- Wegen Gruppenzusammenführen:
  Email an Madeleine Theile → www.
- Probleme mit OA Nr. 3 → wird bis spät. gelöst
- Erinnerung: Fr, 23.11. dies
  ⇒ Abgabe der PA bis spät. 14.00
  (geht schon ab 8:00)

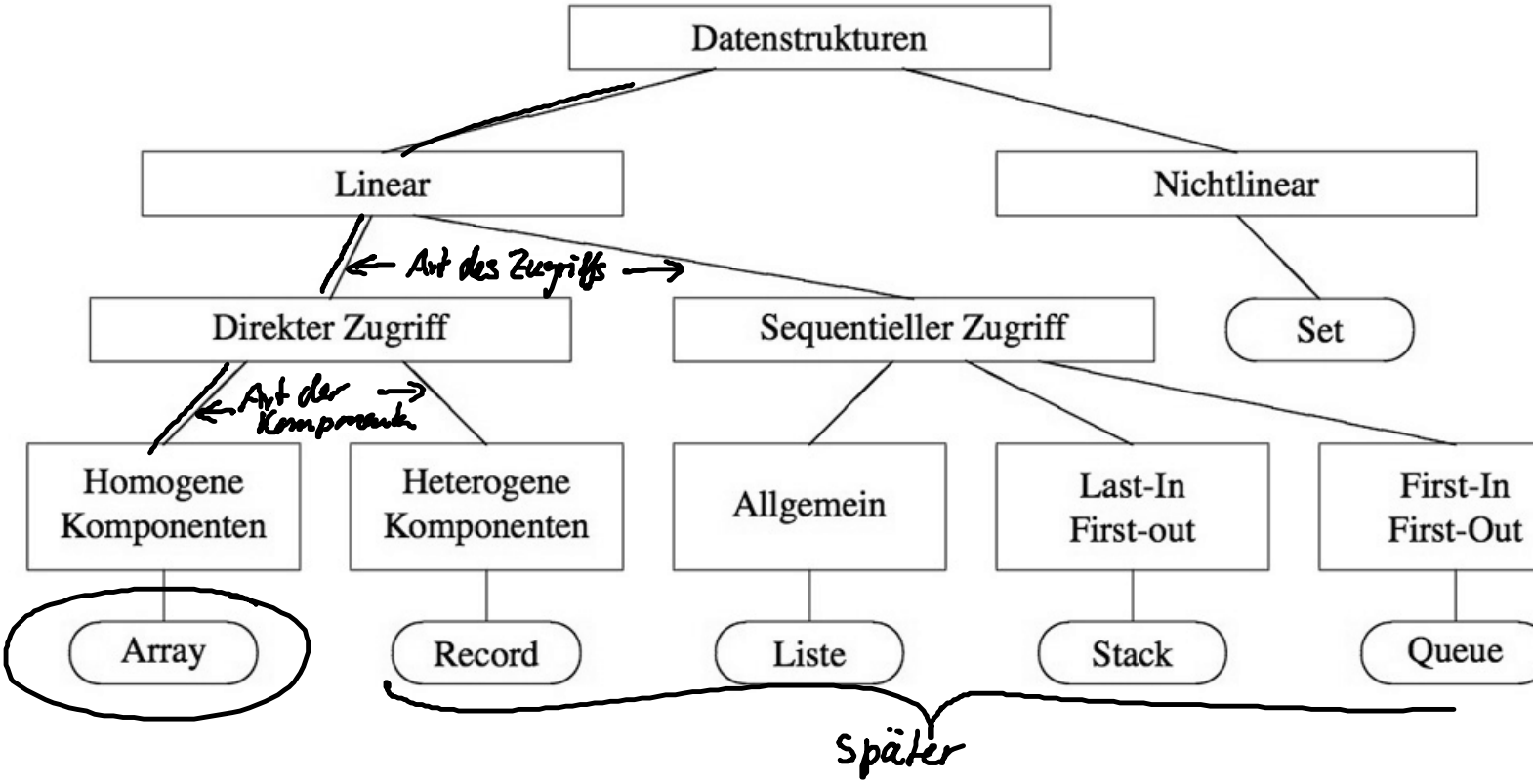
---

§5. Arrays

Bisher: einfache, unstrukturierte Datentypen
Jetzt: zusammengesetzte, strukturierte Datentypen

Arrays ← Standard strukturierter Datentyp in allen Progr.sprachen

Fassen Daten zu komplexeren Objekten zusammen, heißen deswegen
Datenstrukturen



Strukturierte Typen oder Datenstrukturen haben (neben *Wertebereich* und *Operationen*)

- *Komponenten-Daten*, die atomar oder wieder strukturiert sein können
- *Regeln*, die das Zusammenwirken der Komponenten zur gesamten Struktur definiere

Eine *lineare Datenstruktur* hat (bei mindestens 2 Komponenten) eine *Ordnung* auf den Komponent mit folgenden Eigenschaften:

- Es gibt eine eindeutige erste Komponente.
- Es gibt eine eindeutige letzte Komponente.
- Jede Komponente (außer der ersten) hat einen eindeutigen Vorgänger.
- Jede Komponente (außer der letzten) hat einen eindeutigen Nachfolger.

