerade $\Rightarrow m$ ist gerade

Widerspruchsbeweise

Beispiel

Behauptung: $\sqrt{2}$ ist irrational.

- Annahme für den Widerspruchsbeweis: $\sqrt{2}$ ist rational.
- $\sqrt{2}$ ist rational $\Rightarrow \exists n, m \in \mathbb{N} : \sqrt{2} = \frac{n}{m}, ggT(n, m) = 1$

  (Jede positive rationale Zahl lässt sich als Quotient zweier teilerfremden, natürlichen Zahlen schreiben.)
- $\sqrt{2} = \frac{n}{m} \Rightarrow 2 = \frac{n^2}{m^2} \Rightarrow 2m^2 = n^2$
- $2m^2 = n^2 \Rightarrow n^2$ ist gerade
- n^2 ist gerade $\Rightarrow n$ ist gerade $\Rightarrow \frac{n}{2} \in \mathbb{N}$

  (Ist n^2 für ein $n \in \mathbb{N}$ gerade, so ist n gerade.)
- $2 = \frac{n^2}{m^2} \Rightarrow m^2 = \frac{n^2}{2} = 2 \cdot \left(\frac{n}{2}\right)^2$
- $\Rightarrow m^2$ ist gerade $\Rightarrow m$ ist gerade
- $\Rightarrow ggT(m, n) \geq 2 \rightarrow$ Widerspruch zu $ggT(n, m) = 1$! □

Zahlensysteme

Positionelle Zahlensysteme

Zahlensysteme werden zur Darstellung von Zahlen verwendet. Wir verwenden *positionelle* Systeme, in denen die Wertigkeit einer Ziffer von ihrer Position abhängt.

- $21 = 2 \cdot 10^1 + 1$

Hinweis:

Nicht alle Zahlensysteme sind positionell.

- Das römische Zahlensystem ist z.B. *additiv* – alle Ziffern werden addiert (sofern keine Subtraktionsregel benutzt wird).
- `XXI = 10 + 10 + 1 = 21`
- *Hybride* Systeme sind in Asien weit verbreitet:
- 二十一$= 2 \cdot 10 + 1 = 21$

Zahlensysteme (2)

Basen

In positionellen Zahlensystemen kommen die Ziffern aus einer endlichen Menge $\{0, 1, \ldots, b-1\}$ für ein $b \in \mathbb{N}$, $b > 2$. b heißt **Basis** des Systems.

- Eine Zahl $a_n a_{n-1} \ldots a_0$ zur Basis b steht dann für

$$\sum_{i=0}^{n} a_i \cdot b^i = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \cdots + a_0 \cdot b^0$$

Gebräuchliche Basen

- Im Alltag gebräuchlich ist das *Dezimalsystem* mit der Basis 10.

$$21_{10} = 2 \cdot 10^1 + 1 \cdot 10^0$$

- Computer benutzen das *Binärsystem* mit der Basis 2.

$$10101_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 21_{10}$$

Zahlensysteme (3)

Basen und Ziffern

In einem Zahlensystem mit Basis b gibt es die Ziffern $0, 1, \ldots, b-1$.

- Im Dezimalsystem (Basis 10) gibt es die Ziffern $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.
- Im Oktalsystem (Basis 8) gibt es die Ziffern $0, 1, 2, 3, 4, 5, 6, 7$.
- Im Biärsystem (Basis 2) gibt es die Ziffern $0, 1$.

Zählen in anderen Systemen

Dezimal	Oktal	Binär
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101

Dezimal	Oktal	Binär
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011

Darstellen von Zahlen in Rechnern

Nicht-negative Ganzzahlen

Nicht-negative Ganzzahlen werden von Rechnern als Binärzahlen mit einer festen Anzahl Stellen gespeichert. Beispiele (mit 8 festen Stellen):

- $5_{10} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 00000101_2$
- $8_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 00001000_2$
- $63_{10} = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 00111111_2$

Mit 8 Stellen ist $255 = \sum_{i=0}^{7} 2^i = 2^8 - 1$ die größte darstellbare Zahl.

