Hochschule Fulda
University of Applied Sciences

Programmierung 2

Vorlesung 3: Hinter den Kulissen I

Alexander Gepperth, April 2025

Variablen in Java

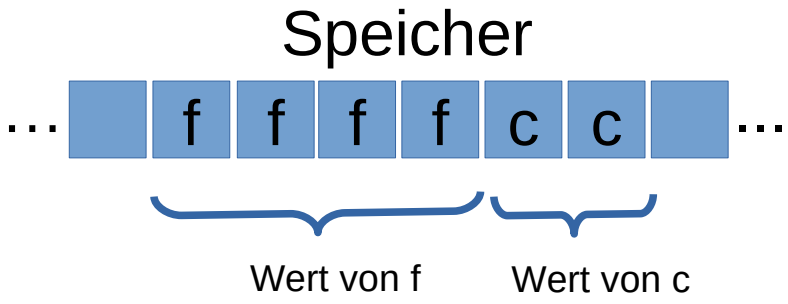
Ein Wort zu Variablen

- eine Variable ist für den Compiler ein Name für einen Speicher**bereich**
  - Startadresse
  - Länge
- Deklaration der Variable **reserviert** den Speicherbereich (auf Stack, Heap oder Method Area)
- Speicherbereich: Größe hängt von Datentyp ab

Ein Wort zu Variablen

- Größe von Variablen im Speicher in C

```
float f ;   // 4 Byte
int i ;     // 4 Byte
char c ;    // 2 Byte
char * s ;  // 8 Byte
int * k  ;  // 8 Byte
```



- warum 8 Byte für `char *` und `int *` ?
  → 2 Arten von Variablen!

Demo

- Verhalten von Variablen in C

Java: primitive Datentypen

Java: primitive Datentypen

- Name für Speicherbereich (Start, Länge)
- In diesem Bereich ist ein **Wert** gespeichert
- Beispiele: `char, int, float, double, ...`

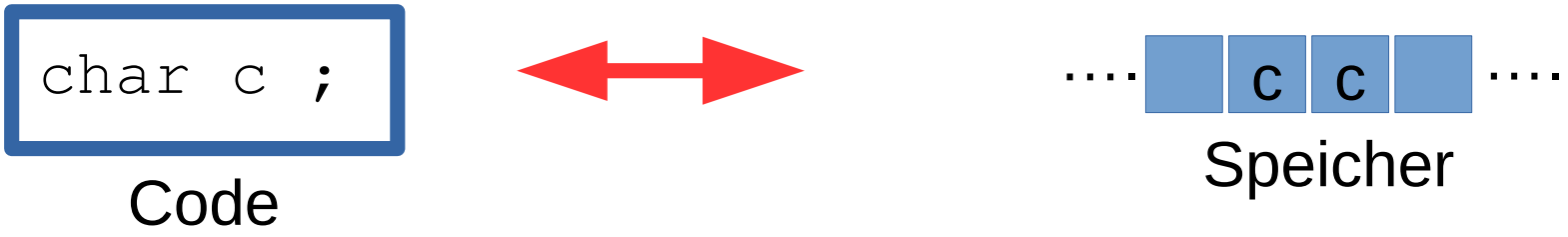
```
char c ;
```

Code

- keine primitiven Typen: `int[], float[], double[]`

Java: primitive Datentypen

- Name für Speicherbereich (Start, Länge)
- In diesem Bereich ist ein **Wert** gespeichert
- Beispiele: `char, int, float, double, ...`



