Ausgewählte Java-Grundlagen

Eigenschaften von Variablen

Name

Typ

Wert (R-Wert, steht in Zuweisung `x = y` rechts)

Speicheradresse (L-Wert, steht in Zuweisung `x = y` links)

Lebensdauer (Allocation bis Deallocation)

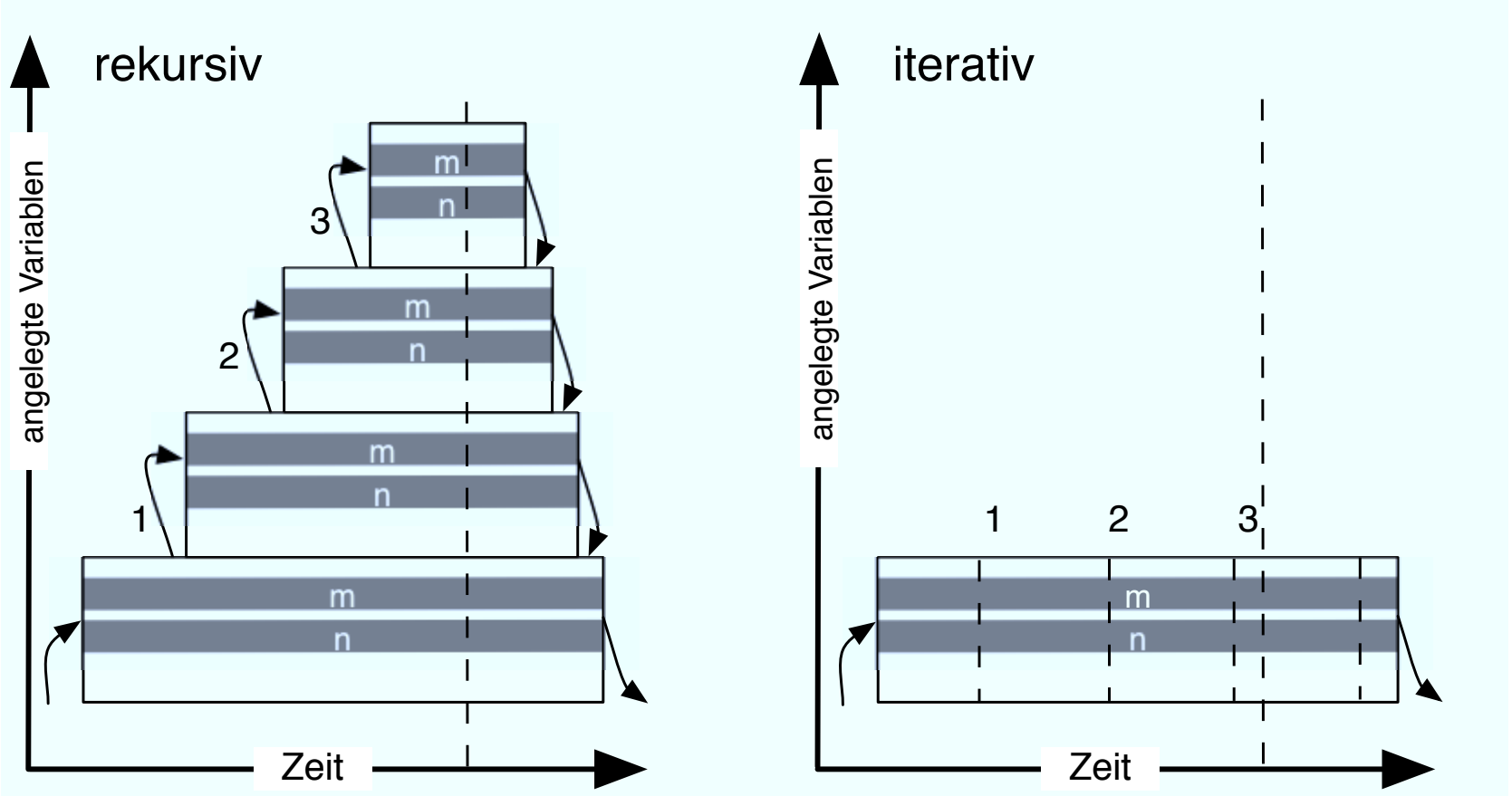
Gültigkeitsbereich (wo im Programm ist Existenz der Variablen bekannt)

