5.2 Strukturierte Datentypen
(Datenstrukturen)

Kennen bisher: 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

Alltag: Tabellen, Personalbogen, Konto
Mathematik: Vektor, Matrizen

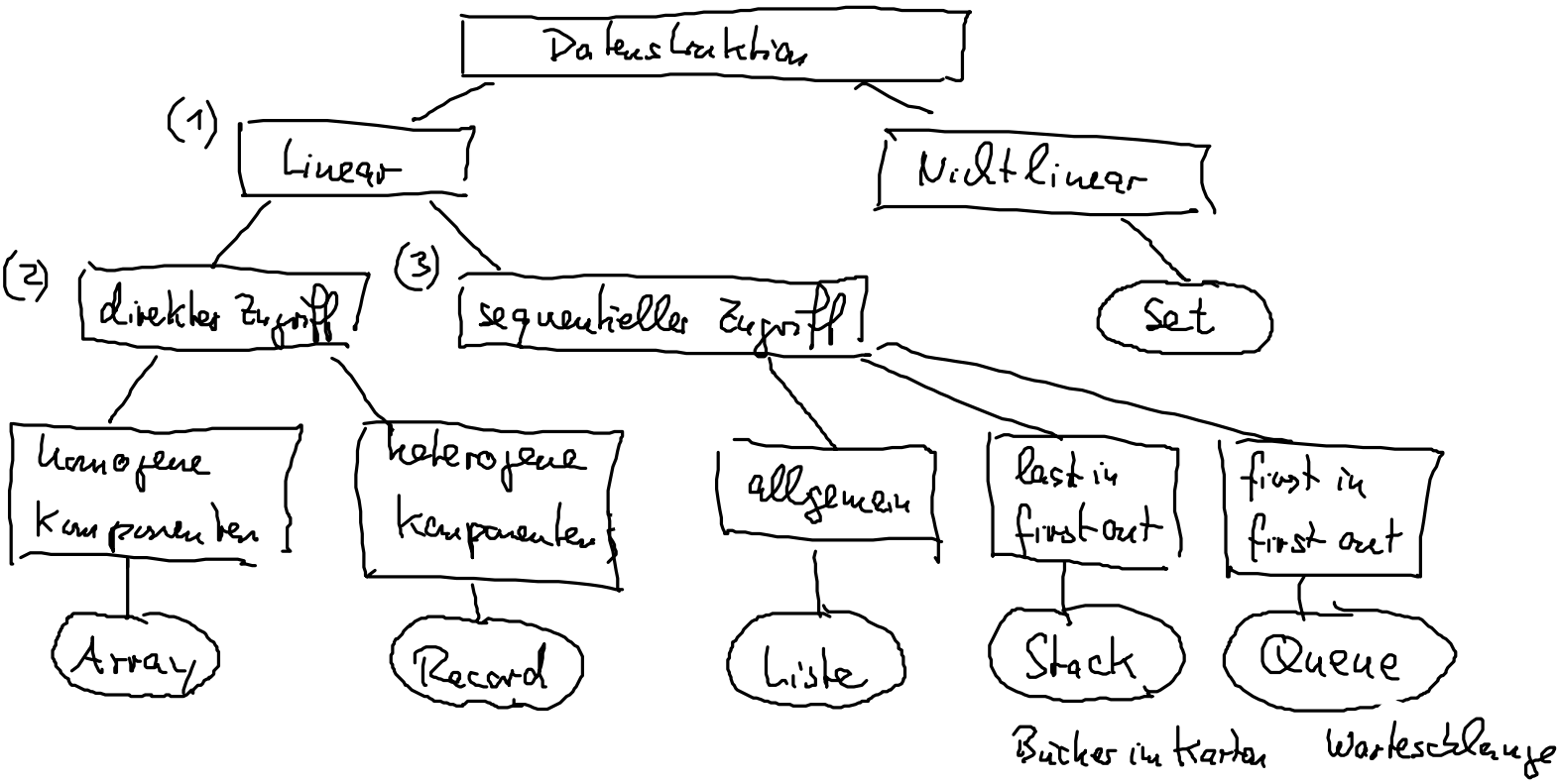
Prog. Sprachen haben einige solche strukturierte Typen eingebaut
(meist Arrays und Strings)

bieten Möglichkeiten um strukturierte Typen selber zu erzeugen
↑ in Java besonders mächtig durch Klassen und Vererbung