- keine primitiven Typen: `int[], float[], double[]`

Java: primitive Datentypen

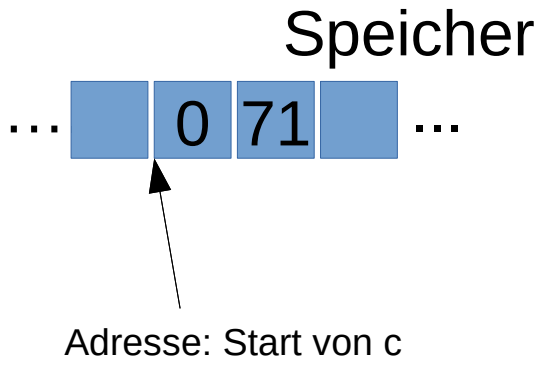
- bei primitiven Datentypen genügt es, sie zu deklarieren um sie zu benutzen
  - Speicherbereich wird durch Deklaration reserviert
  - kann jetzt gelesen und beschrieben werden

```
char c ;
c = '1' ;
if (c == '1') ...
```

Java: primitive Datentypen

- was passiert wenn ein primitiver Datentyp beschrieben wird?

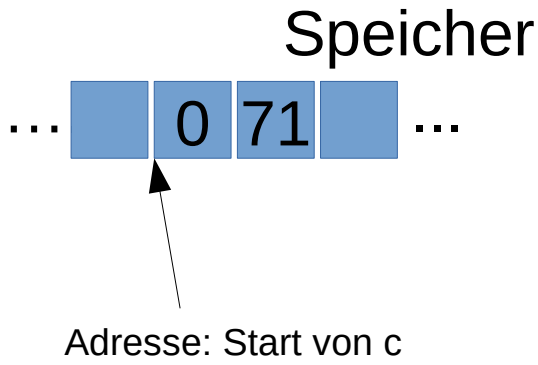
```
char c = '1' ;
c = '5' ;
```



Java: primitive Datentypen

- was passiert wenn ein primitiver Datentyp beschrieben wird?
  - Adresse wird nachgeschlagen

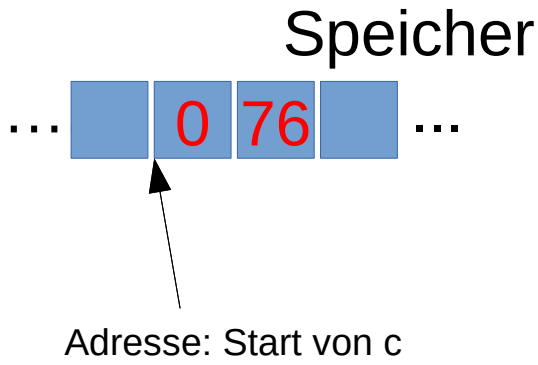
```
char c = '1' ;
c = '5' ;
```



Java: primitive Datentypen

- was passiert wenn ein primitiver Datentyp beschrieben wird?
  - Adresse wird nachgeschlagen
  - Speicherbereich wird geändert

```
char c = '1' ;
c = '5' ;
```



Java: Referenzdatentypen

Java: Referenzdatentypen

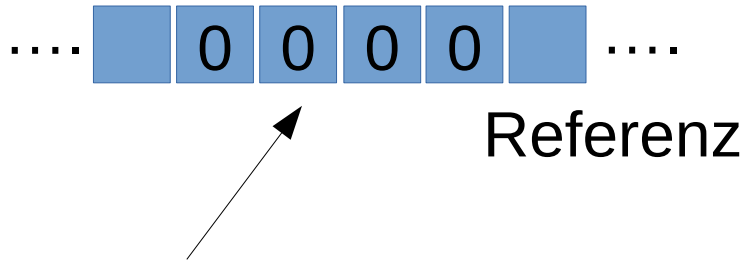
- Ebenfalls: Name für Speicherbereich (Start, Länge)
- dort ist **eine Referenz** gespeichert, kein Wert!
- **Referenz**: Speicheradresse des Werts (woanders im Speicher)

```
char[] c ;
```

Java: Referenzdatentypen

- Ebenfalls: Name für Speicherbereich (Start, Länge)
- dort ist **eine Referenz** gespeichert, kein Wert!
- **Referenz**: Speicheradresse des Werts (woanders im Speicher)

```
char[] c ;
```



Ref. werden auf `null` initialisiert!

