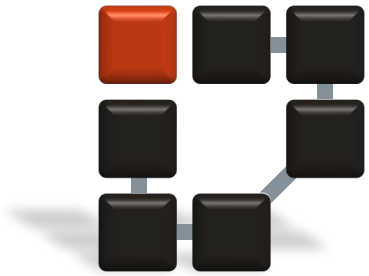
Informatik 1 für Nebenfachstudierende Grundmodul

Java – Referenzdatentypen genauer betrachtet

Kai-Steffen Hielscher
Folienversion: 14. Januar 2020



Informatik 7
Rechnernetze und
Kommunikationssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

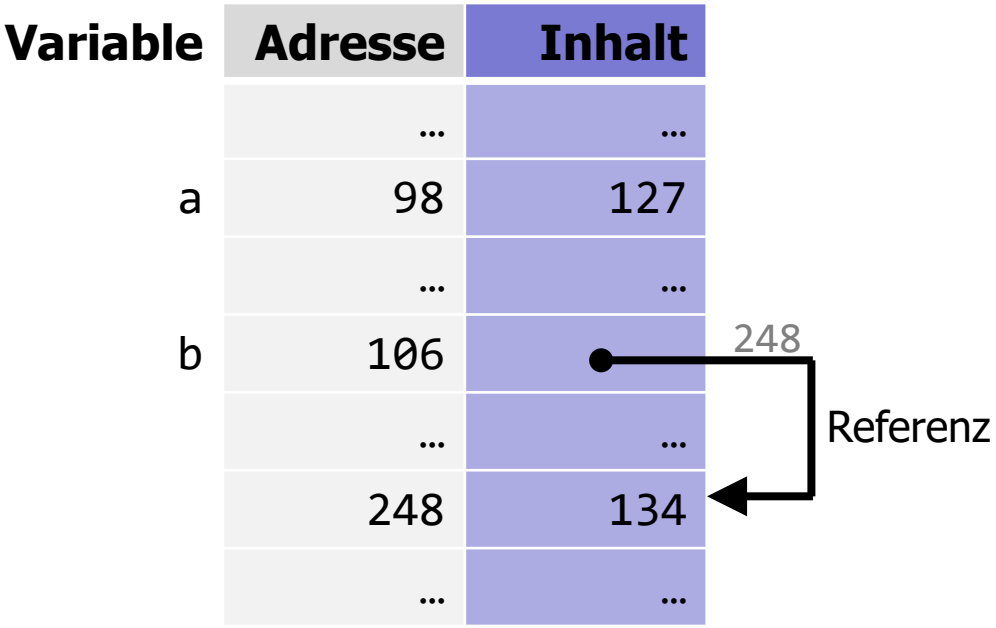
Referenzdatentypen

- eine Variable für einen **primitiven Datentyp** verweist auf einen Bereich im Hauptspeicher, der **direkt den Wert der Variable** enthält
- eine Variable für einen **Referenzdatentypen** verweist auf einen Bereich im Speicher, der die **Adresse des referenzierten Objekts** enthält

Referenzdatentypen

- ```
  byte a = 127;
  Byte b = new Byte(134);
  ```

- Hauptspeicher des Computers


  ```

Referenzdatentypen

- Deklaration wie in Java bei allen Datentypen üblich
  - `Byte b;`
- Initialisierung mit dem **`new`**-Operator
  - `b = new Byte(134);`
  - reserviert sein Stück Speicher für den eigentlichen Inhalt
  - legt den initialen Wert in dem Stück Speicher ab
- oder Deklaration und Initialisierung ein einem Schritt
  - `Byte b = new Byte(134);`

Referenzdatentypen

- Referenzdatentypen
  - siehe Kapitel *Java - Datentypen*
  - Objekte in Java
    - vordefinierte Objekte
    - selbst definierte Objekte
    - Wrapper-Klassen für primitive Datentypen
    - Strings
    - Arrays

Arrays (Felder)

- Ein Array ist eine Folge von Variablen mit gleichem Typ
  - elementare Datentypen
  - Instanzen einer Klasse
- feste Länge
- Deklaration

