3 Gemeinsamkeiten und Unterschiede zwischen Java und C

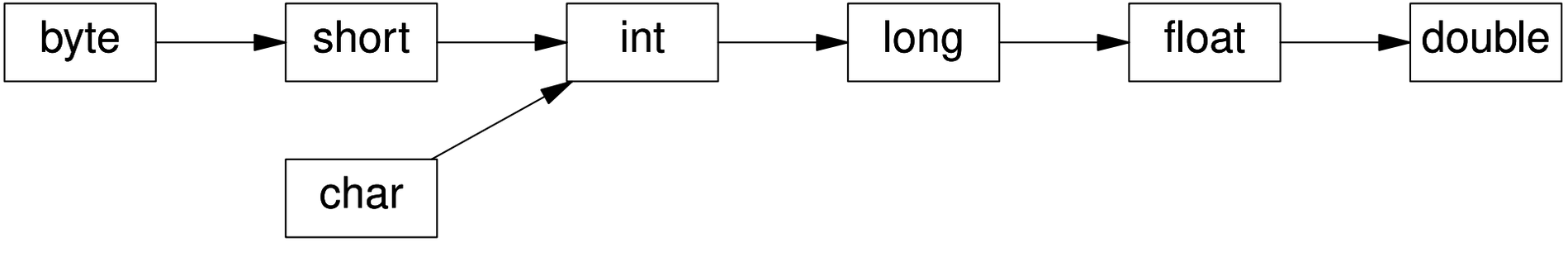
3.1 Äußerlichkeiten

- Kommentare in Java erstrecken sich entweder (wie in C) von `/*` bis `*/` („Blockkommentare“) oder von `//` bis zum Zeilenende („Zeilenkommentare“).
- Blockkommentare können (wie in C) nicht verschachtelt werden, d. h. ein Blockkommentar endet immer beim ersten Auftreten von `*/`.
- Wenn man „echte“ Kommentare als Zeilenkommentare formuliert, kann man Blockkommentare zum „Auskommentieren“ von Programmabschnitten verwenden.
- In Java gibt es keinen Präprozessor, d. h. insbesondere kein `#include` und kein `#define`.
- In C besitzt die Hauptfunktion `main` Resultattyp `int` (der Resultatwert stellt den Exitstatus des Programms dar, der an das Betriebssystem zurückgegeben wird), in Java ist der Resultattyp `void` (der Exitstatus ist standardmäßig 0).
  Auch die Parameterliste von `main` ist unterschiedlich (vgl. § 4.6.3).

3.2 Datentypen

3.2.1 Elementare Typen

Typ	*Java*	*C*
boolean	vorhanden	nicht vorhanden
char	16 Bit	8 Bit
byte	8 Bit	entspricht `signed char`
short	16 Bit	meist 16 Bit
int	32 Bit	meist 32 oder 16 Bit
long	64 Bit	meist 64 oder 32 Bit
unsigned short	vgl. `char`	Größe wie `short`
unsigned int	nicht vorhanden	Größe wie `int`
unsigned long	nicht vorhanden	Größe wie `long`
float	32 Bit	meist 32 Bit
double	64 Bit	meist 64 Bit



- Umwandlungen in Richtung der Pfeile in der vorigen Abbildung erfolgen bei Bedarf implizit, während andere Umwandlungen explizit erfolgen müssen.

- Ganzzahlige Arithmetik wird grundsätzlich mit `int`- oder `long`-Werten ausgeführt; „kürzere“ Werte werden ggf. automatisch expandiert.

- Wie in C, können `char`-Werte wie z. B. `'A'` oder `'9'` wie ganze Zahlen in arithmetischen Ausdrücken und Vergleichen verwendet werden; ihr Wert entspricht der Position des jeweiligen Zeichens im Unicode-Zeichensatz, ohne dass man diese Position genau kennen muss.