Java: Referenzdatentypen

- Ebenfalls: Name für Speicherbereich (Start, Länge)
- dort ist **eine Referenz** gespeichert, kein Wert!
- **Referenz**: Speicheradresse des Werts (woanders im Speicher)

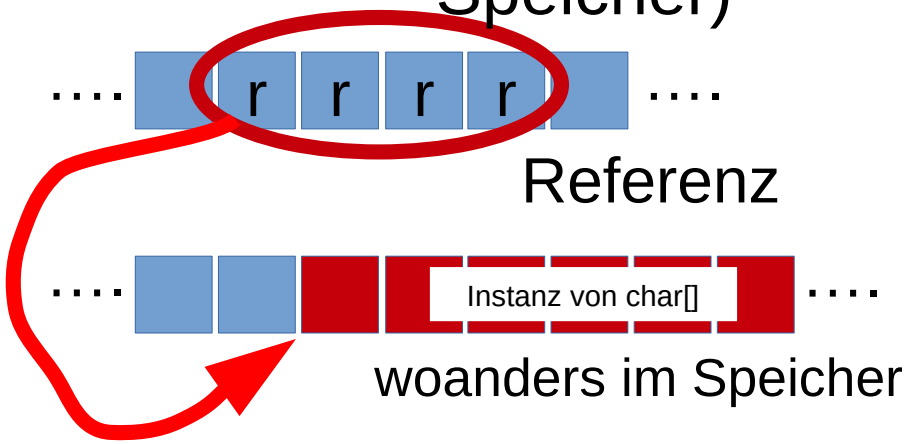
```
char[] c = new char[5];
```

Java: Referenzdatentypen

- Ebenfalls: Name für Speicherbereich (Start, Länge)
- dort ist **eine Referenz** gespeichert, kein Wert!
- **Referenz**: Speicheradresse des Werts (woanders im Speicher)

```
char[] c = new char[5];
```

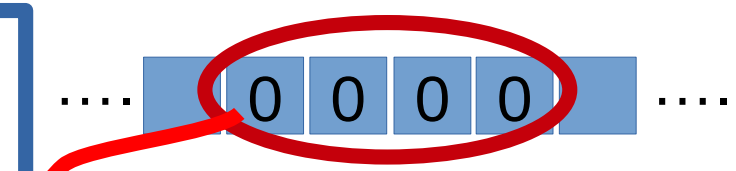
Referenzen sind Pfeile („wo kann ich den Wert finden?“)



Java: Referenzdatentypen

- Deklaration der Referenzvariable reserviert Speicher..
- ... aber nur für die Referenz (Anfangswert `null`) ...

```
char[] c ;
c[3] = '0' ;
```



zeigt “nirgendwo” hin

Java: Referenzdatentypen

- Deklaration der Referenzvariable reserviert Speicher..
- ... aber nur für die Referenz (Anfangswert `null`) ...
- ... nicht für den Wert auf den die Referenz zeigen soll!

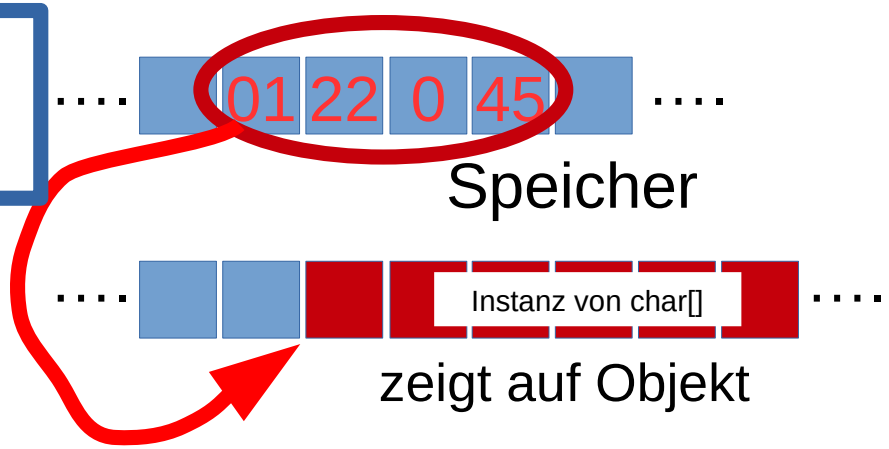
```
char[] c ;
c[3] = '0' ;
```

Funktioniert nicht:
NullPointerException !

