Programmierkurs Java

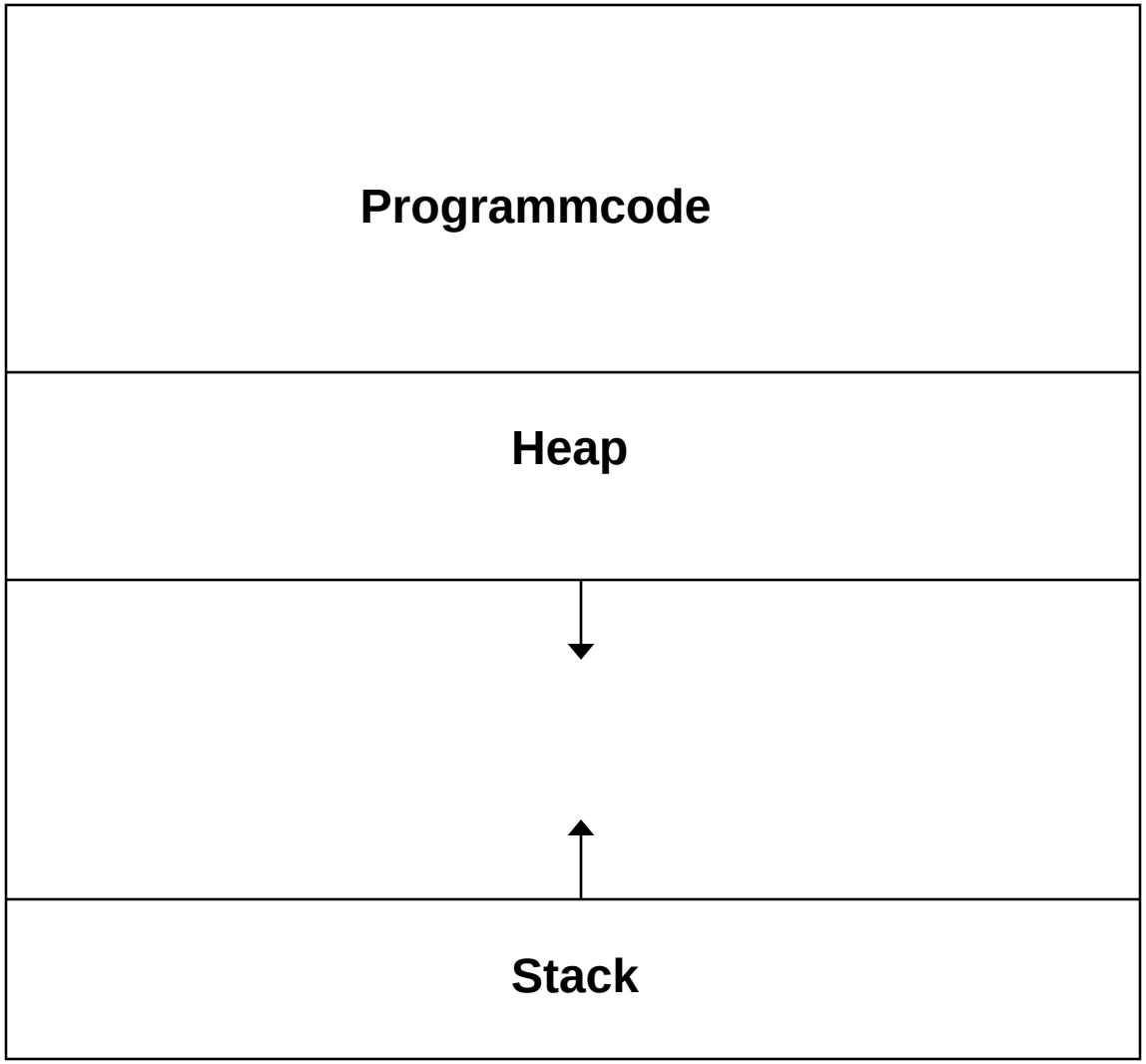
UE 8 - Referenzdatentypen

Dr.-Ing. Dietrich Boles

Gliederung

- Speicherverwaltung
- Schema - Referenzvariablen
- Anmerkungen
- Parameter
- Funktionsrückgabewerte
- Lokale Variablen
- Literale
- Operatoren
- Zuweisung
- Gleichheit
- Zusammenfassung

