Vorlesung an der

Berufsakademie Oldenburg

Unterrichtseinheit 18

Referenzdatentypen

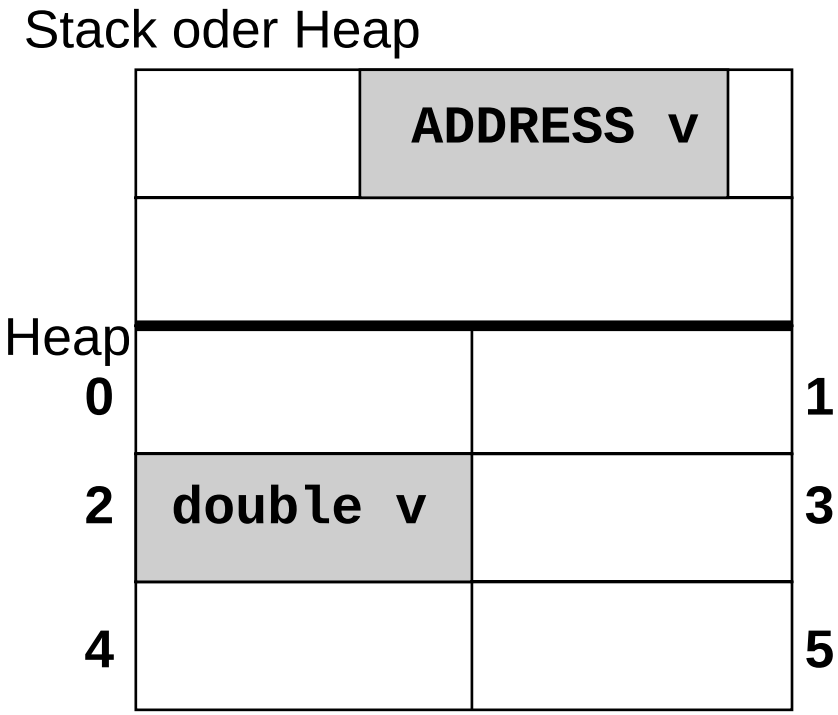
Dr. Dietrich Boles

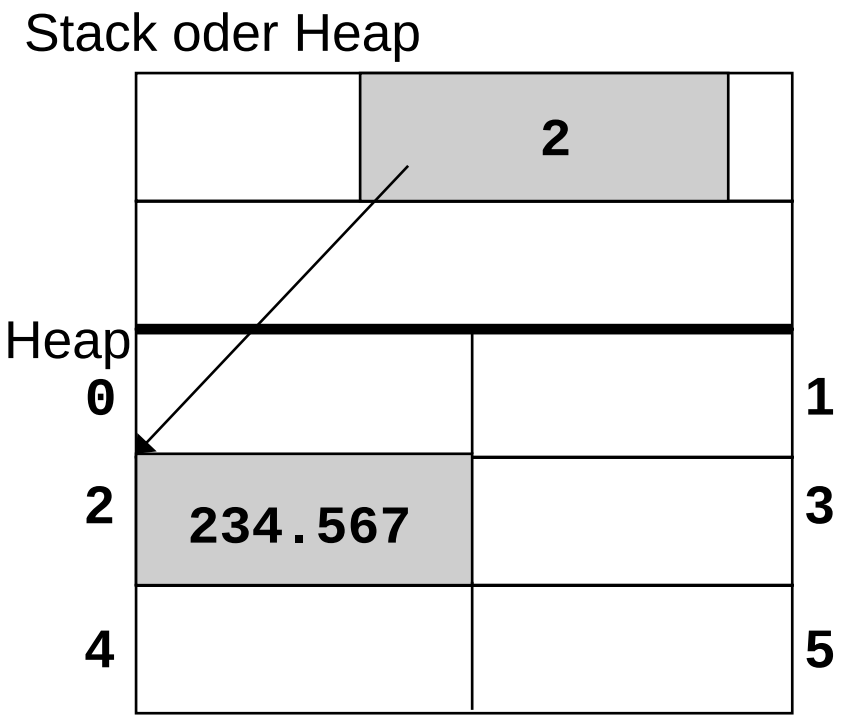
Gliederung

- Referenzdatentypen
  - Schema
  - Anmerkungen
  - Literale
  - Zuweisung
  - Gleichheit
  - Parameter
  - Funktionsrückgabewerte
  - lokale Variablen