Java: Referenzdatentypen

- Deklaration der Referenzvariable reserviert Speicher..
- .. aber nur für die Referenz selbst ..
- .. nicht für das Objekt auf das die Referenz zeigen soll!

```
char[] c = new char [5];
c[3] = '0' ;
```



new reserviert Speicher für das char[]-Objekt und gibt die Adresse zurück!

Java: Referenzdatentypen

- Was passiert bei folgendem Code?

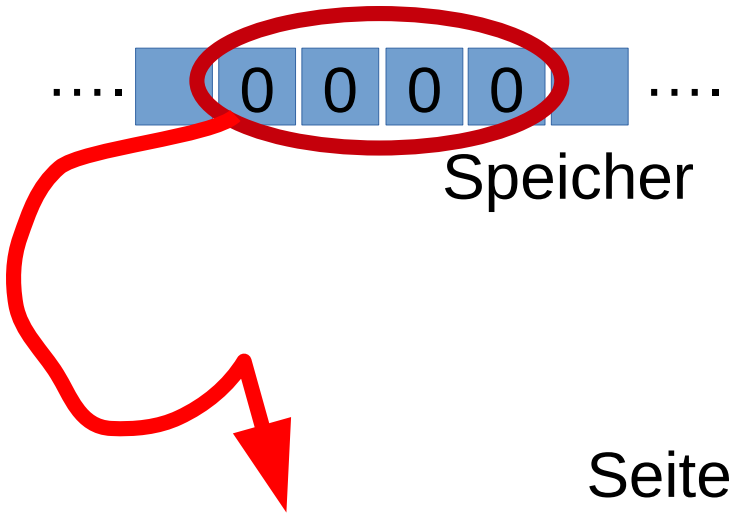
```
char[] c = new char [5];
c[1] = '0' ;
```

Java: Referenzdatentypen

- Was passiert bei folgendem Code?

```
char[] c = new char [5];
c[1] = '0' ;
```

1) Speicher für Referenz reservieren, zeigt nirgendwo hin

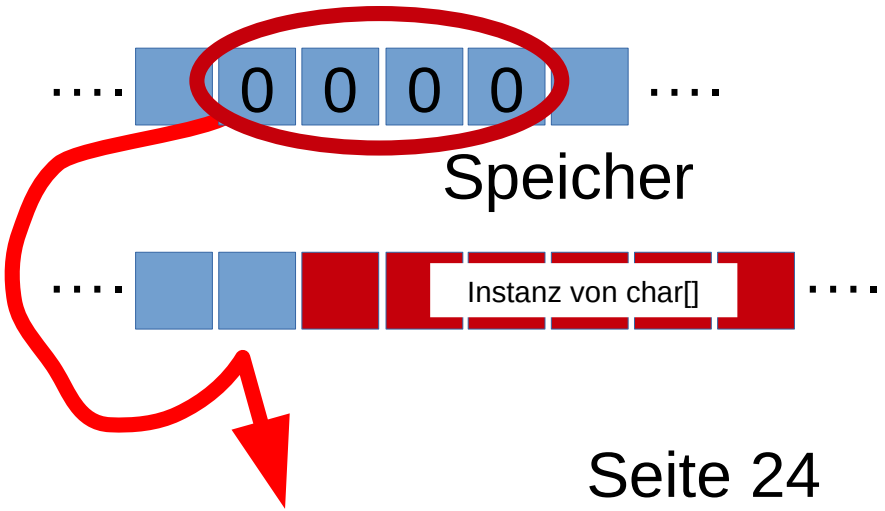


Java: Referenzdatentypen

- Was passiert bei folgendem Code?

```
char[] c = new char [5];
c[1] = '0' ;
```