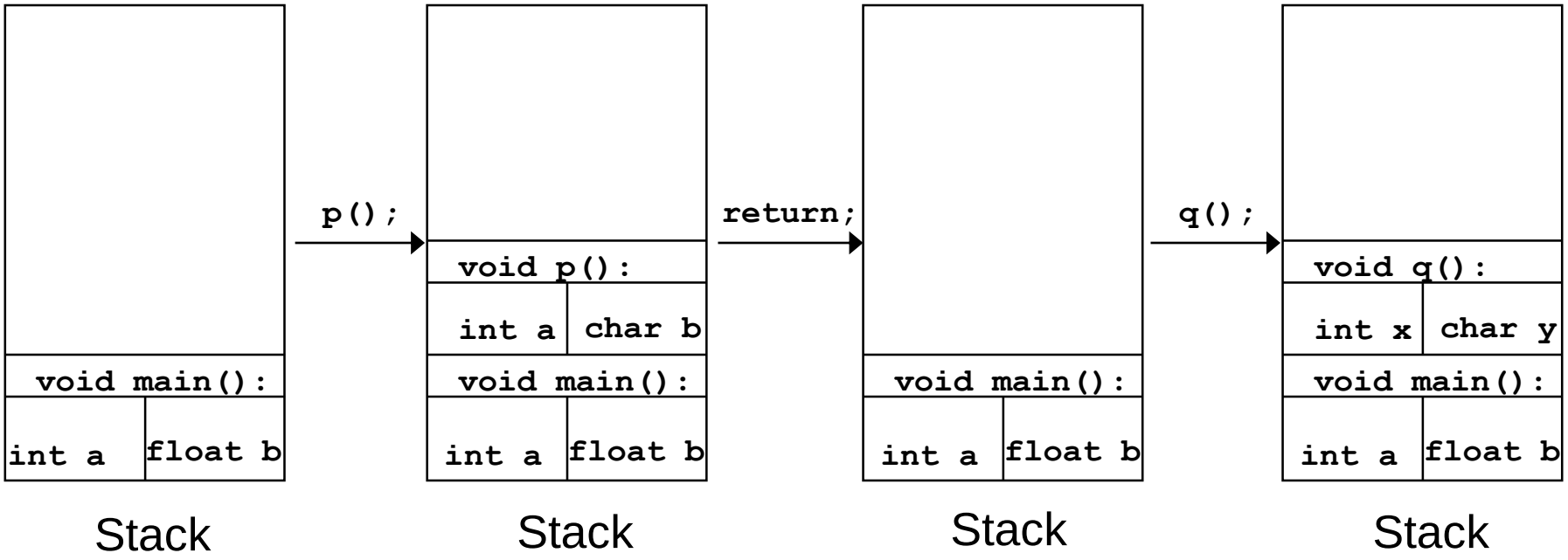
Programm-speicher



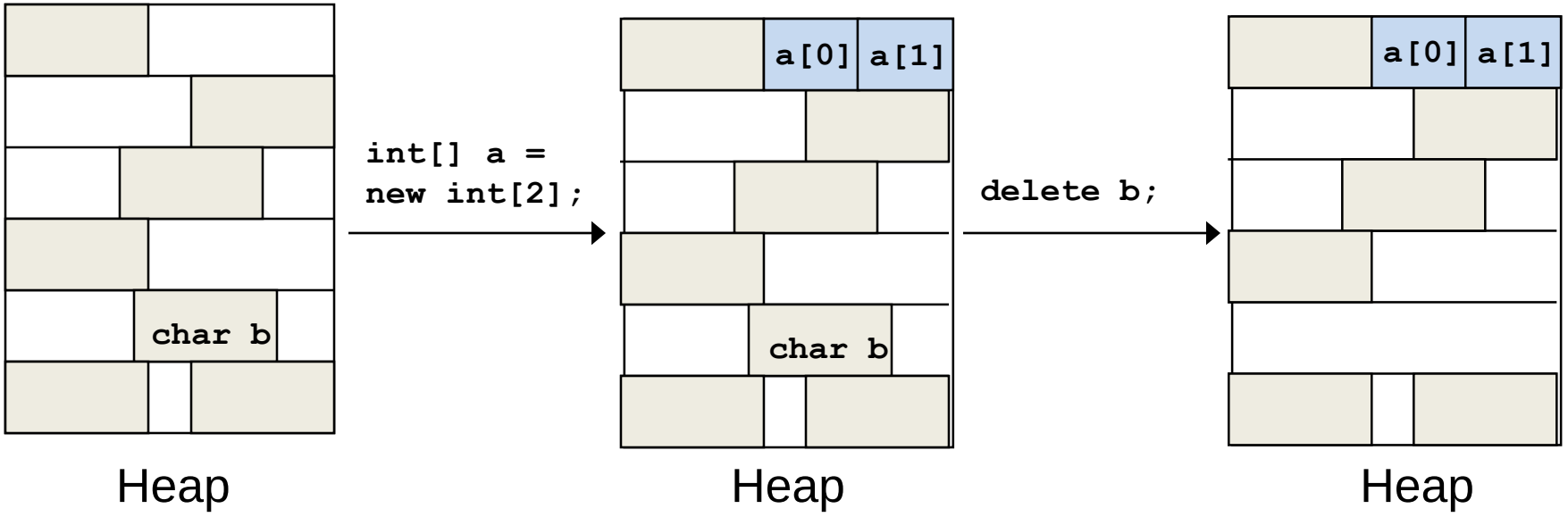
Daten unter Programmkontrolle

lokale Variablen

- Verwaltung durch Laufzeitsystem
- Arbeitet nach dem LIFO-Prinzip "last-in-first-out"
- Verwaltung von Funktionsaktivierungen
- Speicherbereich für lokale Variablen