  ```
  Datentyp[] Arraybezeichner;
  ```

  - z.B.

    ```
    int[] array;
    ```
- hierdurch wird eine **Referenz** auf das Array angelegt
- danach muss das eigentliche Array mit `new` erzeugt werden, dabei wird die Anzahl der Elemente angegeben

  ```
  Arraybezeichner = new Datentyp[Elemente];
  ```

  - z.B.

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

Arrays

- meist in einem Schritt
  - z.B.
    ```
    int[] array = new int[10];
    ```
- Jetzt wird auch klar, warum hier eine Referenz benötigt wird: Vor Erzeugung des Arrays mit `new` ist nicht klar, wie groß das Array sein wird, d.h. wie viel Speicher dafür benötigt wird

Arrays

- Deklaration, Initialisierung des Speicherbereichs und Initialisierung der Werte im Array in einem Schritt möglich, dann kein `new` nötig
- Größe des Speicherbereichs gleich beim Anlegen der Referenz bei der Deklaration bekannt
- Werte des zu erstellenden Arrays in geschweiften Klammern
  - z.B.
    ```
    int[] feld = {0, 1, 2, 3, 4};
    ```

Arrays

- Länge eines Arrays
  Arraybezeichner`.length`
  liefert die Anzahl der Elemente eines Arrays
  - ein Array ist ein Objekt
  - `length` ist eine Komponentenvariable
- Beispiel