- Beispiel: Umwandlung von Klein- in Großbuchstaben

```
char c = 'x'; // ... oder irgendein anderer Kleinbuchstabe.

// Fehler: Rechte Seite hat Typ int => Zuweisung an char verboten.
c = c + 'A' - 'a';
// Korrekt: Rechte Seite explizit in char umwandeln.
c = (char)(c + 'A' - 'a');
// Auch korrekt: Bei += etc. wird automatisch umgewandelt!
c += 'A' - 'a';

// Beachte: 'A' - 'a' ist der Abstand zwischen einem beliebigen
// Großbuchstaben und dem zugehörigen Kleinbuchstaben.
```

3.2.2 Zeiger und Referenzen

- In Java gibt es keine expliziten Zeigertypen (wie z. B. `int*` oder `int (*) ()`) und somit auch keinen Adress- (`&`) und Inhaltsoperator (`*`).
- Stattdessen sind alle nicht-elementaren Typen (d. h. Klassen, Schnittstellen und Reihen) *Referenztypen*, d. h. implizit Zeigertypen.
- Die *Nullreferenz* `null`, die mit allen Referenztypen kompatibel ist, repräsentiert eine Referenz auf „nichts“.
- Anstelle von Funktionszeigern kann man Referenzen auf Objekte verwenden, die die gewünschte „Funktion“ (d. h. Methode) besitzen (vgl. § 7.1).

3.2.3 Reihen (arrays)

Eindimensionale Reihen

- Reihentypen in Java legen nur den Typ der Elemente fest, nicht jedoch ihre Anzahl, z. B.:

```
double [] a;    // (Referenz auf eine) Reihe von double-Werten.
```

- Reihenobjekte müssen zur Laufzeit explizit mit `new` erzeugt werden, z. B.:

```
a = new double [10];    // Reihe mit 10 double-Werten erzeugen.
```

- Anders als in C, muss die Länge bzw. Elementzahl kein konstanter Ausdruck sein, z. B.:

```
// Kommandozeilenargument args[0] in eine ganze Zahl umwandeln
// und eine Reihe mit entsprechend vielen double-Werten erzeugen.
a = new double [Integer.parseInt(args[0])];
```

- Bei Angabe einer negativen Länge erhält man eine Ausnahme des Typs `NegativeArraySizeException`. (Die Länge 0 ist zulässig.)

- Die Länge einer Reihe `a`, d. h. die Anzahl ihrer Elemente, kann mittels `a.length` abgefragt werden.

- Bei der Erzeugung einer Reihe werden ihre Elemente mit einem typabhängigen Standardwert initialisiert:
  - `false` für Typ `boolean`
  - `'\0'` für Typ `char`
  - `0` bzw. `0.0` für alle numerischen Typen
  - `null` für alle Referenztypen (Klassen, Schnittstellen, Reihen)

- Nach Erzeugung einer Reihe kann ihre Länge nicht mehr verändert werden. Um eine Reihe quasi zu „vergrößern“ (oder zu „verkleinern“), muss man eine neue Reihe mit der gewünschten Länge erzeugen und die Elemente umkopieren.

- Für einen ganzzahligen Wert `i` zwischen 0 einschließlich und `a.length` ausschließlich liefert `a[i]` das `i`-te Element der Reihe `a`.

- Für andere Werte von `i` erhält man eine Ausnahme des Typs `ArrayIndexOutOfBoundsException`, d. h. es findet Bereichsprüfung statt.

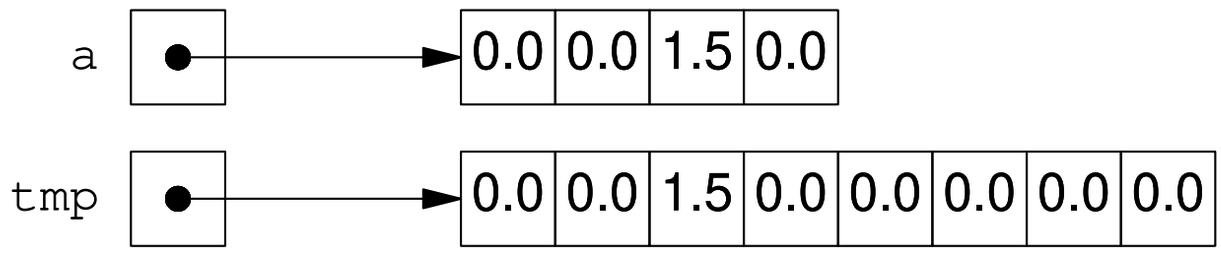
- Wenn `a` die Nullreferenz `null` ist, produzieren die Ausdrücke `a.length` und `a[i]` eine Ausnahme des Typs `NullPointerException`.