Sichtbarkeitsbereich (wo im Programm ist die Variable zugreifbar)

Elementare Typen

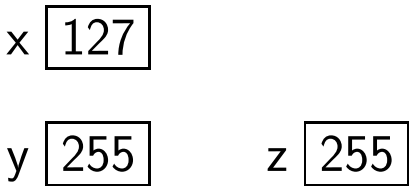
	Größe	Wertebereich	Literale
Integer-Typen:			
`byte`	1 Byte	$-2^7 \ldots 2^7 - 1$	wie `int`
`short`	2 Byte	$-2^{15} \ldots 2^{15} - 1$	wie `int`
`int`	4 Byte	$-2^{31} \ldots 2^{31} - 1$	`0`, `1`, `-1`
`long`	8 Byte	$-2^{63} \ldots 2^{63} - 1$	`0L`, `1L`, `-1L`
Fließkomma-Typen:			
`float`	4 Byte	ca. $-3.4 \cdot 10^{38} \ldots 3.4 \cdot 10^{38}$	`1.2F`, `-1.2E8F`
`double`	8 Byte	ca. $-1.8 \cdot 10^{308} \ldots 1.8 \cdot 10^{308}$	`1.2`, `-1.2E8`
Zeichen:			
`char`	2 Byte	alle 16-Bit-Unicode-Zeichen	`'A'`, `'a'`, `'3'`, `'<'`
Wahrheitswerte:			
`boolean`	1 Byte	`true`, `false`	`true`, `false`

Speicher für Rekursion und Iteration

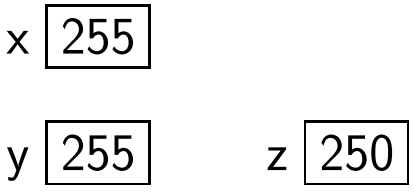


Elementarer Typ vs. Referenztyp

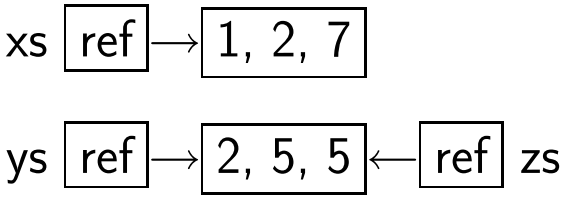
```
int x = 127;
int y = 255;
int z = y;
```



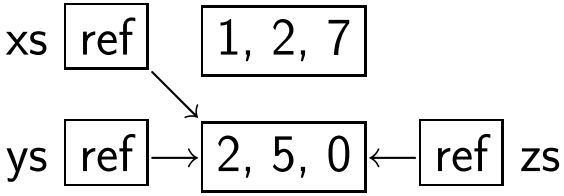

```
x = y;      z -= 5;
```




```
int[] xs = {1, 2, 7};
int[] ys = {2, 5, 5};
int[] zs = ys;
```




```
xs = ys;      zs[2] -= 5;
```



Aufgabe: Elementarer Typ vs. Referenztyp

Suchen Sie in Gruppen zu zwei bis drei Personen Antworten auf folgende Fragen:

1. Welche elementaren Typen und Referenztypen haben wir schon gesehen?
2. Warum gilt der Umgang mit elementaren Typen als sicherer und der mit Referenztypen als fehleranfälliger?
3. Warum werden Referenztypen dennoch häufig eingesetzt?