Darstellen von Zahlen in Rechnern

Nicht-negative Ganzzahlen

Nicht-negative Ganzzahlen werden von Rechnern als Binärzahlen mit einer festen Anzahl Stellen gespeichert. Beispiele (mit 8 festen Stellen):

- $5_{10} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 00000101_2$
- $8_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 00001000_2$
- $63_{10} = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 00111111_2$

Mit 8 Stellen ist $255 = \sum_{i=0}^{7} 2^i = 2^8 - 1$ die größte darstellbare Zahl.

Negative Ganzzahlen

Negative von positiven Zahlen unterscheiden braucht ein Vorzeichen.

- Rechner haben nur 0 und 1 $\rightarrow$ erste Ziffer dient als Vorzeichen,
- 0 für +, 1 für -.
- Negative Zahlen werden im *Zweierkomplement* gespeichert,
- was dem Rechner arithmetische Operationen erleichtert.

Darstellen von Zahlen in Rechnern (2)

Überläufe

Durch die feste Anzahl Stellen können bei Rechnungen zu Zahlen entstehen, die außerhalb des Wertebereichs liegen.

- $16_{10} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 00010000_2$
- $16_{10} \cdot 16_{10} = 00010000_2 \cdot 00010000_2 = 100000000_2 = 256_{10}$
- $\rightarrow$ wird zu $00000000_2 = 0_{10}$ abgeschnitten.

Dieses Ereignis wird **Überlauf** genannt.

- Bei Überläufen das Vorzeichen überschrieben,
- und alle Stellen außerhalb des Wertebereichs abgeschnitten.

Vorsicht

- Java gibt bei Überläufen keinen Fehler aus!
- Aufgabe des Menschen, ausreichend große Datentypen zu verwenden.

Darstellen von Zahlen in Rechnern (3)

Gleitkommazahlen

Gleitkommazahlen haben einen sehr großen Wertebereich (ca. $\pm 9 \cdot 10^{308}$ für `double`). Erreicht wird das durch eine Exponentialdarstellung der Form

$$\text{Vorzeichen} \cdot \text{Mantisse} \cdot 2^{\text{Exponent}}$$

- Der **Exponent** ist eine Ganzzahl mit einer festen Anzahl Stellen.
- Die **Mantisse** ist eine Zahl $\in [1, 2)$ mit einer festen Anzahl Nachkommastellen.

Exponent und Mantisse werden als Binärzahlen gespeichert.

Datentyp	Vorzeichen	Exponent	Mantisse
`float`	ja	8 Stellen	23 Stellen
`double`	ja	11 Stellen	52 Stellen

Double

Besonderheiten von double

`Double` besitzt die Möglichkeit,

- $+\infty$ darzustellen (`Double.POSITIVE_INFINITY`),
- $-\infty$ darzustellen (`Double.NEGATIVE_INFINITY`),
- $+0.0$ und -0.0 (es gilt $+0.0 == -0.0$),
- sowie nicht definierte Ergebnisse (`Double.NaN` – *Not a Number*).

Unendlich

`Double.POSITIVE_INFINITY` / `Double.NEGATIVE_INFINITY` treten auf,

- wenn bei einer Berechnung ein Ergebnis auftritt, was außerhalb des Wertebereichs von `Double` liegt,
- bei Division durch $+0.0$ bzw. -0.0.

Vergleiche und arithmetische Operationen funktionieren mit diesen Konstanten.

Double (2)

Not a Number

`Double.NaN` stellt das Ergebnis einer nicht-definierten Berechnung dar.
Dazu gehören:

- `0.0/0.0`,
- `Math.sqrt(-1)`,
- `Double.POSITIVE_INFINITY * 0`,
- `Double.POSITIVE_INFINITY + Double.NEGATIVE_INFINITY`.

Vergleiche mit `Double.NaN`

- `Double.NaN == d` ist für jeden `double d` falsch,
- insbesondere ist `Double.NaN == Double.NaN` auch `false`.
- `Double.NaN != d` ist für jeden `double d` wahr,
- auch `Double.NaN != Double.NaN` ist `true`.
- <, >, <=, >= verhalten sich wie ==.

Literale

Der Standardfall

Normalerweise intepretiert Java Zahlen als `int` oder `double`.