1) Speicher für Referenz reservieren, zeigt nirgendwo hin
2) Speicher für Objekt reservieren

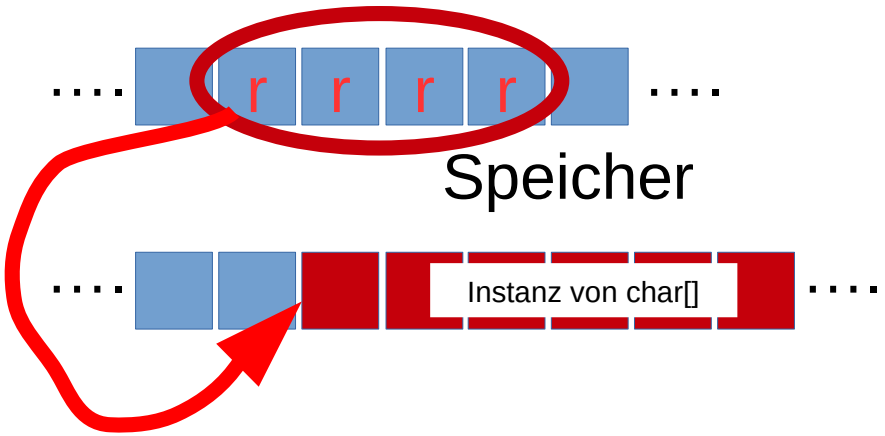


Java: Referenzdatentypen

- Was passiert bei folgendem Code?

```
char[] c = new char [5];
c[1] = '0' ;
```

1) Speicher für Referenz reservieren, zeigt nirgendwo hin
2) Speicher für Objekt reservieren
3) Pfeil/Referenz “umbiegen”

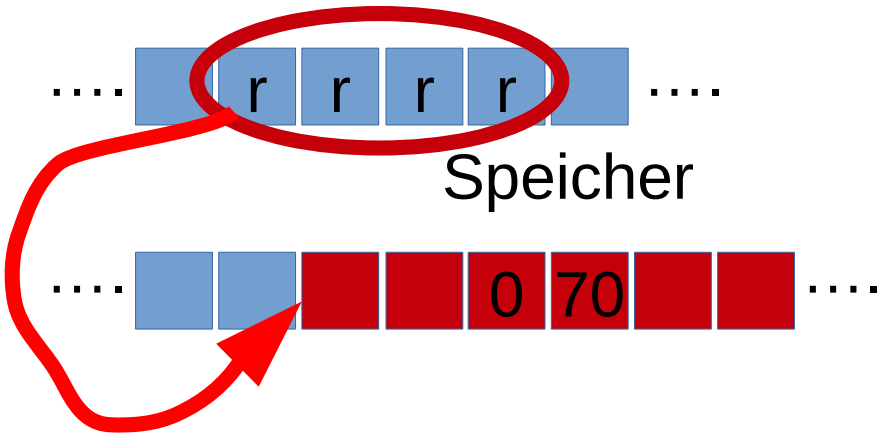


Java: Referenzdatentypen

- Was passiert bei folgendem Code?

```
char[] c = new char [5];
c[1] = '0' ;
```

1) Speicher für Referenz reservieren, zeigt nirgendwo hin
2) Speicher für Objekt reservieren
3) Pfeil/Referenz “umbiegen”
4) Wert schreiben



Fragerunde

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**

```
char[] v1 = new char [5];
v1[0] = 'c' ;
char[] v2 ;
v2 = v1 ;
System.out.println(v2[0]);
```

?

