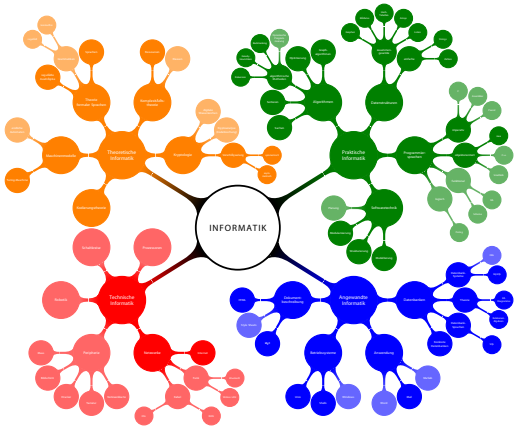
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 13

Arrays (Felder)

Nummerierte Bankschließfächer

Vorlesung Einführung in die Informatik 1 vom 5. Dezember 2013 von Till Tantau



Lernziele von Kapitel 13

1. Konzept des Arrays verstehen.
2. Java-Syntax von Arrays beherrschen.
3. Java-Programme mit Array-Nutzung implementieren können.

Gliederung von Kapitel 13

Die Probleme einer Schweizer Bank.

Eine Schweizer Bank möchte mit einem Programm den Kontostand von Nummernkonten verwalten:

Kontonummer	Betrag
1	500 SFr
2	−20 SFr
3	230000000 SFr
4	−500 SFr
. . .	. . .
500	23423523 SFr

Wie sollte dies in Java abgebildet werden?

Erste mögliche Implementierung.


```
double konto1   = 500;
double konto2   = -50;
double konto3   = 230000000;
double konto4   = -500;
// ...
double konto500 = 23423523;
```

Diese Implementierung ist aber *schlecht*, da wir beispielsweise *jedesmal das Programm ändern müssten, wenn es einen neuen Kunden gibt.*

Zweite mögliche Implementierung.


```
double[] kontos; ...

kontos [1]   = 500;
kontos [2]   = -50;
// ...
kontos [500] = 23423523;
```

Nun ist folgendes möglich:

```
double bilanz_summe = 0;
for (int i = 1; i <= 500; i = i+1) {
  bilanz_summe = bilanz_summe + kontos[i];
}
```

Was sind Arrays?

- *Arrays* (Felder) entsprechen indizierten Variablen in der Mathematik (x_i).
- Sie halten ein *feste* Anzahl von Werten,
- die alle *gleichen Typ* haben müssen.
- Arrays liegen als »Block« irgendwo im Speicher, alle Werte hintereinander weg.

Beispiel

Die Schweizer Bank würde einen Array `kontos` von `double`-Werten benutzen. Man beachte: `kontos` ist *eine* Variable, die gewissenmaßen eine »interne Struktur« hat.

Beispiel

Eine Moleküldatenbank könnte einen Array benutzen, in der jeder Eintrag ein Molekül ist.

Was sind Array-Typen?

- Hat man einen beliebigen Typ `type` gegeben, so ist `type[]` der Typ eines *Arrays von `type`-Werten*.
- Nun kann man eine Variable dieses Typs deklarieren:
  Beispiel: `double[] konto;`
- Der Array-*Typ* legt die Größe des Arrays *nicht* fest. Eine Array-Variable kann Arrays beliebiger Größe aufnehmen.
- Jeder *konkrete* Array hat aber eine *feste* Größe.

Analogie: Der Typ `String` legt auch die Länge der Zeichenkette nicht fest, jede konkrete Zeichenkette hat aber eine feste, unveränderliche Länge.

Lebenszyklus eines Arrays

- Man kann einen Array auf zwei Arten *erzeugen*. Der erzeugte Array hat dann eine *feste, unveränderliche Größe*.
- Dann können Arrays *benutzt* werden.
- Werden sie nicht mehr gebraucht, werden sie *automatisch gelöscht*.

Will man die Größe eines Arrays *ändern*, so muss man einen neuen Array der gewünschten Größe erzeugen und dann die Elemente aus dem alten Array in den neuen Array kopieren.

Wie erzeugt man neue Arrays?