- `int`: Jede Ziffernfolge ohne Dezimalpunkt (`.`) und ohne Exponentialschreibweise (`e`/`E`): `12`, `-1`
- `double`: Jede Ziffernfolge mit Dezimalpunkt (`.`) oder mit Exponentialschreibweise (`e`/`E`): `1.0`, `10e2`

Explizite Typangabe

Man kann Java zwingen, ein Literal (oder Variable) als bestimmten Typ zu interpretieren. Dazu schreibt man `(typ)` vor das Literal / die Variable.

- `(byte) 1` ist vom Typ `byte`,
- `(short) 1` ist vom Typ `short`,
- `(int) 1` ist vom Typ `int`,
- ...

Literale (2)

Affixe

Um Zahlen als `long`, `float` oder `double` zu deklarieren, gibt es die Möglichkeit, einen Buchstaben als Affix anzuhängen.

- `l` oder `L` für `long`: `10l`, `67L`
- `f` oder `F` für `float`: `10f`, `67F`
- `d` oder `D` für `double`: `10d`, `67D`

Präfixe für andere Zahlensysteme

Java erlaubt es, Zahlen als Binär- (Basis 2), Octal- (Basis 8) oder Hexadezimalzahl (Basis 16) einzugeben. Dies wird durch Präfixe realisiert:

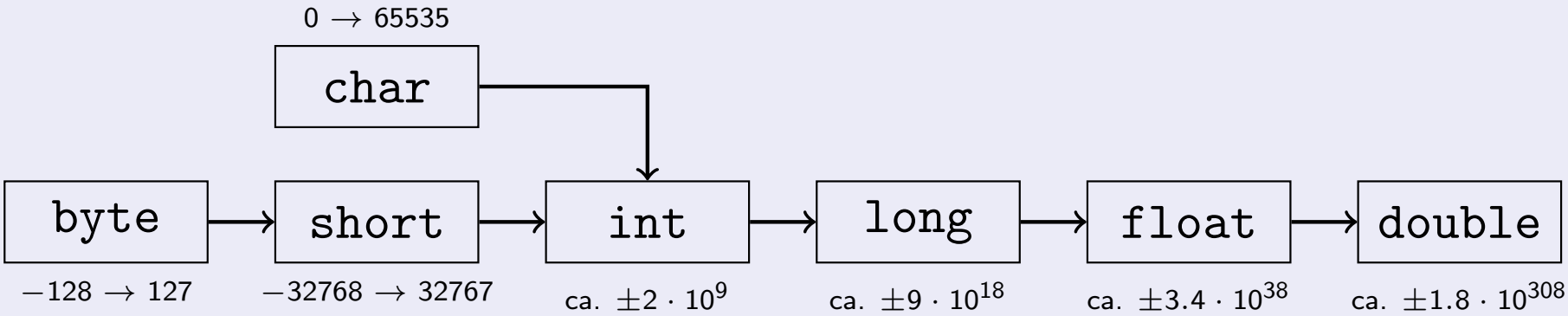
- `0b` oder `0B` für das Binärsystem.
- `0` für das Oktalsystem – `010` steht also für 8, nicht 10!
- `0x` oder `0X` für das Hexadezimalsystem.

Typanpassung

Java bietet sieben primitive Datentypen für Zahlen. Welchen Typ hat das Ergebnis, wenn man mit Zahlen mit unterschiedlichen Typen rechnet?