```
int[] zahlen = new int[24];
System.out.println(zahlen.length); // 24
```

Arrays

- Zugriff auf Element i mittels *array[i]*
  - *i* heißt **Index**
  - **Achtung: die Nummerierung der Elemente beginnt bei 0.**
  - im Beispiel

    ```
    int[] zahlen = new int[24];
    ```

    kann *i* Werte von 0 bis 23 annehmen

  - z.B.

    ```
    zahlen[0] = 1;
    zahlen[5] = 4711;
    zahlen[9] = 42;
    ```

Arrays

- auch mehrdimensionale Arrays möglich
- eigentlich ist dies ein Array, das wieder Array enthält
- z.B.

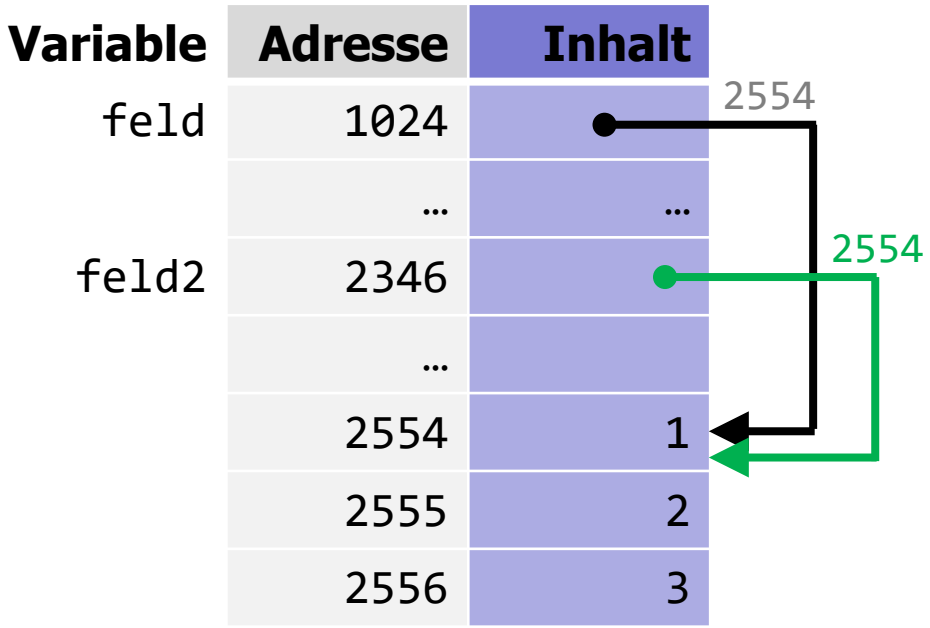
```
double[][] matrix = new double[3][3];
matrix[0][0] = 1.0;
matrix[0][1] = 0.0;
matrix[0][2] = 0.0;
matrix[1][0] = 0.0;
matrix[1][1] = 1.0;
matrix[1][2] = 0.0;
matrix[2][0] = 0.0;
matrix[2][1] = 0.0;
matrix[2][2] = 1.0;
```

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Arrays

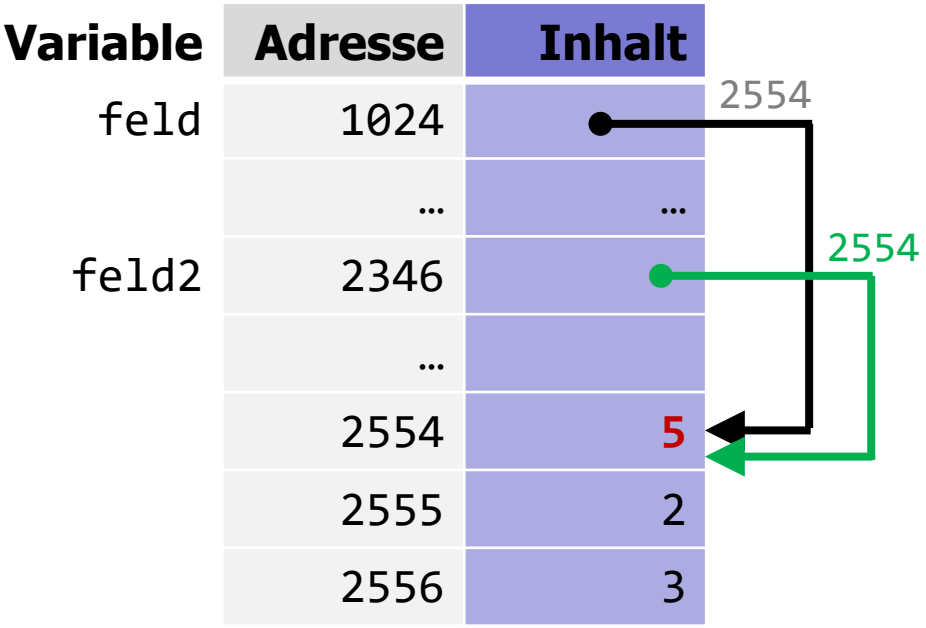
- Zuweisungen erzeugen nur eine neue Referenz, keine Kopie der Elemente des Arrays
  - z.B.

```
byte[] feld = {1, 2, 3};
byte[] feld2 = feld;
```



Arrays

- Änderung des Speicherinhalts wirkt sich auf beide Arrays aus
  - z.B.
    ```
    feld2[0] = 5;
    ```
  - im Beispiel verweisen dann sowohl `feld` als auch `feld2` auf die gleichen Elemente, sowohl `feld2[0]` als auch `feld[0]` hat nun den Wert `5`.



Arrays

- Kopie eines Arrays anlegen
  - mittels vordefinierter Methode

```
System.arraycopy(quelle, quellstartindex,
                 ziel, zielstartindex, anzahl);
```

    - z.B.:

```
int[] array = new int[10];
array[0] = 422;
array[1] = array[0] – 27;
…
int[] kopie = new int[array.length];

System.arraycopy(array, 0, kopie, 0, array.length);
```

Arrays

- Kopie eines Arrays anlegen
  - manuell
    - Schleife, um die einzelnen Elemente zu kopieren
    - **Vorsicht:** Index beginnt bei 0, endet bei *array*`.length-1`
    - z.B.:

```
int[] array = new int[10];
array[0] = 422;
array[1] = array[0] – 27;
…
int[] kopie = new int[array.length];

for(int i = 0; i < array.length; i++) {
    kopie[i] = array[i];
}
```

Arrays

- Kopie eines Arrays anlegen
  - Vorsicht bei mehrdimensionalen Arrays
    - ein mehrdimensionales Array ist ein Array von Arrays
    - bei zwei Dimensionen zwei geschachtelte Schleifen nötig
    - z.B.