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**

```
char[] v1 = new char [5];
v1[0] = 'c' ;
char[] v2 ;
v2 = v1 ;
System.out.println(v2[0]);
```

Ausgabe:
c

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**

```
char[] v1 = new char [5];
v1[0] = 'c' ;
char[] v2 ;
v2 = v1 ;
v2[0] = 'X' ;
System.out.println(v1[0]);
```

?

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**
- Über den verbogenen Pfeil alle Operationen möglich!

```
char[] v1 = new char [5];
v1[0] = 'c' ;
char[] v2 ;
v2 = v1 ;
v2[0] = 'X' ;
System.out.println(v1[0]);
```

Ausgabe:
X

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**

```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```

?

Referenzen sind Pfeile

- Wichtig: Unterscheidung zwischen Referenzvariable und dem Objekt auf das sie zeigt
- Zuweisungen an Referenzvariable **verbiegen den Pfeil!**

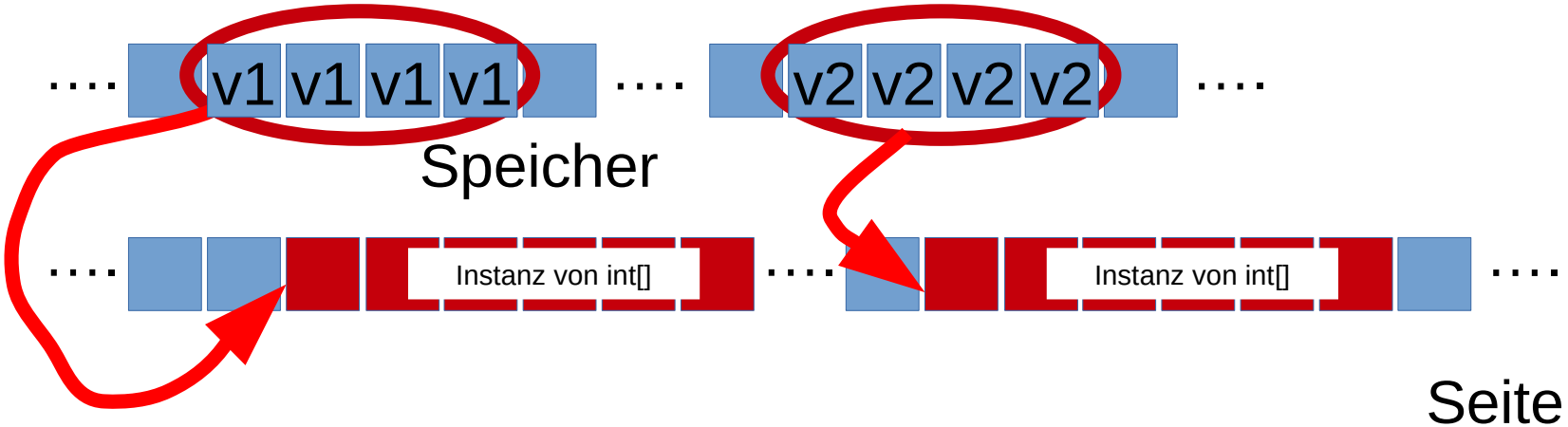
```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```

Ausgabe:
1

Referenzen sind Pfeile

- Grafische Darstellung (vor `v1 = v2`)