Direkter Zugriff (auch *random access* genannt) bedeutet, dass man auf jede beliebige Komponent zugreifen kann, ohne vorher auf andere Komponenten zugreifen zu müssen. Beispiele sind:

- Ein Regalbrett mit Büchern; auf jedes Buch kann direkt zugegriffen werden.
- CDs mit direkter Ansteuerung von Musikstücken.

Sequentieller Zugriff bedeutet, dass man auf die i-te Komponente nur zugreifen kann, nachdem man vorher auf die Komponenten $1, 2, \ldots, i-1$ zugegriffen hat. Beispiele sind:

- Ein Stapel von Büchern; um das i-te zu nehmen, müssen erst die $i-1$ obersten entfernt werden
- Musikstücke auf einem Tonband.

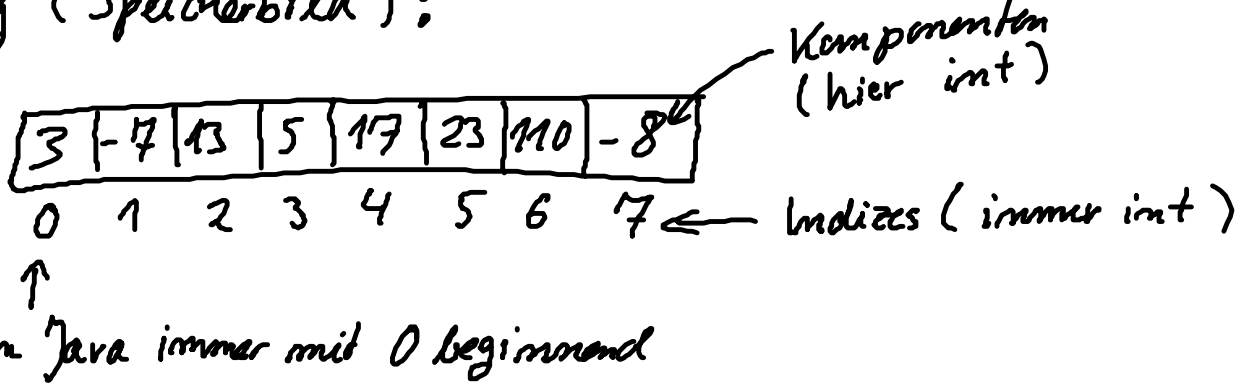
Kennzeichen der Datenstruktur Array sind:

- feste Komponentenzahl (die in Java erst zur Laufzeit festgelegt werden muss),
- direkter Zugriff auf Komponenten mittels *Indizes*,
- homogener Grundtyp,
- Indizes können berechnet werden.

→ in Mathematik ≙ Vektor

Arrays haben einen Indextyp (in Java nur int)
und einen Komponententyp (in Java jeder Typ möglich)

Vorstellung (Speicherbild):

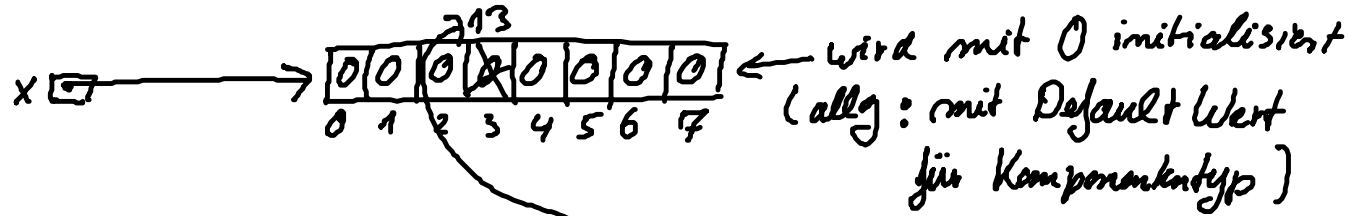


Arrays sind Referenztypen, d.h. Array Objekte müssen mit "new" erzeugt werden.

```
int[] x = new int[8];
```

Komponententyp — Array — Name — neue Erzeugung mit "new" — Anzahl der Komponenten

Speicherbild dazu:



Zugriff auf Komponenten („Feldzugriff"):

```
x[3] = 13; // Zuweisung an x[3]
int a = x[0]; // a = 0
```

Indizes können berechnet werden (Ausdruck vom Typ int):

```
int a;
int b = 2;
a = x[2*b+1]; // a = x[5]
```