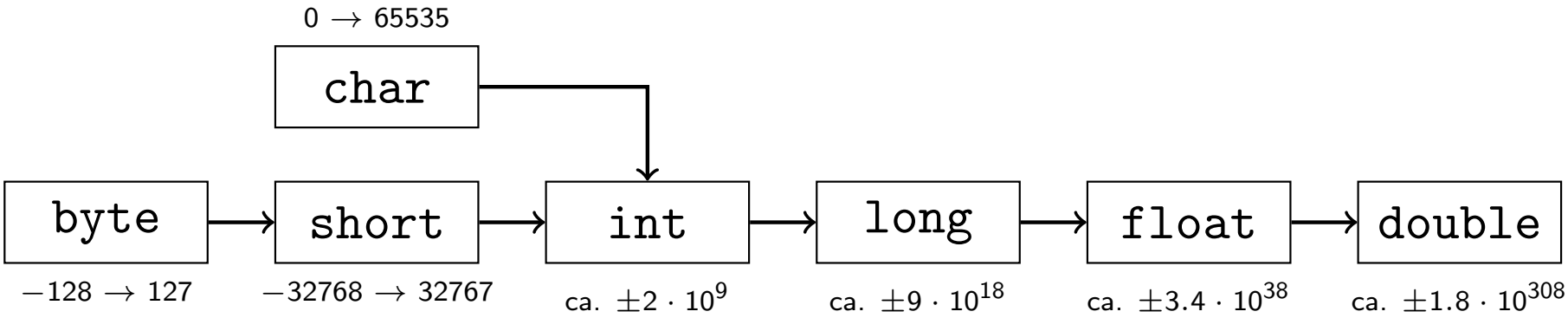
Typanpassung bei Operationen

- Operanden der Typen `byte` und `short` werden automatisch in `int` umgewandelt.
- Das Ergebnis einer Operation hat dann denselben Typ wie der größte Operand (größte im Bezug auf seinen Wertebereich).



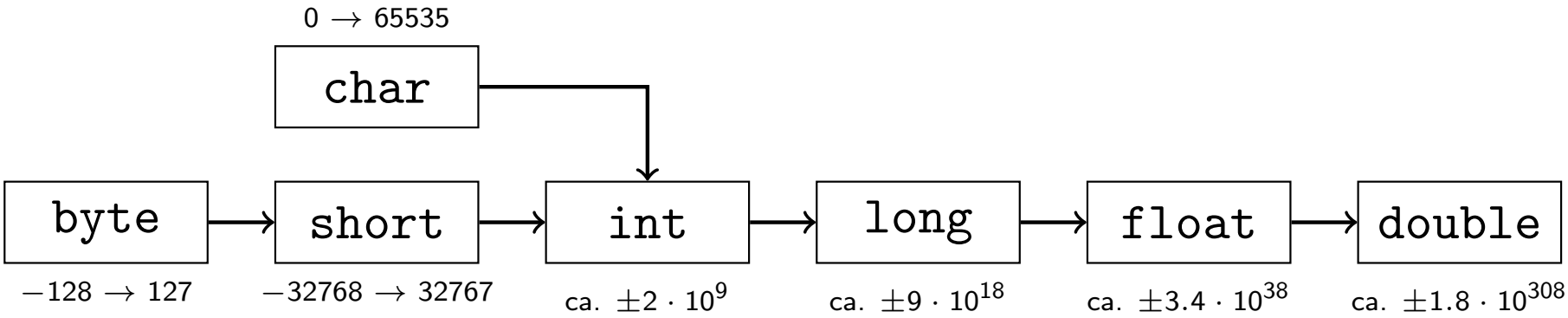
- Für einen `int i` und einen `double d` ist das Ergebnis `i + d` also vom Typ `double`.

Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

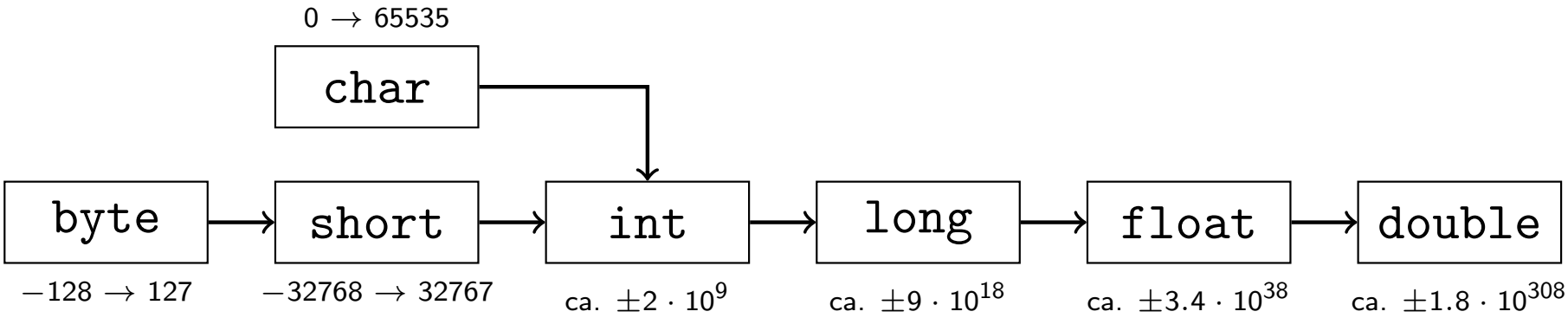
Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!

