

5.2 Strukturierte Datentypen

(Datenstrukturen)

keinen Listes: unstrukturierte Typen (atomar, einfach)

in Java: double, float ← Fließkommazahlen
int long ← ganze Zahlen
char ← Zeichen
boolean ← Wahrheitswerte

strukturierte Typen

↑
zusammengesetzt

haben neben Wertebereich und Operationen

- Komponenten - Daten (atomar oder strukturiert)
- Regeln für das Zusammenwirken der Komponenten

Beispiel : Tabellen, Personalbogen, Konto

Mathematik : Vektor, Matrizen

Prog. Sprachen haben einige solche strukturierte Typen eingebaut
(meist Arrays und Strings)

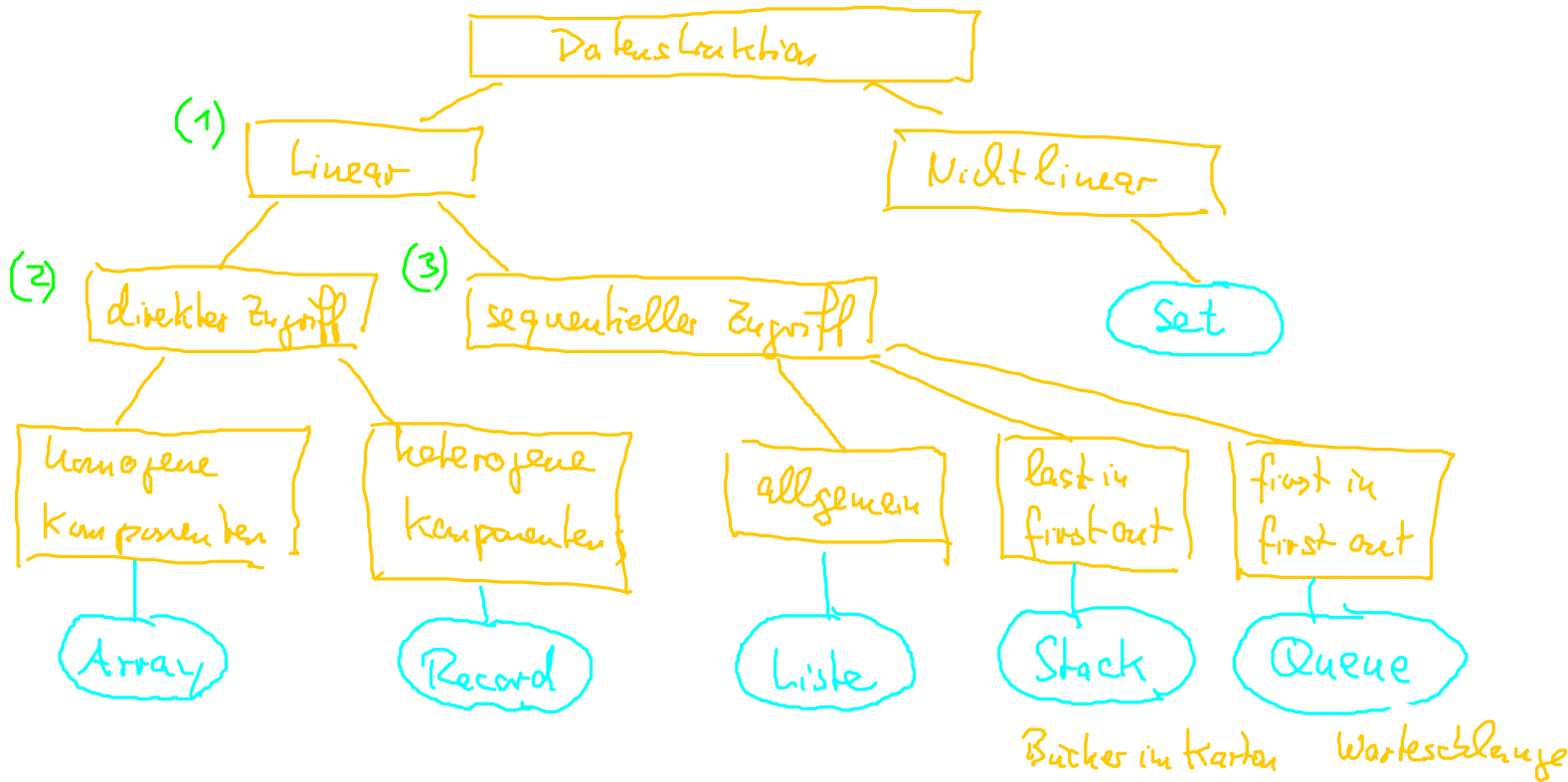
breiten Möglichkeiten um strukturierte Typen selber zu erzeugen