Function war ein solcher selbstdef. zusammengesetzter

Typ

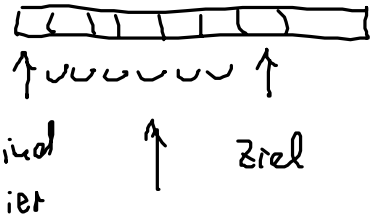
Übersicht über Strukturierungsmöglichkeiten:



(1) linear ≙ es gibt lineare Reihenfolge der Komponenten
- ∃ eindeutige erste Komponente
- ∃ ---- 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 Zeitverzug zugreifen
Bücher im Regal, Speicherplätze im Hauptspeicher

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



sind hier ↑ Ziel

müssen alle "lesen" bis wir am Ziel sind

Bücher, im Karton, Zug am Verladebahnhof

5.3 Arrays

↑ verbreitetste Datenstruktur, exist. in jeder Prog. Sprache

Kennzeichen

- feste Komponentenzahl (in Java (muss sie) erst zur Laufzeit festgelegt werden)
- direkter Zugriff auf Komponenten mittels Indices

  Indices können berechnet werden

  (Indexarithmetik)
- homogener Grundtyp

  Vektoren in der Mathematik entsprechen am ehesten Arrays

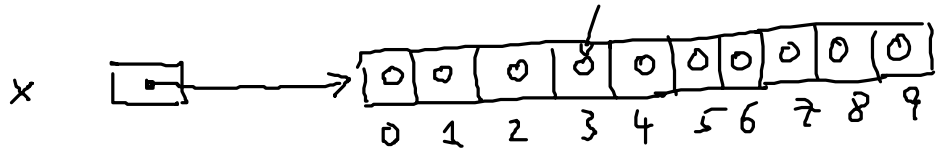
Arrays in Java:

eingebaute Array Typen als Referenztyp

```
int[]  x  =  new int[10];
```

- `int[]` ↑ Grundtyp, Array typ
- `x` ↑ Name
- `new` ↑ Erzeugung mit new
- `int[10]` ↑ Ausdruck der int liefert

im Speicher



Zugriff auf Komponenten mit Syntax

<Feldzugriff> ::= <Referenzausdruck> [<Ausdruck>]

↑ Zugriff auf Komponente

↑ muss Referenz liefern

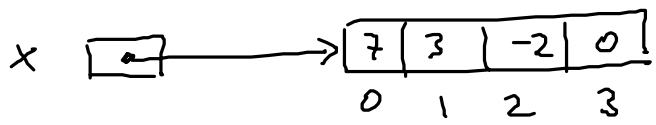
↑ muss int liefern

int a = x[3]; greift auf Komp. mit Index 3 von x zu, weist diesen Wert der Variablen 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 };



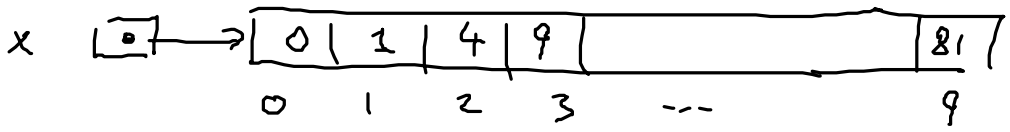
Außerdem ist int [] x = äquivalent zu int x [] =

deutet klarer 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!

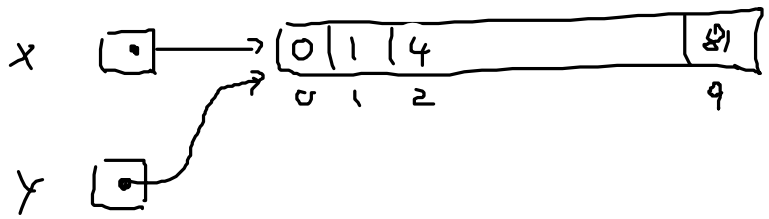


`int[] y;` ⇐ nur Deklaration

möchte Quadratzahlen aus x und y kopieren

`y = x;` ⇐ erzeugt keine Kopie

y, x zeigen jetzt auf denselben Speicherbereich



Änderung von y[3] "ändert auch x[3]

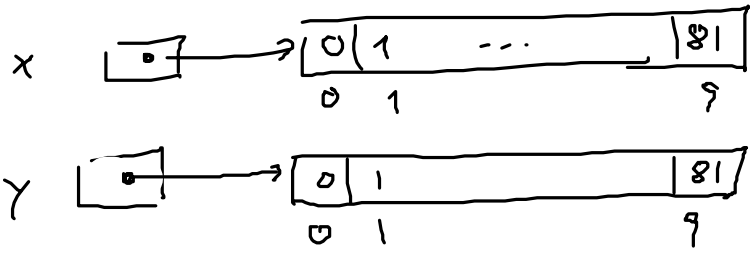
Wie also eine echte Kopie herstellen?