Erste Methode

Bei der Deklaration einer Array-Variable darf man mittels einer speziellen Notation direkt einen Array angeben:

```
double[] konto = {50, 5000, -200, 10};
// Jetzt enthält die Variable konto
// einen Array der Länge 4
```

Zweite Methode

Man erzeugt einen leeren Array einer bestimmten Größe mittels einer anderen speziellen Notation:

```
double[] konto = new double[4];
// Jetzt enthält die Variable konto
// einen Array der Länge 4
```

Wie greift man auf Array zu?

- Hat eine Variable `var` den Typ `type[]`, so kann man mittels `var[5]` auf das sechste (!) Element zugreifen:
- Die Zählung fängt nämlich wie bei Strings bei 0 an.
- Zugriff außerhalb der Größe des Arrays führt zum Absturz:

```
int[] werte = new int[1000];

// Beliebter Anfängerfehler:
for (int i = 1; i <= 1000; i = i+1) {
  werte[i] = 0;

  // Tausendmal berührt,
  // Tausendmal ist nichts passiert,
  // Tausend und eine Nacht,
  // Da hat es ArrayOutOfBounds gemacht.
}
```

Noch ein paar Stolpersteine.

- Die Operationen `+` und `charAt` für Strings gibt es *nicht* für Arrays.
- Für eine Array-Variable `var` liefert aber `var.length` die Größe des Arrays. Anders als für Strings dürfen aber *keine Klammern* dahinter gesetzt werden (ja, das *ist* vollkommen unlogisch – life is hard).

Berechnung des Skalarprodukts zweier Vektoren.


```
double[] vector1 = {3, 5, 6, 10};
double[] vector2 = {-0.5, -1, 5, 10};

// Berechne das Skalarprodukt

double scalar_product = 0;

for (int i=0;
     i < vector1.length && i < vector2.length;
     i = i+1) {
  scalar_product = scalar_product +
                   vector1[i] * vector2[i];
}
```

Umdrehen eines Arrays.


```
double[] vec = new double[1000];

// ... vec wird gefüllt

// Drehe vec um
for (int i=0; i<vec.length/2; i = i+1)
{
  // Vertausche vec[i] und vec[vec.length-i-1]
  double temp          = vec[i];
  vec[i]               = vec[vec.length-i-1];
  vec[vec.length-i-1] = temp;
}
```

5-Minuten-Aufgabe

Zur Übung

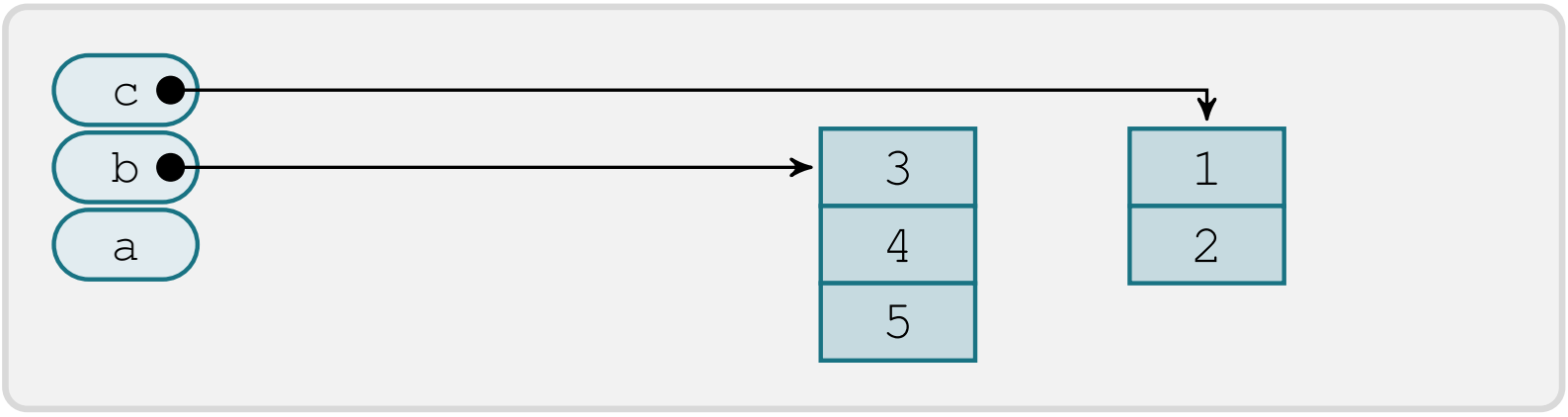
Geben Sie ein Programm mit einer For-Schleife an, das zwei Arrays verkettet.
Es sollen in `z` zuerst gerade die Werte aus `a1` kommen, gefolgt von den Werten aus `a2`.

```
// Zwei Arrays
char[] a1 = {'h', 'a', 'l'}; // oder etwas anders
char[] a2 = {'l', 'o'};      // oder etwas anders

// Der Array, in den die Verkettung hinein soll:
char[] z  = new char [a1.length+a2.length];

// Ihr Programm:
...
```

Wohin mit dem Array?