```
double[][] matrix = new double[3][3];
double[][] kopie  = new double[3][3];

matrix[0][0] = 1.0;
…

for(int i = 0; i < 3; i++) {
  for(int j = 0; j < 3; j++) {
    kopie[i][j] = matrix[i][j];
  }
}
```

Arrays

- Schleifen eignen sich gut zur Bearbeitung von Arrays
  - Iteration über Elemente des Arrays
  - Beispiel mit for-Schleifen:

```
int[] array = new int[10];
int summe = 0;

for(int i = 0; i < array.length; i++) {
    array[i] = 2 * (i + 1);
}

for(int i = 0; i < array.length; i++) {
    summe += array[i];
}

System.out.println(summe);
```

Arrays

- Schleifen eignen sich gut zur Bearbeitung von Arrays
  - seit Java 5 gibt es eine vereinfachte for-Schleifen-Notation
    `for ( typ variablenname : ausdruck )`
  - iteriert über die Elemente eines Arrays, vermeidet explizite Angabe eines Index und der Grenzen für den Index
  - Beispiel:

```
int[] array = {2, 4, 6, 8, 10};
int summe = 0;

for(int x : array) {
    summe += x;
}
```

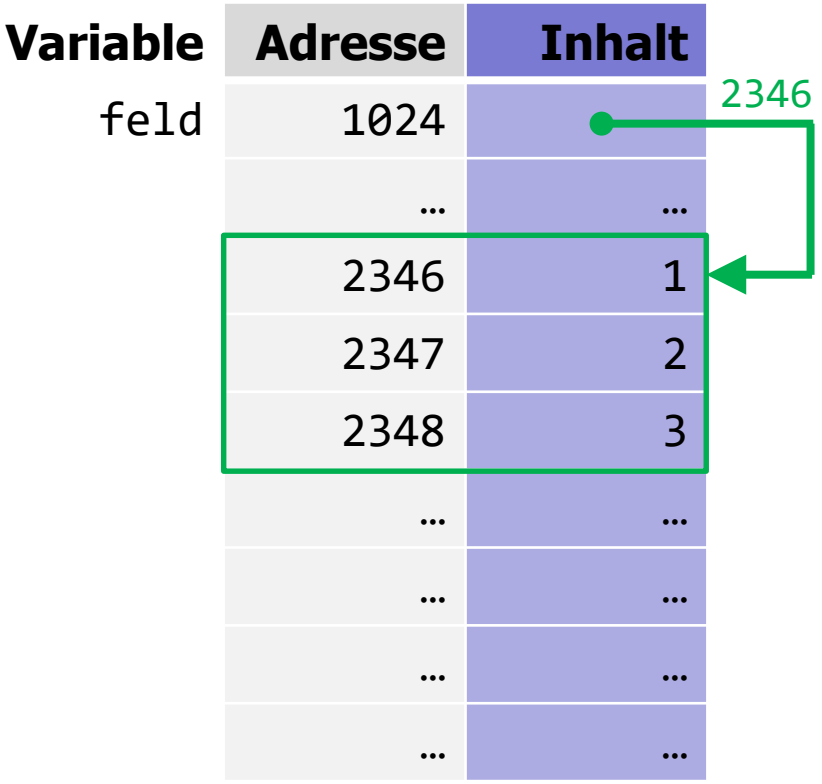
Referenzdatentypen

- bei Referenzdatentypen müssen die Objekte mit `new` erzeugt werden (bei Strings nicht explizit)
- dabei wird ein Speicherbereich für diese Objekte reserviert
- **Garbage Collector** von Java gibt den Speicher automatisch frei, wenn er nicht mehr benötigt wird
  - wenn keine Referenz mehr auf diesen Speicherbereich verweist

Arrays

- Garbage Collector
  - z.B.

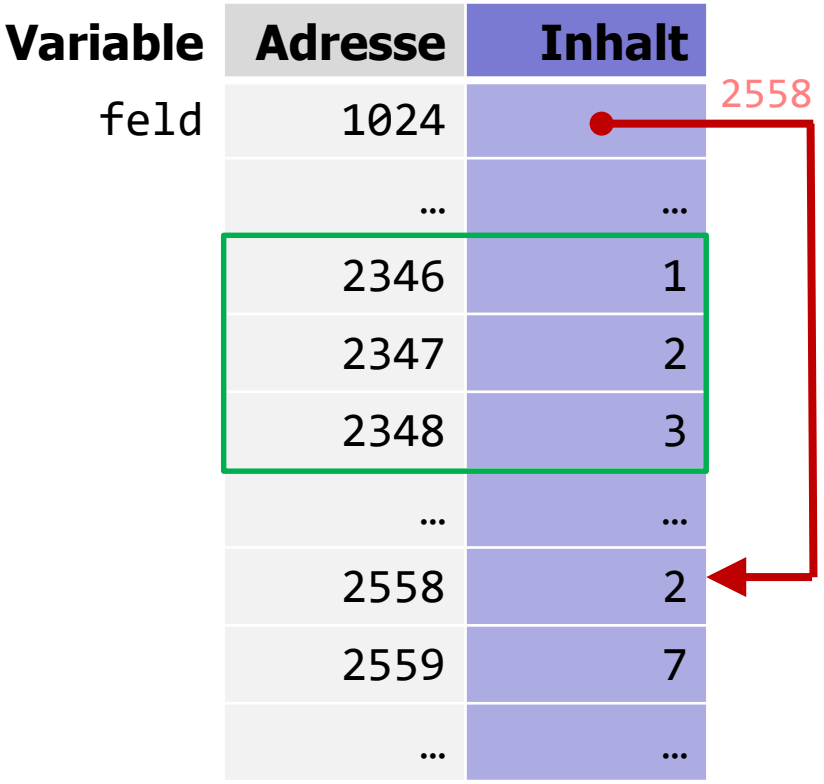
```
byte[] feld = {1, 2, 3};
```



Arrays

- Garbage Collector
  - z.B.

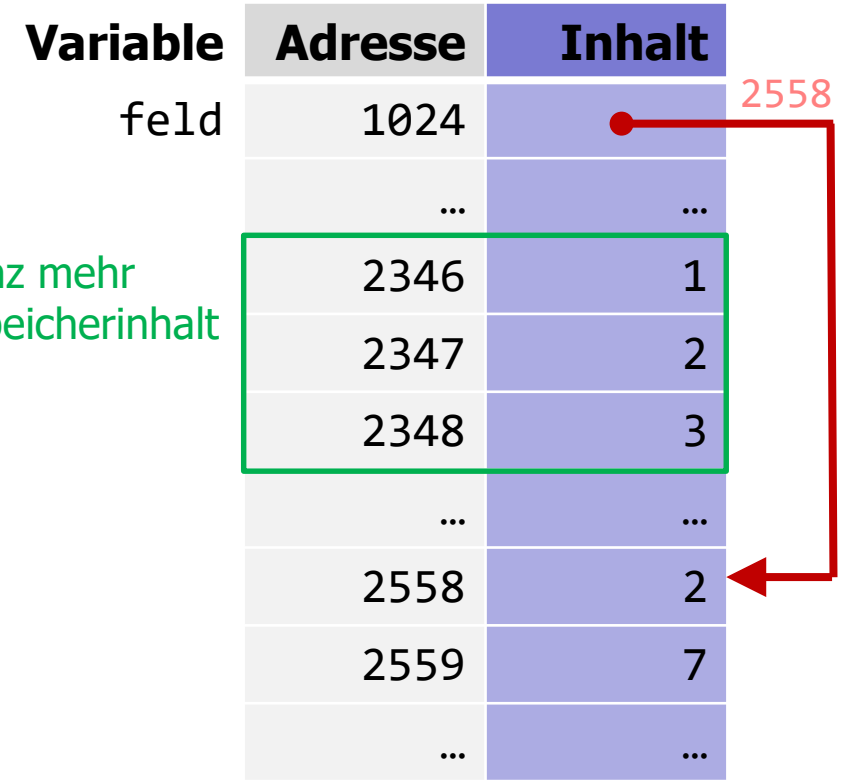
```
byte[] feld = {1, 2, 3};
…
feld = new byte[2];
feld[0] = 2;
feld[1] = 7;
```



Arrays

- Garbage Collector
  - z.B.

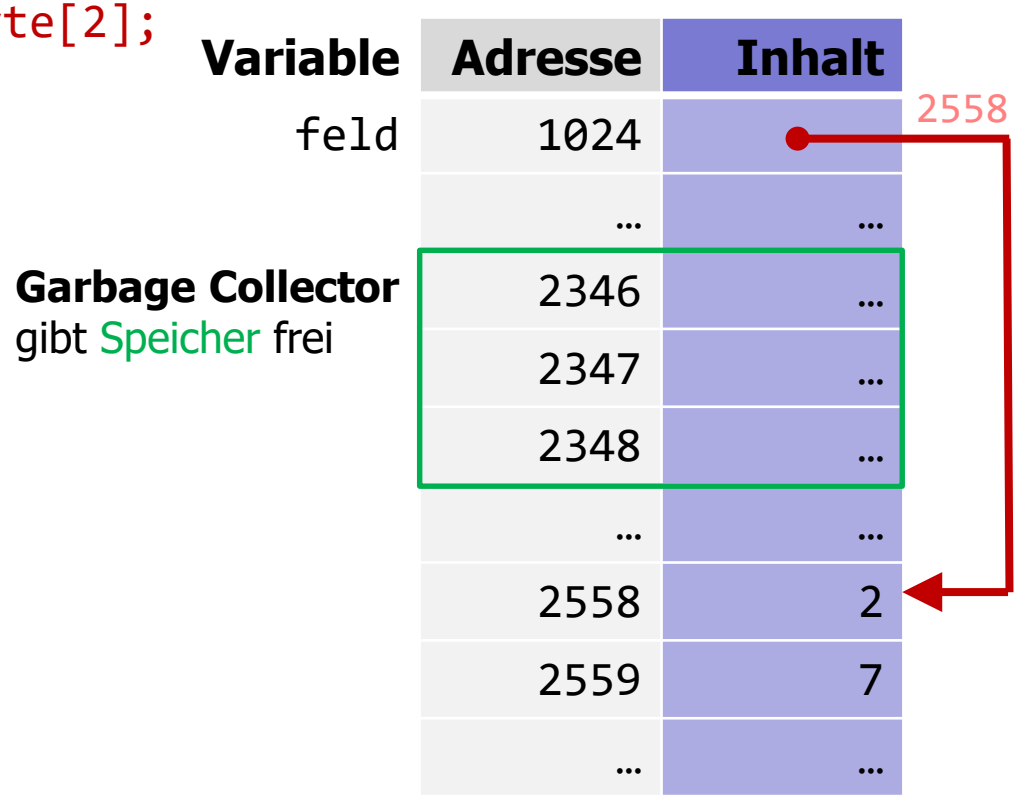
```
byte[] feld = {1, 2, 3};
…
feld = new byte[2];
feld[0] = 2;
feld[1] = 7;
```



Arrays

- Garbage Collector
  - z.B.