- Wenn `a` und `b` Reihenvariablen sind, wird bei einer Zuweisung `a = b` lediglich die *Referenz* `b` kopiert, aber nicht die Elemente der von `b` referenzierten Reihe.

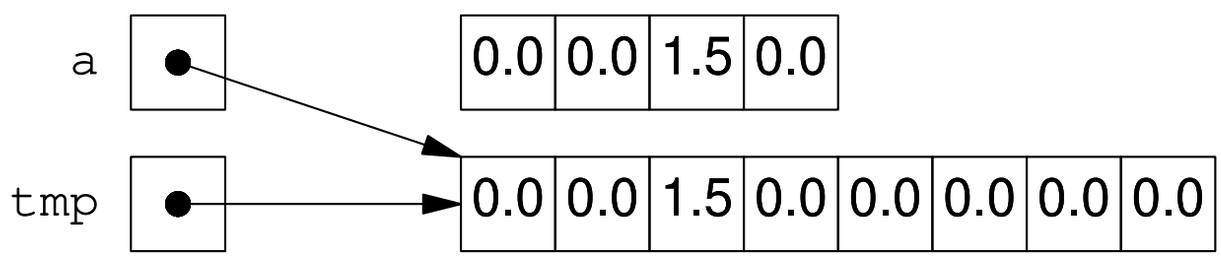
Beispiel

```
double [] a = new double [4];
a[2] = 1.5;

double [] tmp = new double [8];
for (int i = 0; i < a.length; i++) tmp[i] = a[i];
```




```
a = tmp;
```



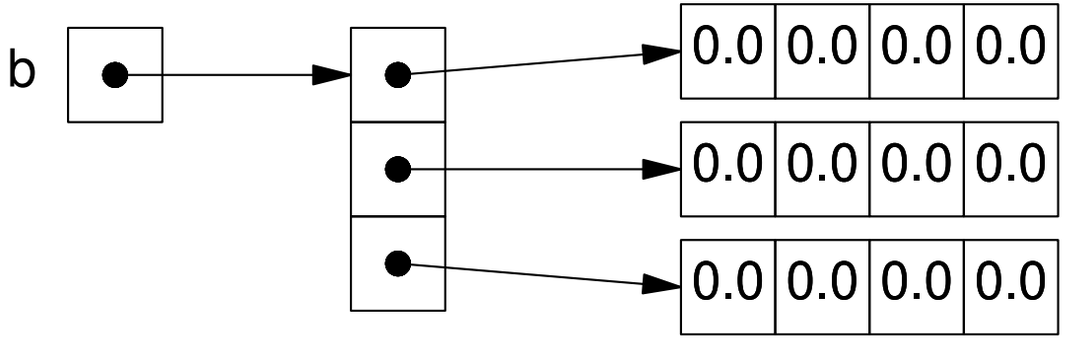

```
// Die nicht mehr referenzierte Reihe wird (zu einem nicht näher
// spezifizierten Zeitpunkt) vom System automatisch freigegeben.
```

Mehrdimensionale Reihen

- Mehrdimensionale Reihen erhält man indirekt als Reihen von Reihen, z. B.:

```
int m = ..., n = ...;
double [] [] b = new double [m] [n];
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; j++) {
    System.out.println(b[i][j]);
  }
}
```