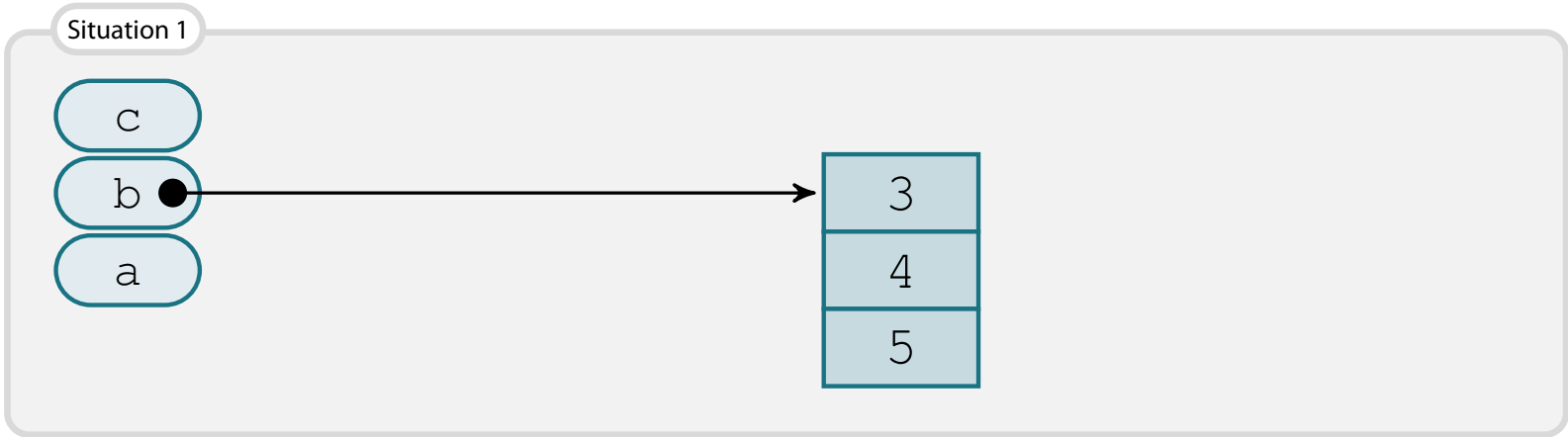
- Bei einem Array kann der Übersetzer offenbar nicht den Speicherbedarf vorher bestimmen.
- Deshalb reserviert der Übersetzer lediglich den Platz für einen *Verweis*.

```
int    a;
int[] b = {3,4,5};
int[] c = {1,2};
```