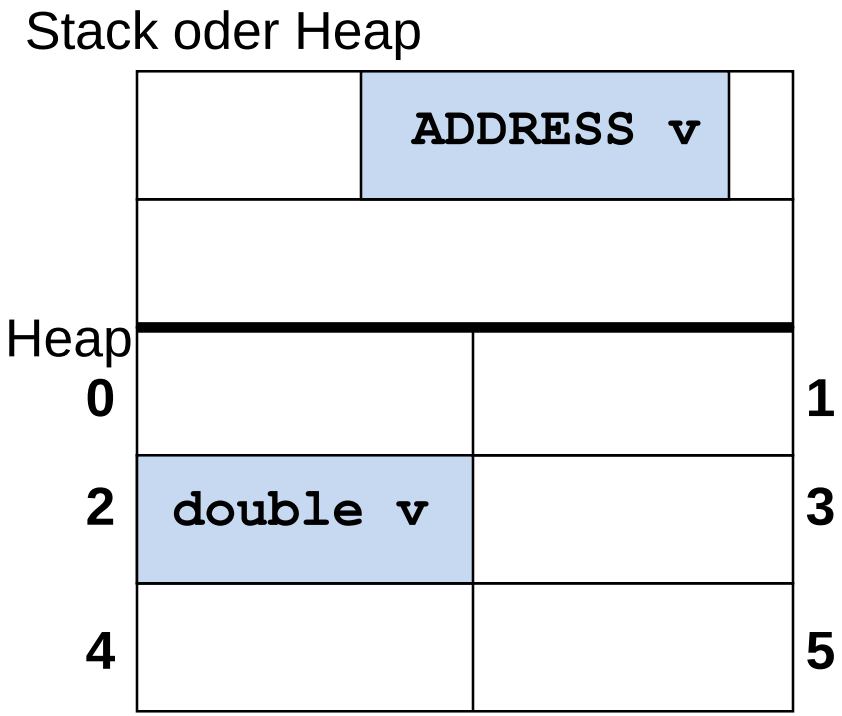


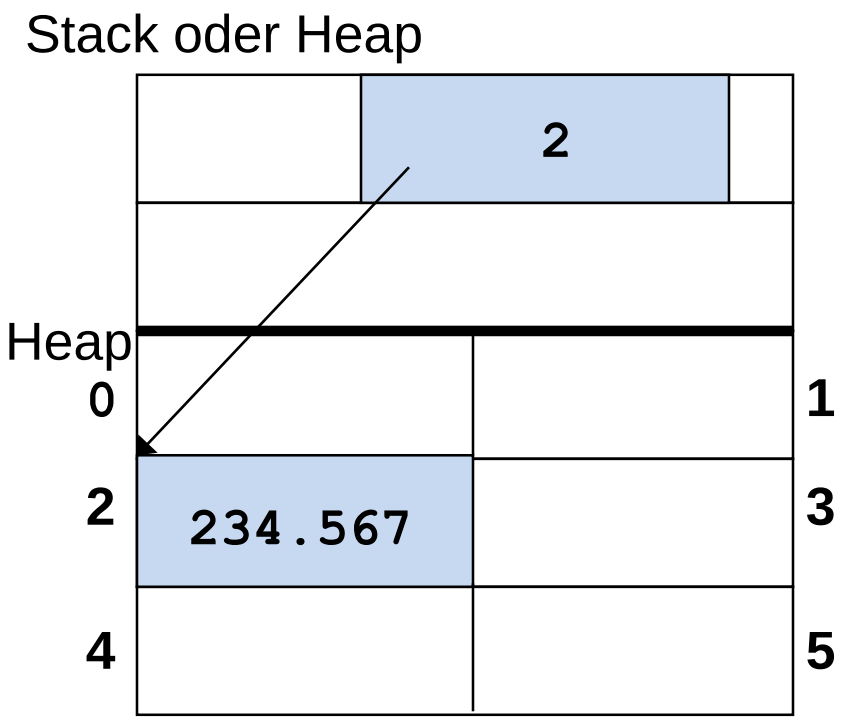
- Programmkontrollierte Verwaltung
- Speicherzuweisung per Anweisung (new/delete)
- Zugriff über "Zeiger-Variablen" (Referenzvariablen)
- Unterstützung durch Laufzeitsystem (Garbage Collection, ...)
- Java: Speicherbereich für Arrays und Objekte



Schema – Referenzvariablen

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





Anmerkungen (1)

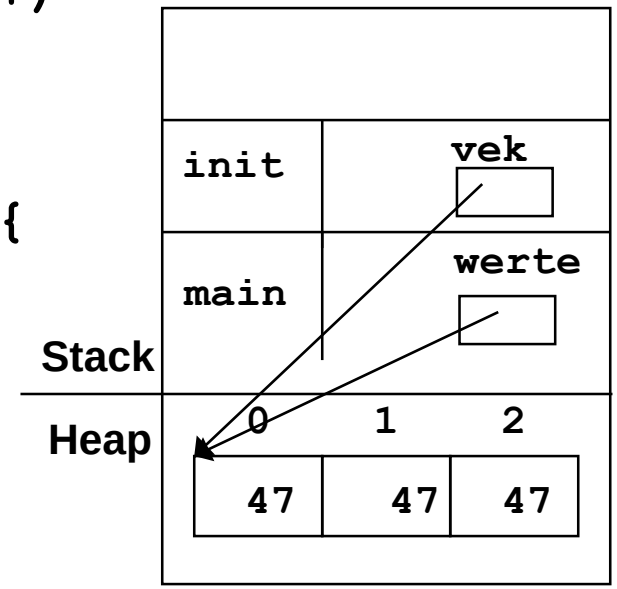
- Motivation
  - es existieren bestimmte Variablen, die lediglich **Adressen** ("**Referenzen**", "Zeiger") auf andere Speicherbereiche speichern können
  - die Kontrolle über die referenzierten Speicherbereiche liegt beim Programmierer bzw. Anwendungsprogramm
  - das Anwendungsprogramm kann zur Laufzeit 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 Arrays und Klassen/Objekte
- Alles was auf den kommenden Folien am Beispiel von Arrays demonstriert wird, kann 1:1 auf Objekte übertragen werden!

Anmerkungen (2)

- es existieren einige Unterschiede zwischen "normalen" Variablen (Variablen von einem Standarddatentyp) und Referenzvariablen (Variablen vom Referenzdatentyp):
  - Parameterübergabe
  - Funktionswert
  - Lokale Variablen
  - Literale
  - Operatoren
  - Zuweisung
  - Gleichheit