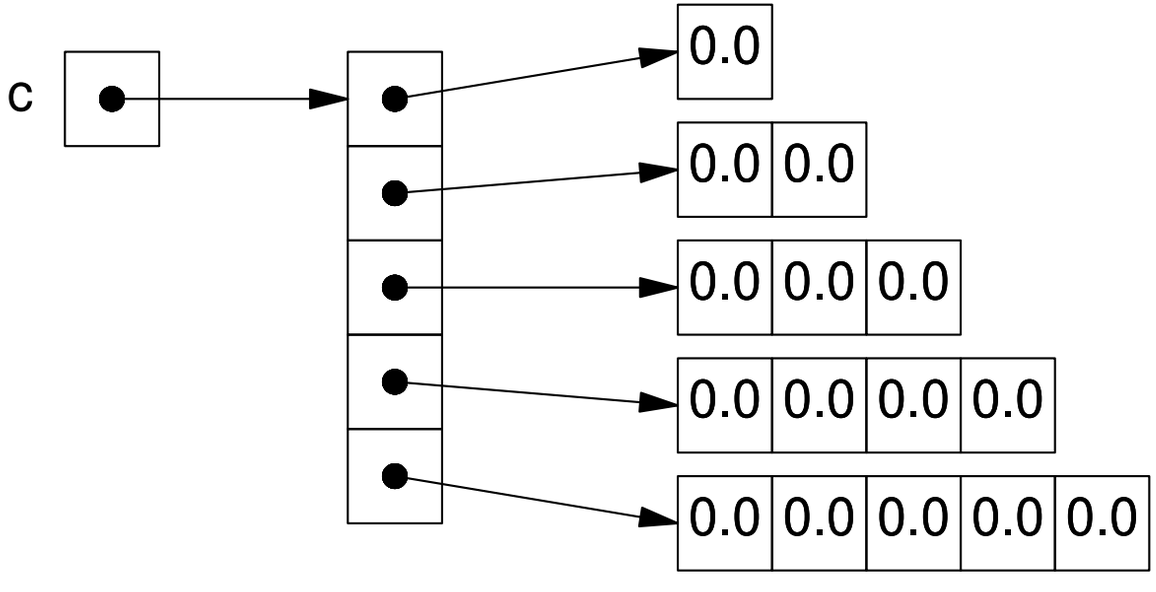
Hier ist `b` (eine Referenz auf) eine Reihe mit `m` Elementen (d. h. `b.length` liefert `m`); jedes dieser Elemente ist selbst wieder (eine Referenz auf) eine Reihe mit `n` Elementen des Typs `double` (d. h. `b[i].length` liefert jeweils `n`).



- Reihen von unterschiedlich großen Reihen kann man z. B. wie folgt erzeugen:

```
int n = ...;
double [] [] c = new double [n] [];
for (int i = 0; i < n; i++) {
  c[i] = new double [i+1];
}
```

Hier ist `c` (eine Referenz auf) eine Reihe mit `n` Elementen, die anfangs alle den Wert `null` besitzen; anschließend wird an jedes Element `c[i]` (eine Referenz auf) eine Reihe mit `i+1` Elementen des Typs `double` zugewiesen.



Reiheninitialisierer

- Reihen können auch durch Reiheninitialisierer erzeugt werden, z. B.:

```
double [] a = { 1.0, 2.0, 3.0 }; // double-Reihe mit 3 Elementen.

a = new double [] { 4.0, 5.0 };  // double-Reihe mit 2 Elementen.

double [] [] b = { { 1.0 }, { 2.0, 3.0 } };    // Zweidim. Reihe.
```

- Bei der Deklaration einer Reihenvariablen (erste und dritte Zeile im vorigen Beispiel) kann der Reiheninitialisierer `{ ..... }` direkt verwendet werden, in allgemeinen Ausdrücken (zweite Zeile) muss er mit einem `new`-Ausdruck kombiniert werden.

- Um mehrdimensionale Reihen zu erzeugen und zu initialisieren, können Reiheninitialisierer verschachtelt werden.

- Die Elemente eines Reiheninitialisierers können beliebige Ausdrücke sein.
  Bei mehrdimensionalen Reihen können sie selbst wieder Reihen sein.

3.2.4 Zeichenketten (strings)

- Zeichenkettenkonstanten wie z. B. `"abc"` besitzen in Java den vordefinierten Typ `String`.
- Die Länge eines `String`-Objekts `s` kann mit `s.length()` abgefragt werden. (Achtung: Die Länge einer Reihe `a` erhält man mittels `a.length`, ohne Klammern.)
- Für `i` zwischen 0 einschließlich und `s.length()` ausschließlich liefert `s.charAt(i)` das `i`-te Zeichen der Zeichenkette `s`. Für andere Werte von `i` erhält man eine Ausnahme des Typs `StringIndexOutOfBoundsException`.
- Wenn `s` die Nullreferenz `null` ist, produzieren die Ausdrücke `s.length()` und `s.charAt(i)` (sowie alle anderen Methodenaufrufe auf `s`) eine Ausnahme des Typs `NullPointerException`.
- Zeichenketten können mit dem Verkettungsoperator `+` verkettet werden, d. h. für `String`-Objekte `s` und `t` liefert `s + t` ein neues `String`-Objekt, das die Zeichen von `s` und `t` nacheinander enthält.
  Dementsprechend weist `s += t` die Verkettung von `s` und `t` wieder an die `String`-Variable `s` zu.
- Wenn nur ein Operand einer Verkettungsoperation eine Zeichenkette ist, wird der andere Operand automatisch in eine Zeichenkette umgewandelt (vgl. § 5.10).