↑
in Java besonders mächtig durch Klassen
und Vererbung

Fraction war ein solcher selbst def. zusammengesetzter

Typ

Übersicht über Strukturierungs möglichkeiten:



- (1) linear $\hat{=}$ es gibt lineare Reihenfolge der Komponenten
- \exists eindeutige erste Komponente
 - \exists ---- letzte ----
 - jede Komponente (außer erster) hat einen eindeutigen Vorgänger
 - jede Komponente (außer letzter) hat einen eindeutigen Nachfolger

(2) direkter Zugriff (random access)

kann auf jede Komponente ohne Zeitverzögerung zugreifen
Bücher im Regal, Speicherplätze im Hauptspeicher

(3) kann bzgl. der linearen Ordnung nur sequenziell zugreifen



Bücher, im Karton, Zug an Verladestation

5.3 Arrays

↑ verbreitetste Datenstruktur, exist. in jeder Prog. Sprache

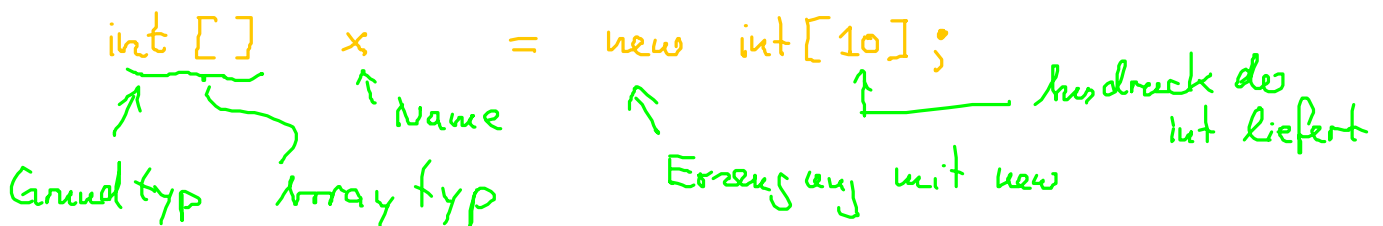
Merkmale

- feste Komponentenanzahl (in Java erst zur Laufzeit festgelegt werden) ^(muss i.e.)
- direkter Zugriff auf Komponenten mittels Indices
Indices können berechnet werden
(Indexarithmetik)
- homogener Grundtyp

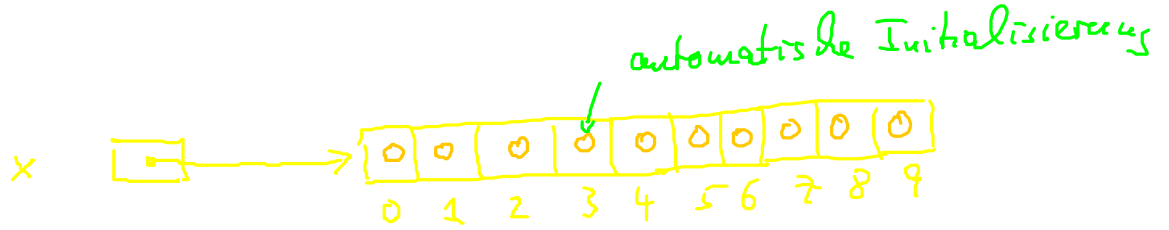
Vektoren in der Mathematik entsprechen den 1D Arrays