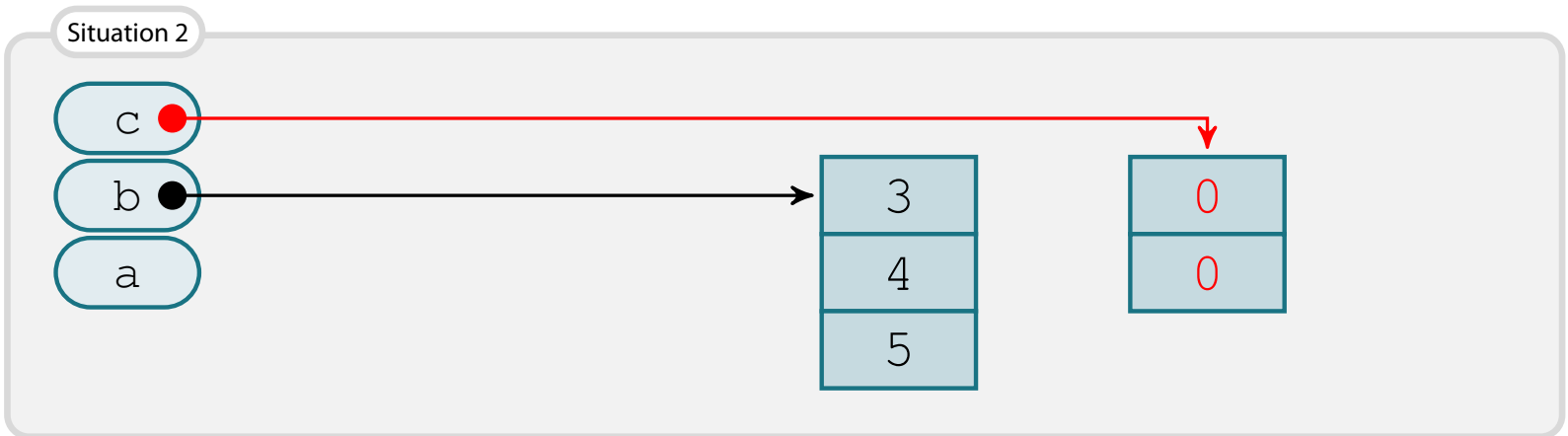
Erzeugung eines neuen Arrays mittels `new`.


```
int    a;
int[] b = {3,4,5};
int[] c;                    // Situation 1
c = new int[2];             // Situation 2
```



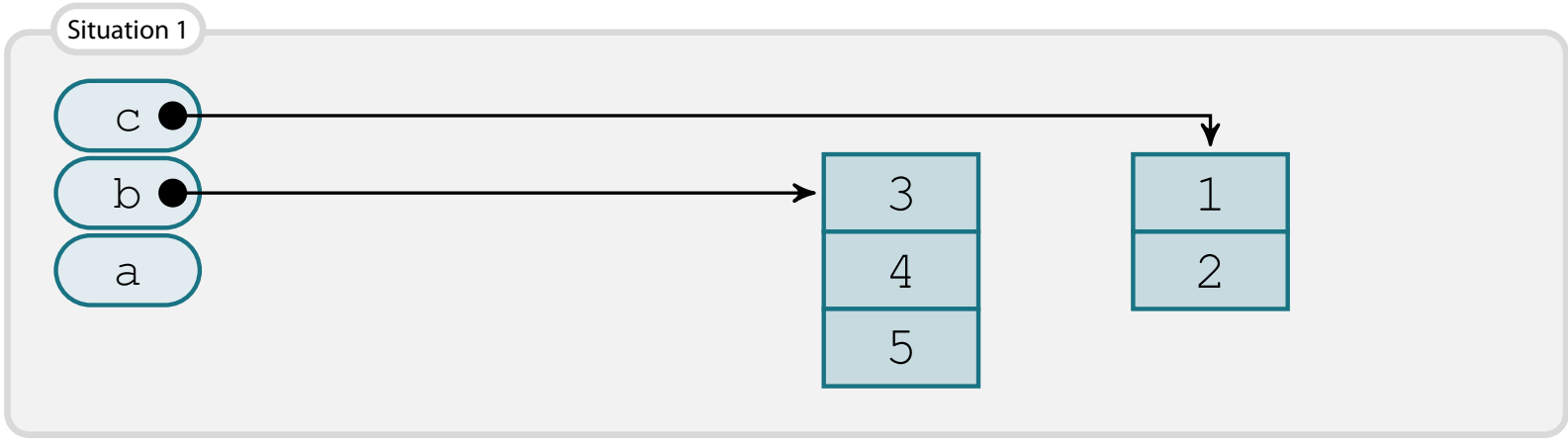
Erzeugung eines neuen Arrays mittels `new`.


```
int    a;
int[] b = {3,4,5};
int[] c;                    // Situation 1
c = new int[2];             // Situation 2
```