2 Möglichkeiten

① Bei eingebauten Typen mit for Schleife

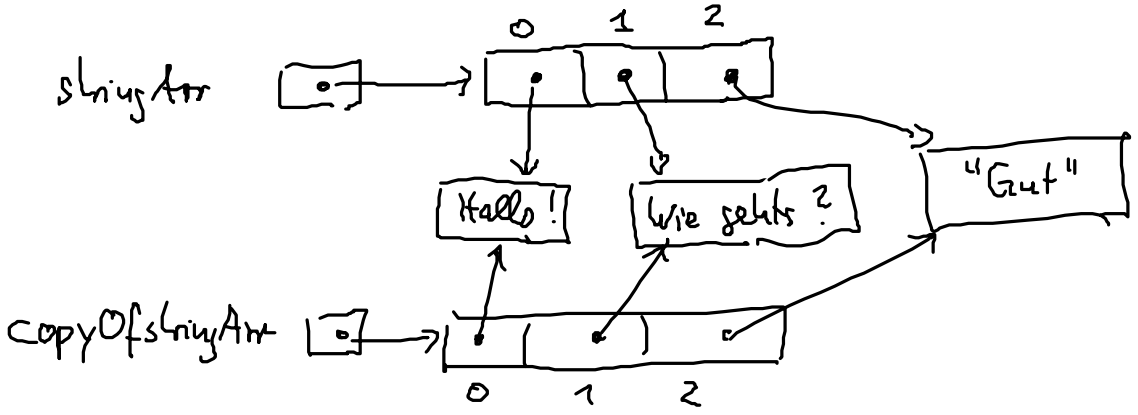
```
y = new int[x.length];
for (int i=0; i < x.length; i++) {
```

```
    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 Clonable implementieren]

```
String[] copyOfStringArr = (String[]) stringArr.clone();
```

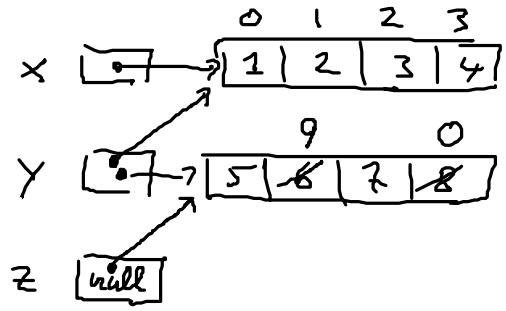
- `String[]` ↑ Stringarray wird erwartet
- `(String[])` muss gecastet
- `stringArr.clone()` kopiert allgemeines Objekt

Nochmal zurück zu Komponenten zugriff

<Feldzugriff> ::= <Referenzausdruck> [<Ausdruck>]

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

```
int[] x = {1, 2, 3, 4},
      y = {5, 6, 7, 8},
      z;
y[x[0]] = 9;
(z = y)[(y = x)[0] + y[1]] = 0;
```



- `(y = x)[0]` → wieder Feldzugriff
- Referenzausdruck ergibt Wert 1
- `y[1]` → Wert 2
- 3

Soll demonstrieren, dass man die Syntaxregeln richtig interpretieren können muss

Wir werden keinen solchen Code schreiben

Java überprüft zur Laufzeit, ob Feldzugriff

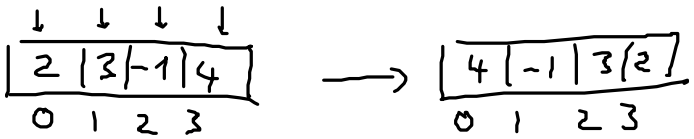
auch wirklich Komponenten liefert, die im Array vorhanden sind, andernfalls gibt es eine

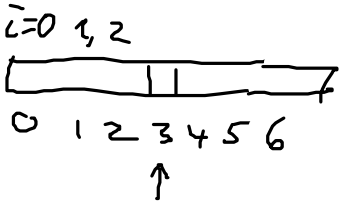
ArrayIndexOutOfBoundException

5.3.2 Mehrdimensionale Arrays

Grundtyp sind wieder Arrays → auf unterste Ebene verschieben

Hier noch Beispiel: Umdrehen eines Arrays






```
int[] vector = new int[n];
// Belegung mit Werten
int limit = vector.length / 2;
int tmp; →
for ( int i = 0; i < limit , i++ ) {
    tmp = vector[i];
    vector[i] = vector[ vector.length -1 -i ];
                                  ↖ Indexarithmetik
    vector[ vector.length - 1 - i ] = tmp;
}
```