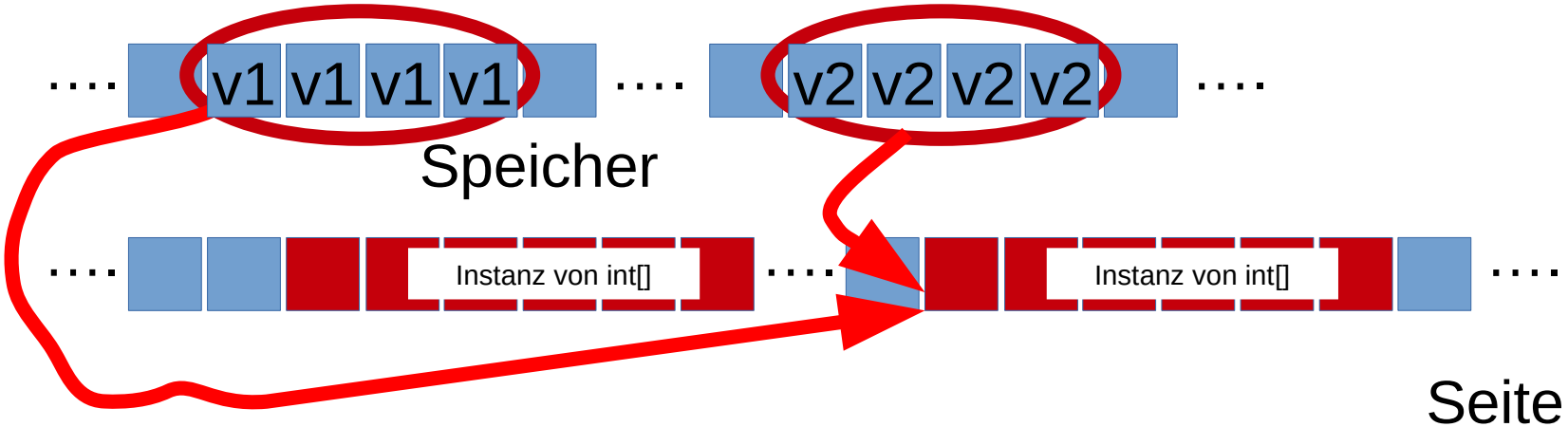
```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```



Referenzen sind Pfeile

- Grafische Darstellung (nach `v1 = v2`)
- klar dass Operationen auf A auch B ändern!

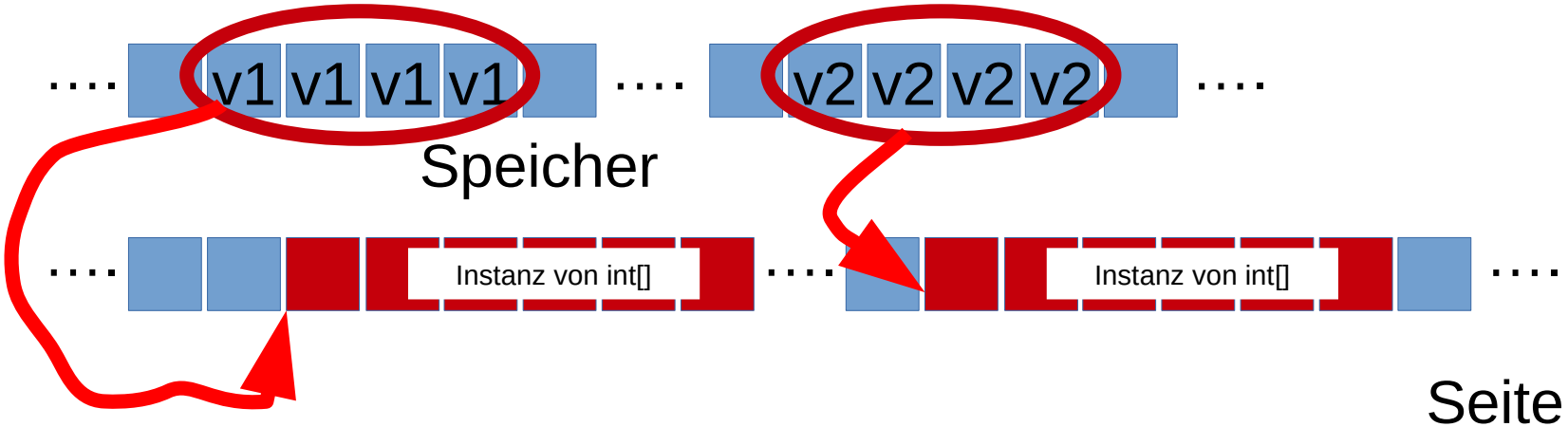
```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```