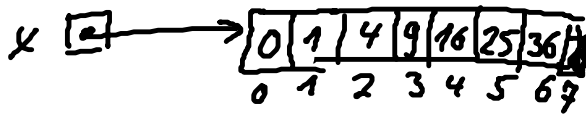
beliebiger Ausdruck mit Rückgabetyp int

Es muss ein gültiger Index benutzt werden, d.h. verboten ist z.B.:

```
a = x[-1]; // throws ArrayIndexOutOfBoundsException
a = x[8]; //  "                    "
```

Vorteil von Arrays u.a.: Man kann sie durchlaufen und dabei Statements ausführen.

```
int i = 0;
while (i < 8) {
    x[i] = i*i;
    i++;
}
```



üblicherweise mit for-Schleife:

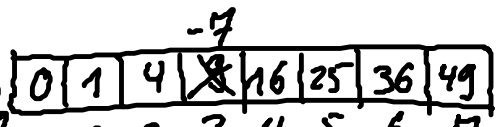
```
for (int i=0; i<8; i++) {
    x[i] = i*i;
}
```

noch kompakter:

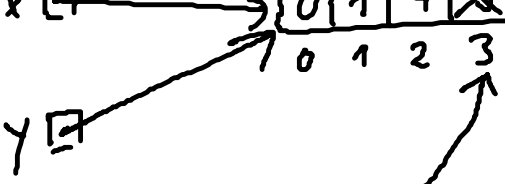
```
for (int i=0; i<8; x[i]=i*i++);
```

Initialisierung | logische Bedingung | Aktualisierungsausdruck

Arrays sind Referenztypen

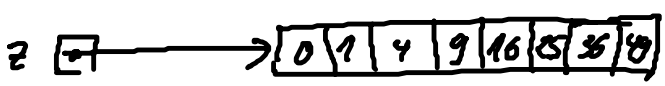



```
int[] y;
y = x;
y[3] = -7;
```



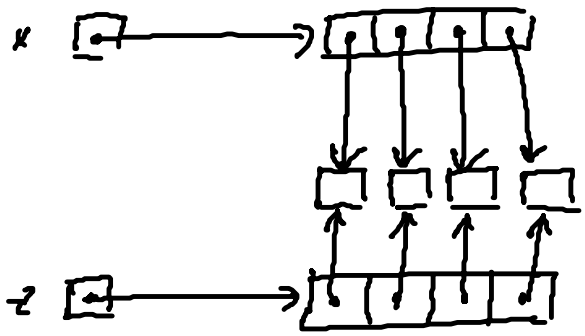
Kopieren von Arrays entweder mit clone() oder direkt mit

```
int[] z = new int[8];
for (int i=0; i<8; i++) {
    z[i] = x[i];
}
```



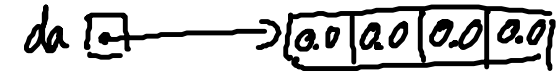
Komponententyp darf jeder Typ sein (auch Referenztyp)

Für Referenztypen als Komponententyp funktioniert das Kopieren mit der for-Schleife so nicht mehr, denn:

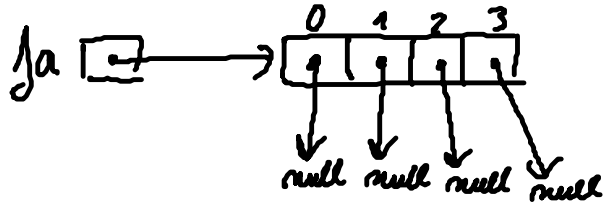


Hier hilft Methode clone(), über die Arraytypen verfügen.

```
double[] da = new double[4];
```




```
Fraction[] fa = new Fraction[4];
```



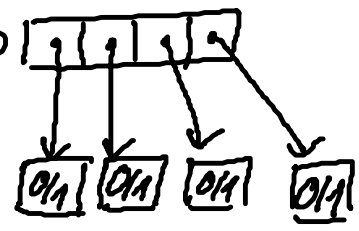
"leere Adresse"
Default Wert bei Referenztypen

```
fa[0].getNumerator(); // throws NullPointerExc.
```

beliebter Fehler

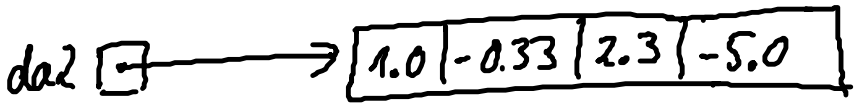
richtig: Zuerst Objekte erzeugen:

```
for (int i=0; i<4; i++) {
    fa[i] = new Fraction();
}
```



Alternativ bei Standardtypen und Strings Erzeugung auch direkt per „Initialisierungsliste":

```
double[] da2 = {1.0, -0.33, 2.3, -5.0}
```




```
String[] stra = {"Hallo", "hier", "bin", "ich"};
```

Jedes Array kennt seine Länge:

```
stra.length
```

stra → Arrayname

length → public Variable length (keine Methode wie bei Strings)

Mach for-Schleifen einfach:

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