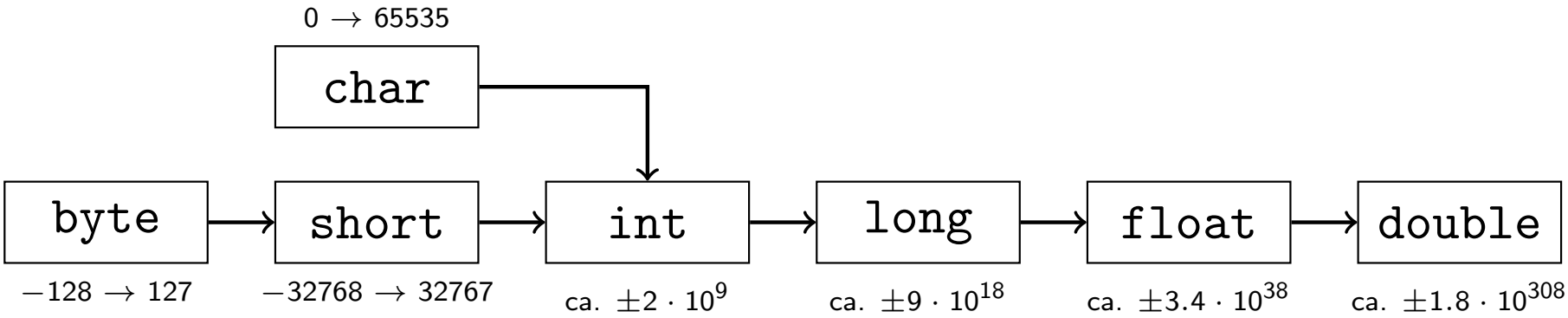
Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!
- `i + d` wird zu `double` > `int`: Fehler!

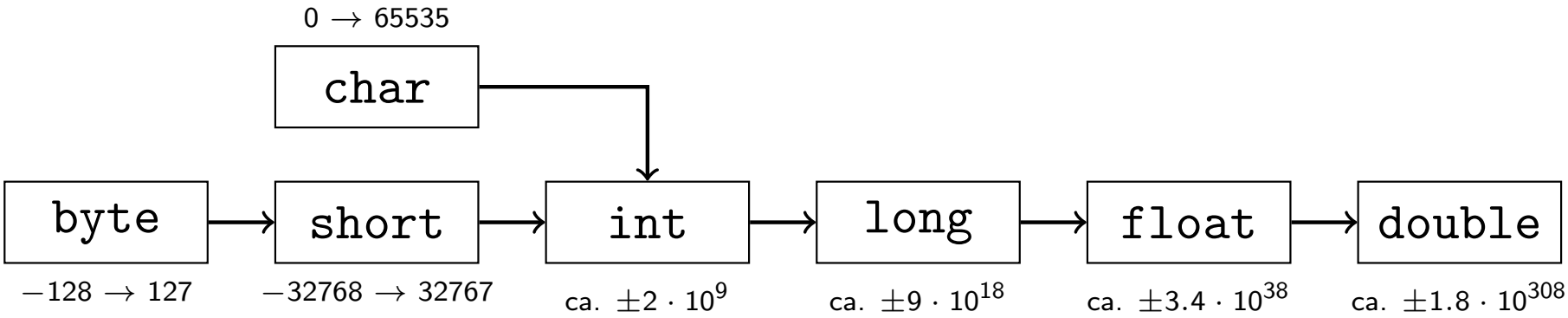
Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!
- `i + d` wird zu `double` > `int`: Fehler!
- 3L ist ein `long` > `int`: Fehler!