Garbage Collection

- Durch das Verbiegen von Referenzen können Objekte “verlorengehen”
- Objekte ohne Referenz werden **automatisch** gelöscht: **Garbage Collection**

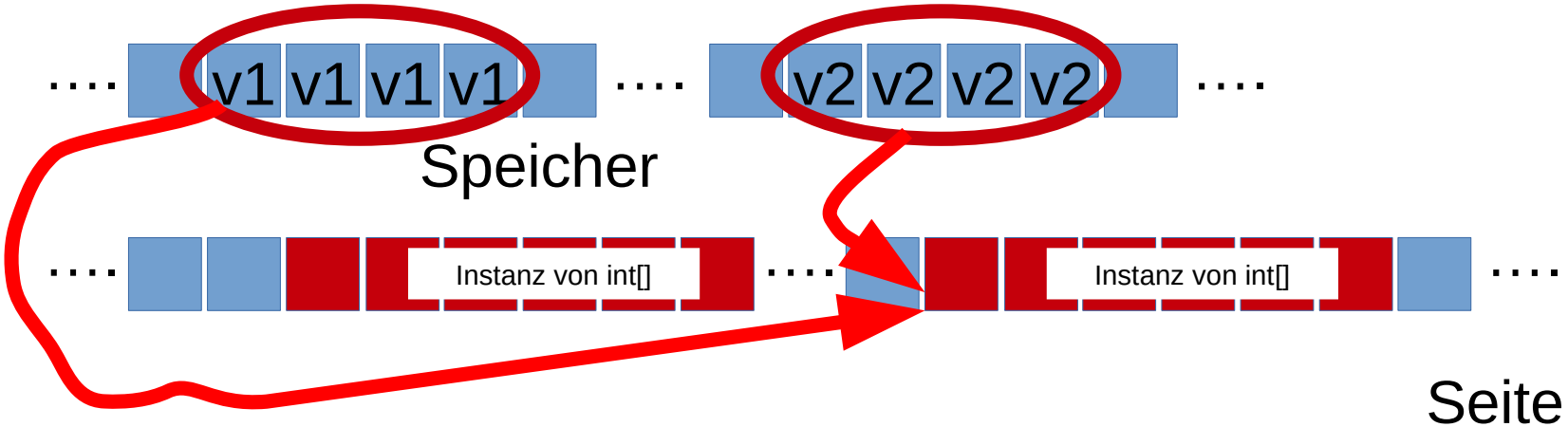
```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```



Garbage Collection

- Durch das Verbiegen von Referenzen können Objekte “verlorengehen”
- Objekte ohne Referenz werden **automatisch** gelöscht: **Garbage Collection**

```
int[] v1 = new int [5];
v1[0] = 1000 ;
int[] v2 = {1,2,3,4} ;
v1 = v2 ;
System.out.println(v1[0]);
```



Zusammenfassung

- Jede Variable in Java ist entweder Referenz oder primitiv
- In beiden Fällen sind Variablen Namen für Speicherbereiche
  - primitiver Datentyp: Speicherbereich repräsentiert Wert
  - Referenzdatentyp: Speicherbereich repräsentiert Adresse eines Werts(“Pfeil”)

Zusammenfassung

- Referenzdatentypen sind:
  - Arrays primitiver Typen: `char[], float[], int[], …`
  - Klassen:
    - Vordefinierte Klassen: `java.util.Scanner, String, ...`
    - selbst definierte Klassen: `Playground, GameObject, GameLoop, ...`
- Primitive Datentypen sind: `int, float, char, double, byte, long, ...`

Zusammenfassung

- Primitive Datentypen sind einfach zu handhaben
- Referenzdatentypen sind mächtig, erfordern allerdings Verständnis vor allem bei
  - Zuweisungen (bereits behandelt)
  - Testen auf Gleichheit
  - Parameterübergabe an Methoden

CUT: Q&A