Arrays in Java:

eingebaute Array Typen als Referenztyp



im Speicher



Zugriff auf Komponenten mit Syntax

$\langle \text{Feldzugriff} \rangle ::= \langle \text{Referenz Ausdruck} \rangle [\langle \text{Ausdruck} \rangle]$

↑
Zugriff auf
Komponente

↑
muss
Referenz liefern

↑
muss int
liefern

int $a = x[3];$

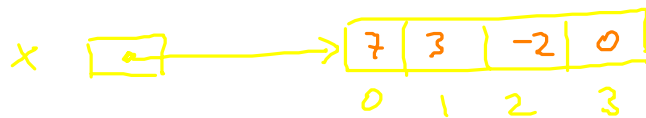
greift auf Komp. mit Index 3 von x zu, weist diesen Wert der Variable a zu

$x[7] = 3;$

der Komponente mit Index 7 wird der Wert 3 zugewiesen

Bei Definition ist direkte Zuweisung von Werten möglich gemäß

int $x = \{ 7, 3, -2, 0 \};$



Außerdem ist int x = äquivalent zu int $x[]$ =
deutet Klammer den Array typ an

Arrays spielen Hand in Hand mit for Schleifen

```
int n = 10;
```

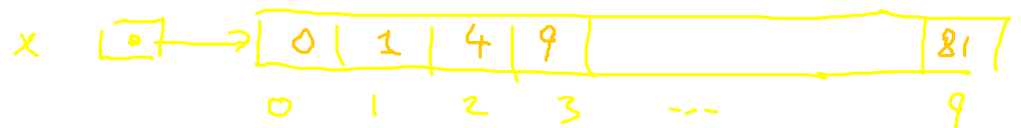
```
int[] x = new int[n];
```

```
for (int i=0; i < x.length; i++) {
```

```
    x[i] = i * i;
```

```
}
```

Arrays kennen ihre Länge!



```
int[] y; ← nur Deklaration
```

möchte Quadratzahlen aus `x` nach `y` kopieren

```
y = x; ← erzeugt keine Kopie
```

`y, x` zeigen jetzt auf denselben Speicherbereich



Änderung von `y[3]` "ändert" auch `x[3]`

Wie also eine echte Kopie herstellen?

2 Möglichkeiten

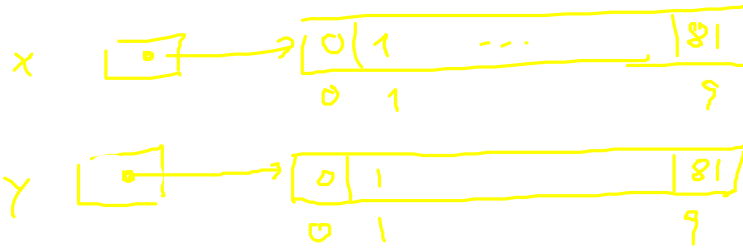
① bei eingebauten Typen mit for Schleife

```
y = new int[x.length];
```

```
for (int i=0; i < x.length; i++) {
```