- Beispiel:

```
int x = 2, y = 3;
System.out.println(
 "Die Summe von " + x + " und " + y + " ist " + (x + y) + ".");
```

- `s.equals(t)` überprüft, ob die Zeichenketten `s` und `t` (d. h. eigentlich die von `s` und `t` referenzierten `String`-Objekte) „gleich“ sind, d. h. die gleichen Zeichen in der gleichen Reihenfolge enthalten.

- Der Vergleich `s == t` überprüft lediglich, ob die Referenzen `s` und `t` gleich sind, d. h. ob `s` und `t` *dasselbe* `String`-Objekt referenzieren.

- Die Klasse `String` bietet zahlreiche weitere Operationen an, z. B. Suchen einzelner Zeichen oder Zeichenfolgen in einer Zeichenkette, Bilden von Teilzeichenketten usw.

- Anders als in C, sind Zeichenketten keine `char`-Reihen, und sie besitzen kein abschließendes Nullzeichen.

- Außerdem sind Zeichenketten *unveränderbar*, d. h. der Inhalt eines `String`-Objekts kann nach seiner Erzeugung nicht mehr geändert werden.
  Um eine Zeichenkette quasi zu „ändern“, muss man eine neue Zeichenkette mit geändertem Inhalt erzeugen.

3.3 Operatoren

3.3.1 Übersicht

Operator	*Anwendung*	*Bedeutung*
`new`	präfix	Objekterzeugung (vgl. § 3.2.3 und § 4.9)
`.name`	postfix	Feldzugriff (vgl. § 4.5)
`.name(args)`	postfix	Methodenaufruf (vgl. § 4.6)
`[index]`	postfix	Reihenelementzugriff (vgl. § 3.2.3)
`++`	postfix	Inkrement
`--`	postfix	Dekrement
`++`	präfix	Inkrement
`--`	präfix	Dekrement
`+`	präfix	Identische Funktion (unäres Plus)
`-`	präfix	Arithmetische Negation (unäres Minus)
`~`	präfix	Bitweises Komplement
`!`	präfix	Logische Negation
`(type)`	präfix	Typumwandlung (cast)

Operator	*Anwendung*	*Bedeutung*
`*`	infix	Multiplikation
`/`	infix	Division
`%`	infix	Rest bei Division
`+`	infix	Addition oder Verkettung von Zeichenketten
`-`	infix	Subtraktion
`<<`	infix	Bitverschiebung nach links
`>>`	infix	Arithmetische Bitverschiebung nach rechts
`>>>`	infix	Logische Bitverschiebung nach rechts
`< > <= >=`	infix	Numerische Vergleiche
`instanceof`	infix	Dynamischer Typtest (vgl. § 5.7)
`==`	infix	Test auf Gleichheit
`!=`	infix	Test auf Ungleichheit
`&`	infix	Bitweise oder Boolesche Und-Verknüpfung
`^`	infix	Bitweise oder Boolesche Exklusiv-Oder-Verknüpfung
`|`	infix	Bitweise oder Boolesche (Inklusiv-)Oder-Verknüpfung
`&&`	infix	Logische Und-Verknüpfung
`||`	infix	Logische Oder-Verknüpfung
`? :`	infix	Verzweigung
`=`	infix	Zuweisung
`+= -= ...`	infix	Kombinierte Zuweisung
`->`	infix	Lambda-Ausdruck (vgl. § 7.2)

3.3.2 Erläuterungen

- Blau geschriebene Wörter sind Platzhalter.
- In den Tabellen besitzen alle Operatoren einer Gruppe denselben Vorrang, der höher ist als der Vorrang der nächsten Gruppe.
- Daher ist der Ausdruck `-a++ * b < c + d` beispielsweise äquivalent zu `((-(a++)) * b) < (c + d)`.
- Die Zuweisungsoperatoren und der Verzweigungsoperator sind rechtsassoziativ, alle anderen binären Operatoren linksassoziativ.
- Daher ist der Ausdruck `a - b - c` beispielsweise äquivalent zu `(a - b) - c` (und nicht zu `a - (b - c)`), während `a = b = 1` äquivalent zu `a = (b = 1)` ist.