```
byte[] feld = {1, 2, 3};
…
feld = new byte[2];
feld[0] = 2;
feld[1] = 7;
```

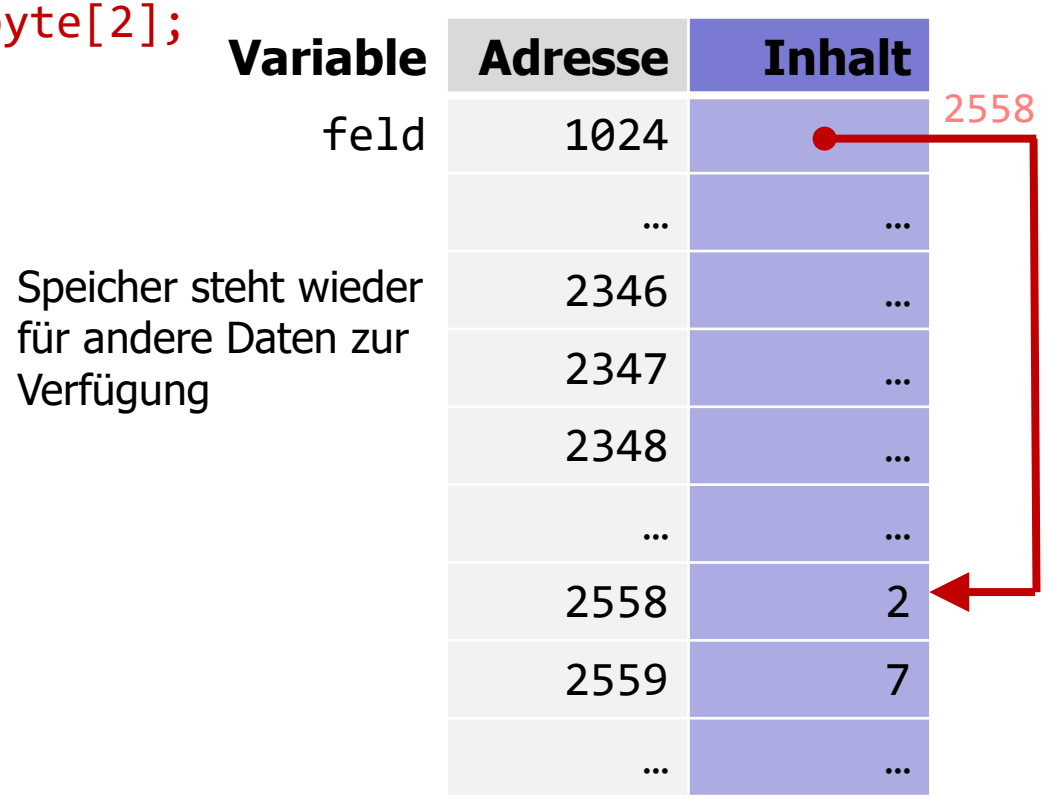


Arrays

- Garbage Collector
  - z.B.