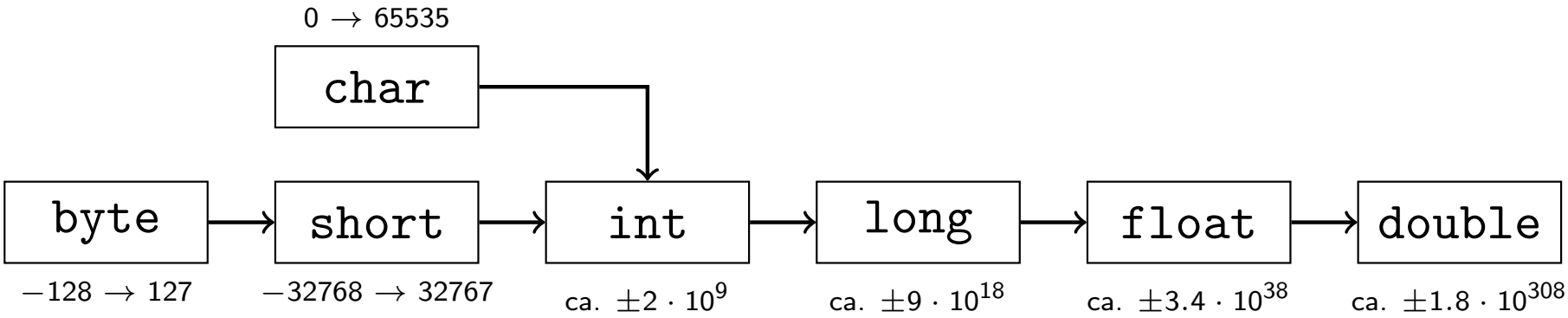
Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!
- `i + d` wird zu `double` > `int`: Fehler!
- 3L ist ein `long` > `int`: Fehler!
- `ell + f` wird zu `float` > `long`: Fehler!

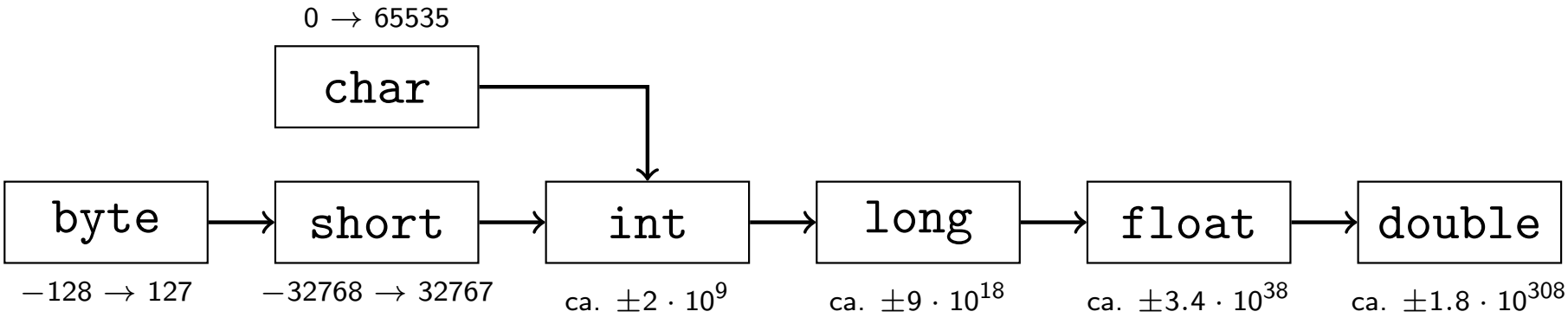
Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!
- `i + d` wird zu `double` > `int`: Fehler!
- 3L ist ein `long` > `int`: Fehler!
- `ell + f` wird zu `float` > `long`: Fehler!
- `ell + f` wird zu `float` < `double`: okay

Typanpassung – Beispiele




```
1  byte b = (byte) 0;
2  short s = (short) 1;
3  int i = 2;
4  long ell = 3L;
5  float f = 4f;
6  double d = 5.0;
7  s = s + b;
8  i = i + d;
9  i = 3L;
10 ell = ell + f;
11 d = ell + f;
12 f = 1.0;
```

- `s + b` wird zu `int` > `short`: Fehler!
- `i + d` wird zu `double` > `int`: Fehler!
- 3L ist ein `long` > `int`: Fehler!
- `ell + f` wird zu `float` > `long`: Fehler!
- `ell + f` wird zu `float` < `double`: okay
- `1.0` ist ein `double` > `float`: Fehler!
- Daher: Aufpassen, dass der Typ der linken Seite wirklich zur rechten Seite passt.