3.3.3 Anmerkungen

- Anders als in C, wird der linke Operand eines Operators garantiert vor dem rechten ausgewertet.
  (Zum Beispiel liefert der Ausdruck `a[i++] - a[i++]` für `i = 0` garantiert den Wert von `a[0] - a[1]`; in C wäre z. B. auch `a[1] - a[0]` zulässig.)
- Ebenso werden die Argumente eines Methodenaufrufs garantiert von links nach rechts ausgewertet.
  (Zum Beispiel wird beim Ausdruck `f(i++, i++)` für `i = 0` garantiert der Methodenaufruf `f(0, 1)` ausgeführt; in C wäre z. B. auch `f(1, 0)` zulässig.)
- Die logischen Operatoren `!`, `&&` und `||` können nur auf `boolean`-Werte angewandt werden und liefern als Resultat wieder einen `boolean`-Wert (`true` oder `false`).
  (In C können sie auf Werte aller elementaren Typen angewandt werden und liefern als Resultat einen `int`-Wert, 1 oder 0.)
- Bei den Operatoren `&&` und `||` wird der rechte Operand (wie in C) nur ausgewertet, wenn dies zur Ermittlung des Ergebnisses notwendig ist.
  (Zum Beispiel produziert der Ausdruck `a != null && a.length > 0` für `a` gleich `null` keine `NullPointerException`, weil `a.length` in diesem Fall gar nicht ausgewertet wird.)

- Bei einer Verzweigung `x ? y : z` wird (wie in C), abhängig vom Wert des ersten Operanden (`x`), entweder nur der zweite (`y`) oder nur der dritte Operand (`z`) ausgewertet.
- Anders als in C, wird bei ganzzahliger Division garantiert „in Richtung 0“ gerundet. (In C ist das Rundungsverhalten bei negativen Operanden nicht festgelegt.)
- Für den Rest bei ganzzahliger Division gilt für alle möglichen Werte von `x` und `y`: `x % y == x - x/y * y`.
- Zum Beispiel:

```
 14 /  3  →   4        14 %  3  →  2
-14 /  3  →  -4       -14 %  3  → -2
 14 / -3  →  -4        14 % -3  →  2
-14 / -3  →   4       -14 % -3  → -2
```

- Für positives `y` erhält man mittels `(x % y + y) % y` für beliebige `x` den mathematisch korrekten Wert `x` mod `y`, der immer zwischen 0 und `y - 1` liegt.
- In C führt der Operator >> entweder eine logische oder eine arithmetische Verschiebung nach rechts durch (was nur bei vorzeichenlosen Werten gleichbedeutend ist); der Operator >>> existiert in C nicht.
- Der Komma-Operator, mit dem man in C die sequentielle Ausführung von Teilausdrücken beschreiben kann, existiert nicht.

3.3.4 Fehler und Ausnahmen

- Wenn `new` wegen Speicherplatzmangels kein Objekt erzeugen kann, erhält man einen Fehler des Typs `OutOfMemoryError`.
- Wenn man mit `new` eine Reihe mit negativer Länge erzeugen will, erhält man eine Ausnahme des Typs `NegativeArraySizeException`.
- Ein Feldzugriff, ein Methodenaufruf oder ein Reihenelementzugriff über eine Nullreferenz produziert eine Ausnahme des Typs `NullPointerException`.
- Ein Reihenelementzugriff mit einem unzulässigen Index produziert eine Ausnahme des Typs `ArrayIndexOutOfBoundsException`.
- Eine unzulässige Typumwandlung produziert eine Ausnahme des Typs `ClassCastException` (sofern sie nicht schon vom Übersetzer als unzulässig erkannt wird; vgl. § 5.8).
- Eine ganzzahlige Division durch 0 (bzw. eine entsprechende Rest-Operation) produziert eine Ausnahme des Typs `ArithmeticException`.
  Eine Gleitkommadivision wie z. B. `1.0/0.0` oder `-1.0/0.0` liefert jedoch einen unendlichen Wert mit entsprechendem Vorzeichen, während `0.0/0.0` den Sonderwert „not a number“ liefert.