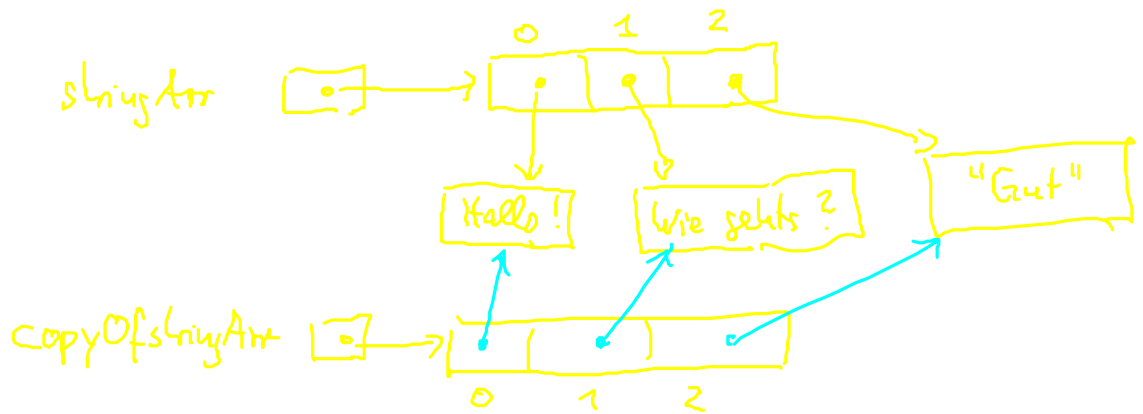
$y[i] = x[i];$

}



② funktioniert nicht mehr, wenn der Grundtyp ein Referenztyp ist (z.B. String)

`String[] stringArr = { "Hallo!", "Wie geht's?", "Gut" };`



Versuch mit for-Schleife eine Kopie herzustellen wie bei ① scheitert!

Zum echten Kopieren gibt es bei den meisten Referenztypen (bei Arraytypen der Fall) die Methode `clone()`

[alle, die das Interface `Cloneable` implementieren]

$\text{String}[] \text{ copyOf}(\text{String} \text{ arr}) = (\text{String}[]) \text{String} \text{ arr} . \text{clone}();$
 ↑
 String array
 wird erwartet

muss
 casten

kopiert alle member
 Objekt

Nachmal zurück zu Komponenten Zugriff

$\langle \text{Feldzugriff} \rangle :: = \langle \text{Referenz Ausdruck} \rangle [\langle \text{Ausdruck} \rangle]$

In einfacher Form $x[3] = 7;$

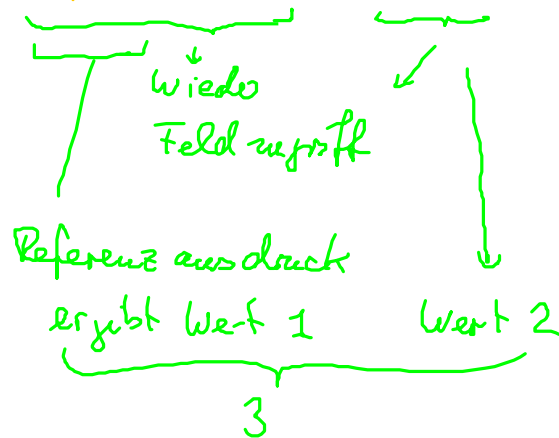
$\text{int}[] x = \{1, 2, 3, 4\},$

$y = \{5, 6, 7, 8\},$

$z;$

$y[x[0]] = 9;$

$(z = y)[(y = x)[0] + y[1]] = 0;$



soll demonstrieren,
 dass man die
 Syntaxregeln
 nicht interpretieren
 können muss
 wir werden keinen
 solchen Code schreiben

Java überprüft zur Laufzeit, ob Feldzugriff

und wirklich Komponenten liefert, die im Array vorhanden sind, andernfalls gibt es eine

ArrayIndexOutOfBoundsException

5.3.2 Mehrdimensionale Arrays

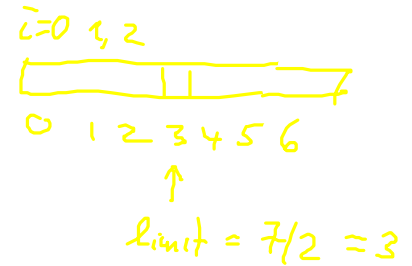
Grundtyp sind wieder Arrays → auf mehrere Werte
verweisen

hier noch Beispiel: Umdrehen eines Arrays



int[] vector = new int[n];

// Belegung mit Werten



int tmp; → int limit = vector.length / 2;

for (int i = 0; i < limit; i++) {

tmp = vector[i];

vector[i] = vector[vector.length - 1 - i];

↖ Indexarithmetik

vector[vector.length - 1 - i] = tmp;

}