Typanpassung – Automatisch und Explizit

In Java gibt es zwei Formen der Typanpasssung (**typecast/cast**): *Automatische (implizite) Typanpassung* und *explizite Typanpassung*.

Automatische Typanpassung

- Wird vom Kompiler bei arithmetischen Operationen mit unterschiedlichen Datentypen durchgeführt.
- Kein menschliches Eingreifen nötig.
- Normalerweise verlustfrei (aber Gleitkommazahlen können ungenauer als Ganzzahlen sein).

Achtung:

- Beim Rechnen werden `byte` und `short` automatisch zu `int`.
- Es kann nichts automatisch in `char` umgewandelt werden, da `char` keine negativen Zahlen unterstützt.

Typanpassung – Automatisch und Explizit (2)

Soll aus einem Datentyp mit größerem Wertebereich einer mit kleinerem Wertebereich werden, ist ein explizite Typanpassung erforderlich.

Explizit Typanpassung

- Muss explizit vom Menschen in den Code geschrieben werden.
- Die Syntax dafür ist `(typ)` vor den Ausdruck zu schreiben, der umgewandelt werden soll: `int i = (int) 5.4;`
- Wird z.B. für die Zuweisung von größeren auf kleinere Datentypen benutzt.
- Gefahr von Verlust bei der Typanpassung.

Der `(typ)`-Operator

- wird von Java als Operator mit einem Argument betrachtet.
- hat eine höhrere Priorität als alle arithmetischen Operationen!
- `(int) 1 + 1.0` ist somit ein `double`.

Explizite Typanpassung

Größere Ganzzahl → Kleinere Ganzzahl

- Das Vorzeichen und die höchsten Bits werden einfach abgeschnitten.

Gleitkommazahl → Ganzzahl

- Die Gleitkommazahl wird zu 0 hin gerundet.
- `56.8` wird zu `56`, `-42.9` zu `-42`, etc.
- Ist die Gleitkommazahl außerhalb des Wertebereichs der Ganzzahl, ist das Ergenis die größte (bzw. kleinste) Ganzzahl des Typs.
- `(long) -1e40` ist also -2^{63}, die kleinste Zahl, die `long` darstellen kann.

Hinweis

Es ist nicht möglich aus nicht-Zahl-Typen, wie etwa `String`, durch automatische oder explizite Typanpassung Zahlen zu machen. Dafür müssen gesonderte Methoden zum Einsatz kommen.

Explizite Typanpassung (2)

Ganzzahl $\rightarrow$ Gleitkommazahl

- Größerer Wertebereich, Kommazahlen statt Ganzzahlen... alles okay?

Explizite Typanpassung (2)

Ganzzahl $\rightarrow$ Gleitkommazahl

- Größerer Wertebereich, Kommazahlen statt Ganzzahlen... alles okay?
- `int` und `float` haben beide 32 Bit Speicher zur Verfügung
- `long` und `double` haben beide 64 Bit Speicher zur Verfügung

Explizite Typanpassung (2)

Ganzzahl → Gleitkommazahl

- Größerer Wertebereich, Kommazahlen statt Ganzzahlen... alles okay?
- `int` und `float` haben beide 32 Bit Speicher zur Verfügung
- `long` und `double` haben beide 64 Bit Speicher zur Verfügung
- `int` und `long` stellen 2^{32} bzw. 2^{64} verschiedene Zahlen da
- `float` und `double` können wegen ihres begrenzten Speichers nicht mehr als 2^{32} bzw. 2^{64} verschiedene Zahlen darstellen

Explizite Typanpassung (2)

Ganzzahl $\rightarrow$ Gleitkommazahl

- Größerer Wertebereich, Kommazahlen statt Ganzzahlen... alles okay?
- `int` und `float` haben beide 32 Bit Speicher zur Verfügung
- `long` und `double` haben beide 64 Bit Speicher zur Verfügung
- `int` und `long` stellen 2^{32} bzw. 2^{64} verschiedene Zahlen da
- `float` und `double` können wegen ihres begrenzten Speichers nicht mehr als 2^{32} bzw. 2^{64} verschiedene Zahlen darstellen
- $\Rightarrow$ es muss Ganzzahlen geben, die nicht verlustfrei von `int` $\rightarrow$ `float` bzw. `long` $\rightarrow$ `double` übertragen werden können