Schema (1)

- Zeiger wie in C++ gibt es in Java nicht, aber Referenzvariablen
- Grundlage für Objektrepräsentation
- Datentyp ADDRESS
- ADDRESS-Variablen: Speicherung von Speicheradressen





Schema (2)

```
ADDRESS(double) v, w;     // Anlegen zweier ADDRESS-
                          // Variablen auf dem Stack

v = new double();         // Speicherplatzzuweisung (Heap)
w = new double();         // v und w enthalten als Werte
                          // Speicheradressen

*v = 4711.0;              // Zuweisung eines Wertes an die
                          // Speicheradresse, auf die v zeigt
*w = *v + 34.567;         // dito für w (Dereferenzierung)

v = w ;                   // v und w zeigen nun auf denselben
                          // Speicherplatz

delete v;                 // Speicherplatzfreigabe
```

Anmerkungen (1)

- Motivation
  - es existieren bestimmte Variablen, die lediglich **Adressen** ("**Referenzen**", "Zeiger") auf andere Speicherbereiche speichern können
  - die Kontrolle über die referenzierten Speicherbereiche liegt beim Programmierer bzw. Anwendungsprogramm, nicht beim Laufzeitsystem
  - das Anwendungsprogramm kann zur Laufzeit entscheiden, wie viel Speicherplatz denn tatsächlich benötigt wird; diese Menge fordert es sich mittels des `new`-Operators explizit an
  - ein "Garbage Collector" gibt automatisch nicht mehr benötigte Speicherbereiche frei
- Referenzdatentypen in Java sind Felder und Klassen/Objekte

Anmerkungen (2)

- es existieren einige Unterschiede zwischen
  "normalen" Variablen (Variablen von einem Standarddatentyp) und
  Referenzvariablen (Variablen vom Referenzdatentyp):

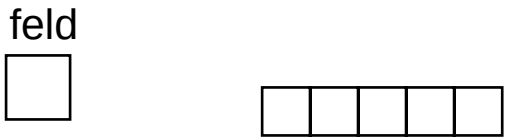
  - Literale
  - Zuweisung
  - Gleichheit
  - Parameterübergabe
  - Funktionswert
  - lokale Variablen

Literale

- Literal für Referenzdatentypen: **`null`**
- **`null`** ist Default-Wert bei der Initialisierung von Referenzvariablen
- Entspricht: Referenzvariable zeigt auf keinen Speicherbereich

```
int[] feld = new int[5];
...
feld = null;

if (feld != null) {
  feld[0] = 47;
}
```



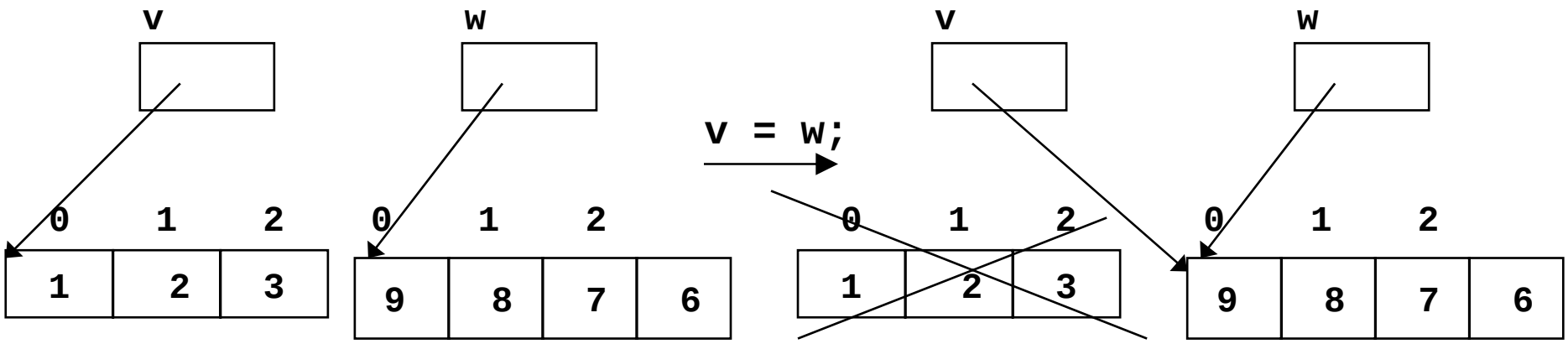
Garbage-Collector wird aktiv!