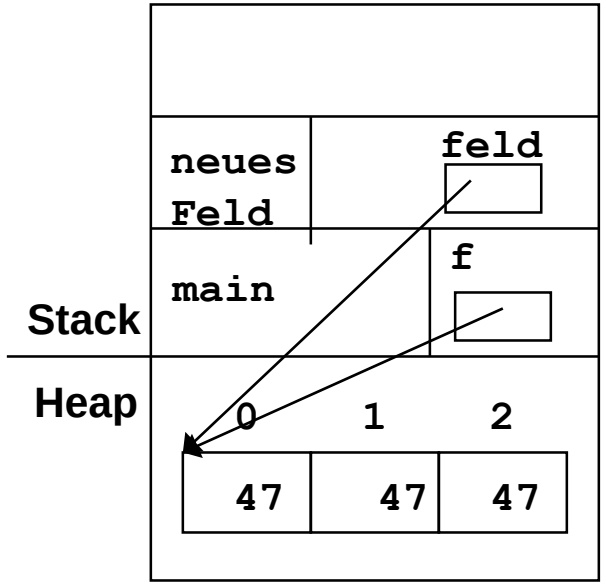
- Sind bei einer Funktionsdefinition Referenzdatentypen als formale Parameter definiert, so werden als aktuelle Parameter die **Referenzen als Wert** übergeben (nicht die Elemente!)
- in einer Funktion können in diesem Fall die **referenzierten Elemente manipuliert** werden

```
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
}
```



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

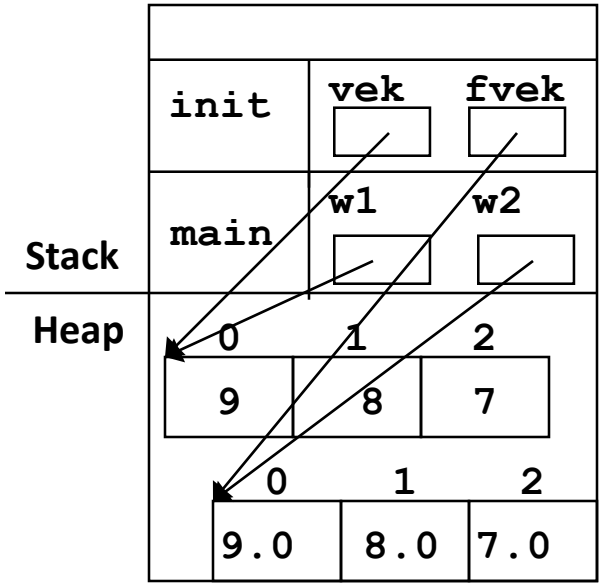
```
static int[] neuesFeld(int groesse, int wert) {
  int[] feld = new int[groesse];
  for (int i = 0; i < feld.length; i++)
    feld[i] = wert;
  return feld;
}

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



Lokale Variablen

- Bez. des Gültigkeitsbereich von Referenzvariablen 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)

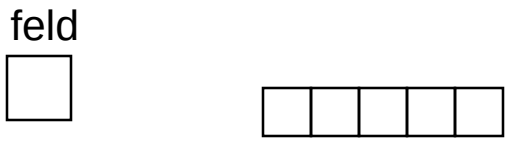
```
static float[] init(int[] vek) {
  float[] fvek = new float[vek.length];
  for (int i = 0; i < vek.length; i++)
    fvek[i] = vek[i];
  return fvek;
}
public static void main(String[] a) {
  int[] w1 = {9, 8, 7};
  float[] w2 = init(w1);
    // w2[2] == 7.0F -> true
}
```



Literale

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

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

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



Garbage-Collector kann aktiv werden!

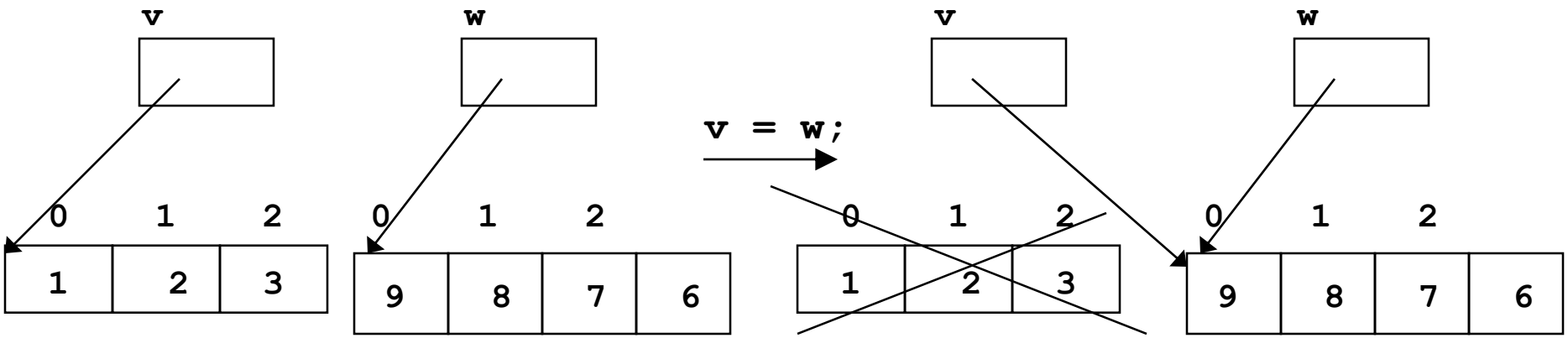
- Zuweisung: **=**
- Gleichheit: **==**
- Ungleichheit: **!=**