Explizite Typanpassung (2)

Ganzzahl → Gleitkommazahl

- Größerer Wertebereich, Kommazahlen statt Ganzzahlen... alles okay?
- `int` und `float` haben beide 32 Bit Speicher zur Verfügung
- `long` und `double` haben beide 64 Bit Speicher zur Verfügung
- `int` und `long` stellen 2^{32} bzw. 2^{64} verschiedene Zahlen da
- `float` und `double` können wegen ihres begrenzten Speichers nicht mehr als 2^{32} bzw. 2^{64} verschiedene Zahlen darstellen
- $\Rightarrow$ es muss Ganzzahlen geben, die nicht verlustfrei von `int` → `float` bzw. `long` → `double` übertragen werden können

Hinweis

- Nur bei sehr großen Zahlen problematisch
- `java.math.BigInteger` und `java.lang.BigDecimal` zum Rechnen mit höherer Genauigkeit

Operatoren

Inkrement und Dekrement

Java bietet zwei Operatoren, um Zahlen um eins zu vergrößern / -kleinern:

- `b = ++a * 2;` steht für `a = a + 1; b = a * 2;`
- `b = a++ * 2;` steht für `b = a * 2; a = a + 1;`
- `b = --a * 2;` steht für `a = a - 1; b = a * 2;`
- `b = a-- * 2;` steht für `b = a * 2; a = a - 1;`

Zuweisungen mit Operationen

- `a += b;` ist eine Kurzformen für `a = (Typ von a) (a + b);`
- Damit ist folgendes völlig okay:

```
1 int i = 2;
2 i += 0.5;
```

- Analog dazu funktionieren `-=`, `*=` und `/=`.

Operatorenvorrang

Operatoren	Beschreibung
`++,--,+,-,!,(Typ)`	Inkrement, Dekrement, Unäres Plus, unäres Minus, logisches Nicht, Typanpassung
`*,/,%`	Multiplikation, Division, Rest
`+,-`	Addition, Subtraktion, Konkatenation von Strings
`<,>,<=,>=`	Numerische Vergleiche
`==,!=`	Gleichheit
`&`	Logisches Und
`^`	Logisches Xor
`|`	Logisches Oder
`&&`	Logisches konditionales Und
`||`	Logisches konditionales Oder
`?:`	Bedingungsoperator
`=`	Zuweisung
`+=,-=,*=,/=`	Zuweisung mit Operation

Gültigkeit von Variablen

Arten der Variablen-Deklaration

Variablen können in Java an drei Stellen deklariert werden:

- Als *Attribut* einer Klasse,
- als *Parameter* einer Methode,
- oder als *lokale Variable* in einer Methode.

```
1  public class Scope {
2
3    private String name; // Attribut
4
5    public void test(int a) { // Methoden-Parameter
6      int b = 0; // lokale Variable
7      for (int i = 0; i < 10; i++) { // lokale Variable
8        int c = 0; // lokale Variable
9      }
10   }
11 }
```

Gültigkeit von Variablen (2)

Gültigkeit

- *Attribute* in der ganzen Klasse gültig.
- *Parameter* einer Methode sind in der ganzen Methode gültig (und nirgendwo sonst in der Klasse).
- *Lokale Variablen* sind in dem Block ({..}) gültig, in dem sie deklariert wurden (und nirgendwo außerhalb).

```
1  public class Scope {
2    public void test(int a) {
3      int b = 0;
4      for (int i = 0; i < 10; i++) {
5        int c = 0;
6        System.out.println(b + c); // okay
7      }
8      System.out.println(c); // Fehler
9    }
10 }
```

Gültigkeit von Variablen (3)

Eindeutigkeit von Variablennamen

- An keiner Stelle des Codes dürfen zwei Variablen mit gleichem Namen gültig sein.
- Einzige Ausnahme: *Parameter* / *lokale Variablen* dürfen denselben Namen wie ein *Attribut* haben.
- Java geht dann zunächst davon aus, dass der Parameter bzw. die lokale Variable gemeint ist.
- $\rightarrow$ Fehlergefahr, nach Möglichkeit vermeiden.

```
1 public class Scope {
2   String a;
3   public void test(int a, int b) { // okay
4     int b = 0; // Fehler
5   }
6 }
```