Zeit: 3 Minuten

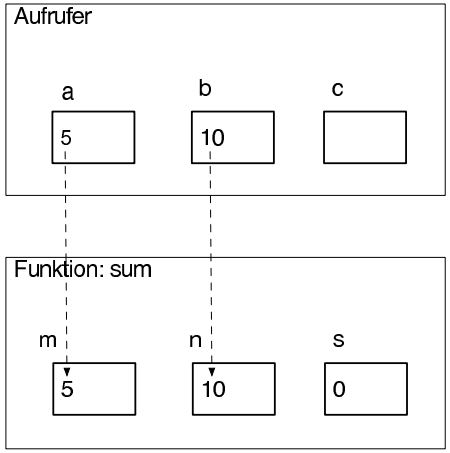
Aufgabe: Arraygröße

Schätzen Sie, wie viele Speicherzellen im Array `dreieck` in `Pascal6` bei verschiedener Anzahl an Zeilen benötigt werden:

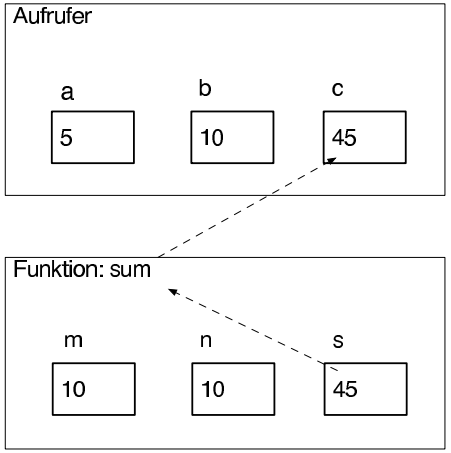
A:	6 Zeilen:	13	50 Zeilen:	101	100 Zeilen:	201
B:	6 Zeilen:	21	50 Zeilen:	1.275	100 Zeilen:	5.050
C:	6 Zeilen:	63	50 Zeilen:	31.875	100 Zeilen:	25.250
D:	6 Zeilen:	441	50 Zeilen:	1.625.625	100 Zeilen:	25.502.500

Call-by-Value (Java)

Aufruf



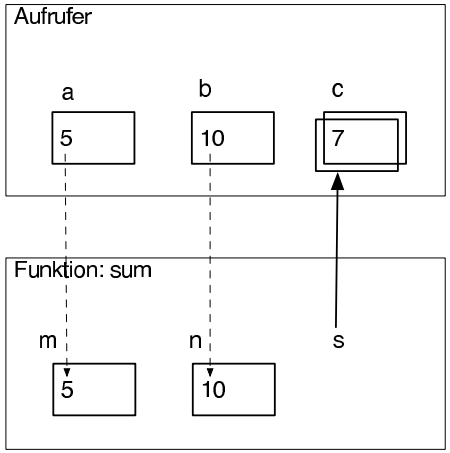
Rückgabe



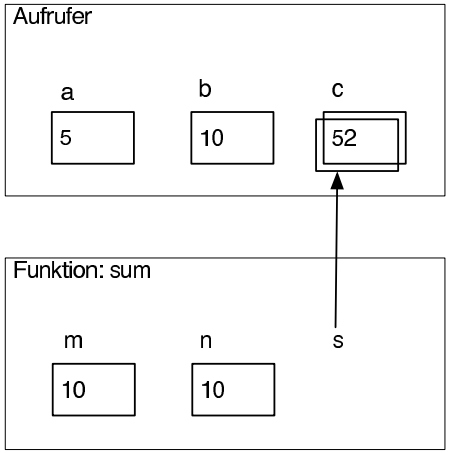

```
public static int addsum(int m, int n) {
    int s = 0;
    for (; m <= n; m++)
        s += m;
    return s;
}
```

Call-by-Reference (in Pascal)

Aufruf



Rückgabe




```
procedure addsum(m:cardinal; n:cardinal; var s:cardinal);
begin
    while m <= n do    (* in Programmiersprache Pascal *)
    begin
        s := s + m;
        m := m + 1
    end;
end;
```