```
byte[] feld = {1, 2, 3};
…
feld = new byte[2];
feld[0] = 2;
feld[1] = 7;
```

Speicher steht wieder für andere Daten zur Verfügung



Arrays und Methoden (Wiederholung)

- wird ein Referenzdatentyp als Parameter übergeben, so wird beim Methodenaufruf lediglich eine Kopie der Referenz erstellt
  - es gibt also nach Aufruf eine weitere Referenz auf die gleichen Daten
  - Methode kann so also Werte in aufrufenden Programmteilen verändern, obwohl das eigentlich mit **call by value** nicht möglich sein sollte
  - **Seiteneffekt**
- Deklaration einer Methode mit Referenzdatentyp: **call by reference**

Arrays und Methoden

- Beispiel:

```
public static int malzwei(int[] n) {
    n[0] *= 2;
    return n[0];
}
…
int a[] = {1};
System.out.println(malzwei(a));
// gibt 2 aus (Rückgabewert),
// a[0] ist aber hier nun auch 2
System.out.println(malzwei(a));
// gibt nun 4 als Rückgabewert aus,
// a[0] ist hier nun 4
```

Quellen zum Selbststudium

- Dirk Louis, Peter Müller: *Java – Eine Einführung in die Programmierung,* Hanser, 2014.
- Dietmar Ratz, Jens Scheffler, Detlef Seese, Jan Wiesenberger: *Grundkurs Programmieren in Java 8,* Hanser, 2014.