Zuweisung (1)

Adress-Zuweisung:

```
int[] v = {1, 2, 3};
int[] w = {9, 8, 7, 6};
v = w; // Adress-Zuweisung
```

```
v[0] = -77;
w[0] == -77 -> true!
```



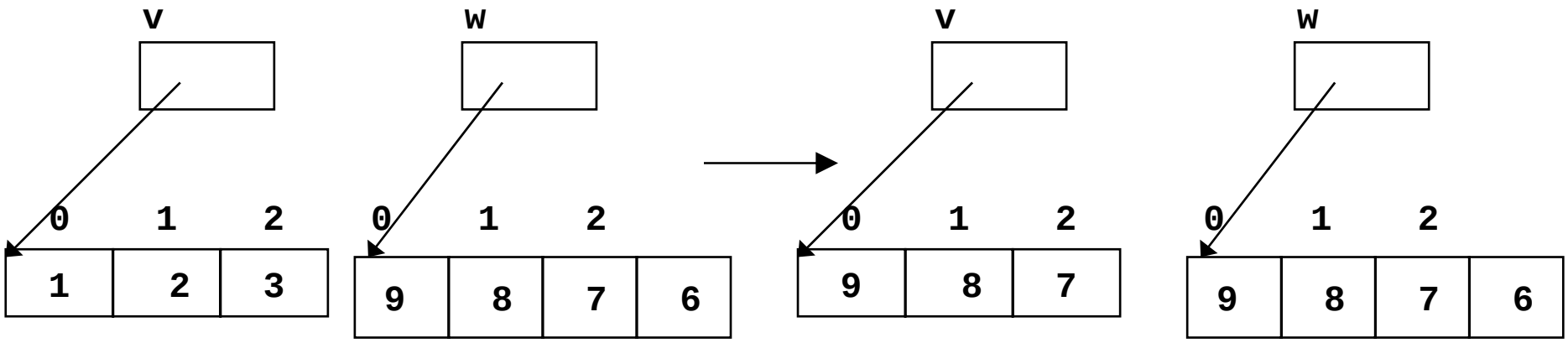
bei der Adress-Zuweisung werden lediglich die Referenzen kopiert, nicht die referenzierten Elemente

Zuweisung (2)

Element-Zuweisung (Wert-Zuweisung):

```
int[] v = {1, 2, 3};
int[] w = {9, 8, 7, 6};
for (int i=0; i<v.length, i++)
  v[i] = w[i]; // Element-Zuweisung
```

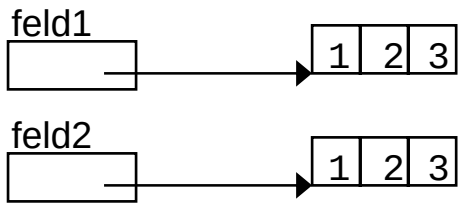
```
v[0] = -77;
w[0] == 9 -> true
```



bei der Element-Zuweisung werden die Werte der referenzierten Elemente kopiert!