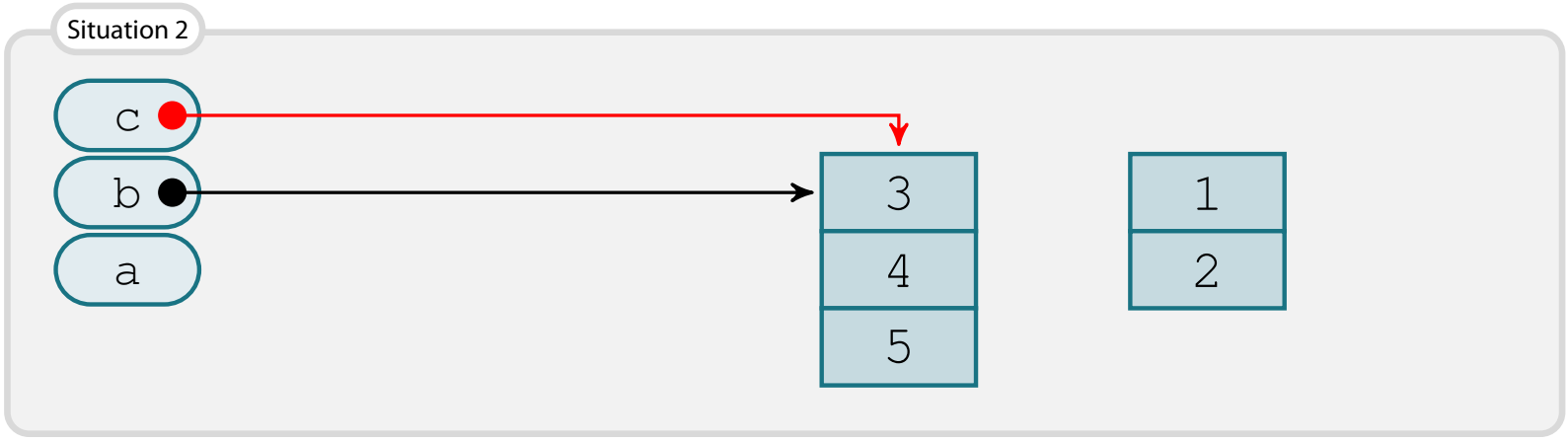
Zuweisung von Arrays.


```
int    a;
int[] b = {3,4,5};
int[] c = {1,2};          // Situation 1
c = b;                    // Situation 2
```



Zuweisung von Arrays.


```
int    a;
int[] b = {3,4,5};
int[] c = {1,2};          // Situation 1
c = b;                    // Situation 2
```



Es gibt unerwartete Effekte bei Zuweisungen von Arrays.

- Wenn man mittels `b = c` einer Arrayvariable `b` den in einer anderen Arrayvariable `c` gespeicherten zuweist, so *verweisen `b` und `c` auf denselben Array*.
- Ändert man dann `b`, so ändert man auch gleichzeitig `c`,
- was man meistens nicht will.
- Vergleicht man Arrayvariablen mittels `==`, so wird lediglich überprüft, ob die Arrayvariablen auf denselben Array verweisen, und *nicht, ob die Arrays dieselben Elemente haben*;
- und auch dies will man meistens nicht.

Moral

1. Zuweisung von Arrays sind mit Vorsicht zu genießen.
2. Vergleiche von Arrays sind mit Vorsicht zu genießen.

Zusammenfassung

Der Array-Typ

Ist `type` ein Typ, so ist `type[]` der Typ eines *Arrays von Werten vom Typ* `type`.

- In einer Variable vom Typ `type[]` können Arrays *beliebiger Größe* gespeichert werden,
- jedoch hat jeder *einzelne Array* eine feste, unabänderliche Größe.

Vergleich von Arrays und Strings

Gemeinsamkeiten:

- Beide speichern Folgen von Elementen gleichen Typs (bei Strings `char`s, bei Arrays beliebiger Typ).
- Jeder einzelne String/Array hat eine feste Größe.
- Die Zählung der Elemente beginnt bei 0.

Unterschiede:

- Bei Arrays kann man einzelne Elemente *ändern*, bei Strings nicht.
- Man kann Strings mittelt `+` verketten, Arrays nicht.
- Bei Arrays schreibt man `a.length` für die Länge, bei Strings `a.length()`.

Zusammenfassung

Arrays kann man auf zwei Arten erzeugen

- `int[] a = { 0, 0, 0, 0 };`
- `int[] a = new int [4];`

Über Arrays *iteriert* man mit Schleifen

```
int[] a = ...;
for (int i = 0; i < a.length; i = i+1) {
  ... // tue etwas mit a[i]
}
```

Vergleich von und Zuweisung von Arrays

Die Befehle `a = b;` und `a == b` machen für Arrays *nicht* das, was man erwartet.