Seiteneffekte

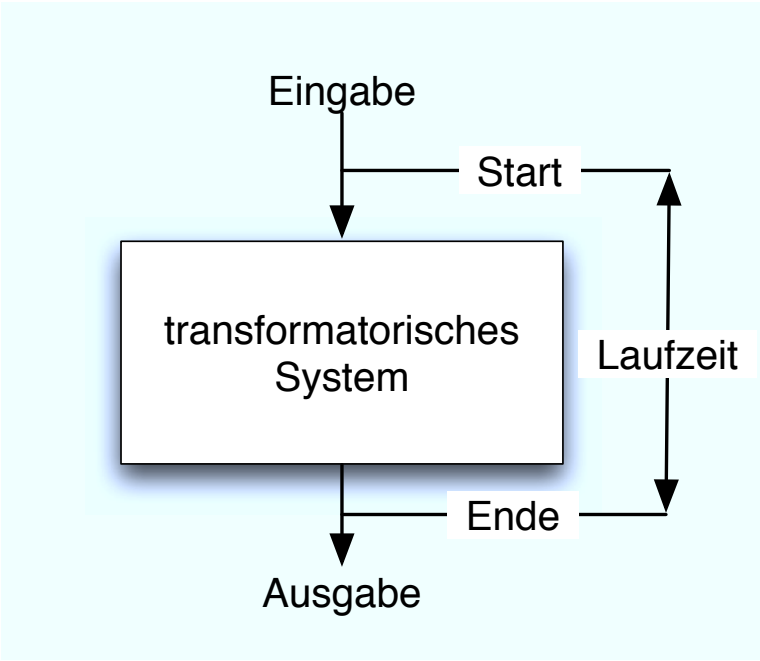
Zuweisungen sowie Ein- und Ausgaben sind Seiteneffekte

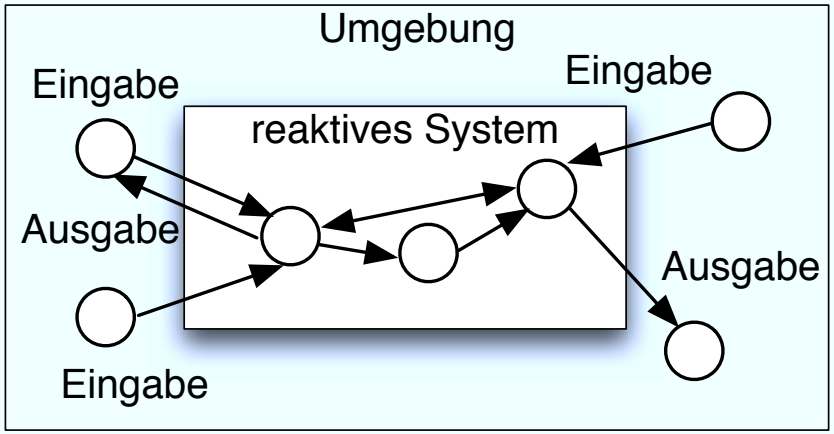
lokale Seiteneffekte nur innerhalb von Methoden (eher harmlos),
globale Seiteneffekte über Methodengrenzen hinweg (gefährlich)

Call-by-Reference bzw. Veränderung von Objekt = globaler Seiteneffekt

globale Seiteneffekte nutzen → **prozeduraler** Programmierstil
globale Seiteneffekte vermeiden → **funktionaler** Programmierstil

Transformatorisch vs. reaktiv





funktionaler Stil $\rightarrow$ transformatorisches System (eher einfach und sicher)

prozeduraler Stil $\rightarrow$ reaktives System (eher komplex und fehleranfällig)

Dilemma und Ausweg

in der Praxis vorwiegend reaktive Systeme nötig

um mit Komplexität und Fehleranfälligkeit umzugehen:

→ funktionaler Stil wo möglich für Teilaufgaben

→ prozeduraler Stil wo nötig für Zusammenfügen der Teile

objektorientierter Stil kann Nachteile des prozeduralen Stils mildern