Gleichheit

```
int[] feld1 = {1,2,3};
int[] feld2 = {1,2,3};
```



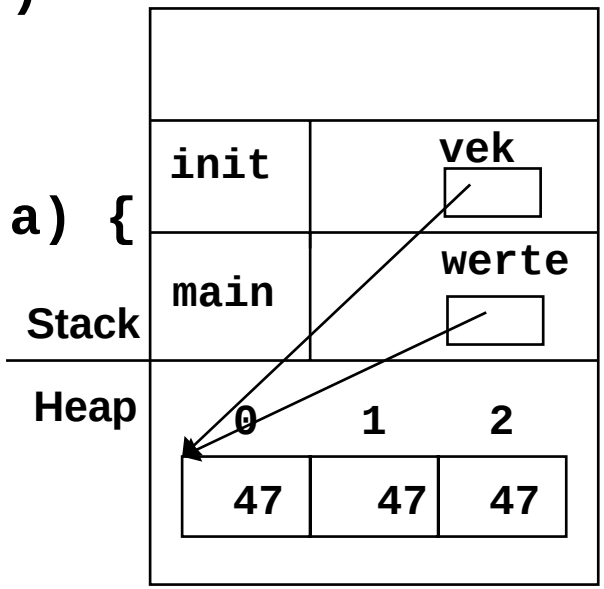

```
feld1 == feld2 -> false!! (Referenzvergleich)
```

```
// Wertevergleich == Elementvergleich
static boolean equals(int[] f1, int[] f2) {
  if (f1 == null || f2 == null) return false;
  if (f1.length != f2.length) return false;
  for (int i=0; i<f1.length; i++) {
    if (f1[i] != f2[i]) return false;
  }
  return true;
}
```

Parameterübergabe

- Sind bei einer Funktionsdefinition Referenzdatentypen als formale Parameter definiert, so werden als aktuelle Parameter die **Referenzen als Wert** übergeben (nicht die Elemente!)
- in einer Funktionen können in diesem Fall die **referenzierten Elemente manipuliert** werden

```
static void init(int[] vek, int value) {
  for (int i=0; i<vek.length; i++)
    vek[i] = value;
}
public static void main(String[] a) {
  int[] werte = new int[3];
  init(werte, 47);
  // werte[i] == 47 -> true
}
```

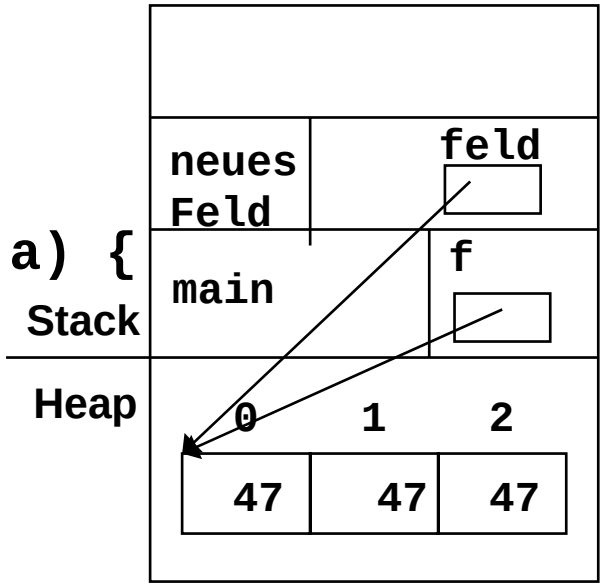


Funktionswerte

- Funktionen können als Funktionswert Werte vom Referenztyp liefern
- Geliefert wird in diesem Fall eine Referenz (Adresse)

```
static int[] neuesFeld(int groesse, int wert) {
  int[] feld = new int[groesse];
  for (int i=0; i<feld.length; i++)
    feld[i] = wert;
  return feld;
}

public static void main(String[] a) {
  int[] f = neuesFeld(3, 47);
}
```



lokale Variablen

- Bez. des Gültigkeitsbereich von Variablen vom Referenztyp gilt dasselbe wie bei "normalen" Variablen
- die referenzierten Elemente werden durch Anwendung des new-Operators lebendig; sie sind solange lebendig, bis sie nicht mehr referenziert werden, also u.U. länger als die Funktion lebendig ist (→ Garbage Collection)

```
static float[] init(int[] vek) {
  float[] fvek = new float[vek.length];
  for (int i=0; i<vek.length; i++)
    fvek[i] = vek[i];
  return fvek;
}
public static void main(String[] a) {
  int[] w1 = {9, 8, 7};
  float[] w2 = init(w1);
    // w2[2] == 7.0F -> true
}
```