```
int[] feld1 = new int[5];
int[] feld2 = new int[6];
...
feld1 = new int[7];
feld2 = feld1;
...
if (feld1 != null)
  feld1[0] = 47;
else if (feld1 == feld2)
  feld2[1] = 67;
```

Adress-Zuweisung:

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

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

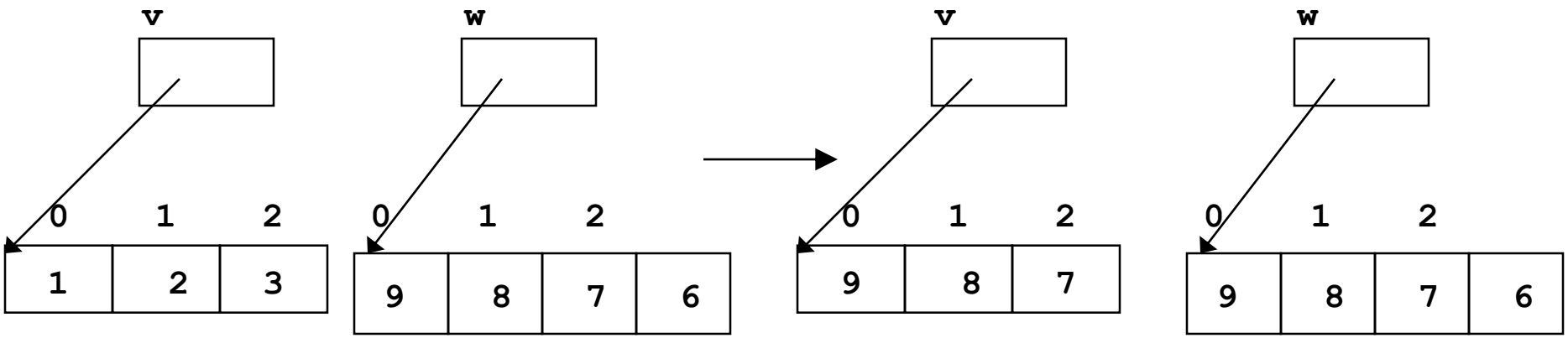


bei der Adress-Zuweisung werden lediglich die Referenzen kopiert, nicht die referenzierten Elemente

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
```

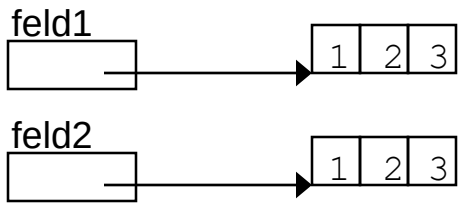
```
v[0] = -77;
w[0] == 9 -> true
```



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;
}
```

Zusammenfassung

- Stack: Speicherbereich für lokale Variablen
- Heap: Speicherbereich für Variablen unter Programmkontrolle (in Java: Referenzdatentypen)

- Referenzdatentypen in Java: Arrays und Objekte
- Referenzvariablen sind Variablen, die Referenzen auf andere Variablen (Arrays, Objekte) speichern können

- Sind bei einer Funktionsdefinition Referenzdatentypen als formale Parameter definiert, so werden als aktuelle Parameter die Referenzen als Wert übergeben; in einer Funktion können in diesem Fall die referenzierten Variablen manipuliert werden