

8. Methoden

8.1 Methoden und Parameter

8.2 Funktionen

8.3 Lokalität, Sichtbarkeit, Lebensdauer

8.4 Überladen von Methoden

8.5 Beispiele

Parameterlose Methoden



Benannte Codestücke, die über ihren Namen aufgerufen werden können

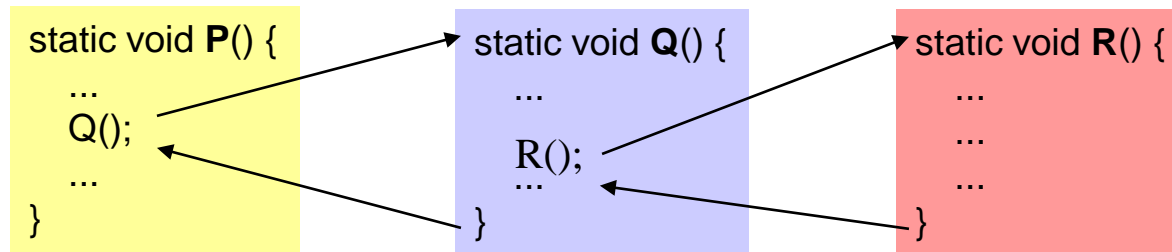
Beispiel: Ausgabe einer Überschrift

```
class Sample {  
    static void printHeader() {           // Methodenkopf  
        Out.println("Artikelliste");      // Methodenrumpf  
        Out.println("-----");  
    }  
  
    public static void main (String[] arg) {  
        printHeader();                    // Aufruf  
        ...  
        printHeader();                    // Aufruf  
        ...  
    }  
}
```

Zweck von Methoden

- Wiederverwendung
- Benutzerspezifische Operationen
- Strukturierung des Programms

Wie funktioniert ein Methodenaufruf?



Namenskonventionen

Methodennamen sollten mit *Verb* und *Kleinbuchstaben* beginnen

Beispiele

printHeader
findMaximum
traverseList
...

Parameter



Werte, die vom Rufer an die Methode übergeben werden

```
class Sample {  
    static void printMax (int x, int y) {  
        if (x > y) Out.print(x); else Out.print(y);  
    }  
  
    public static void main (String[] arg) {  
        ...  
        printMax(100, 2 * i);  
    }  
}
```

formale Parameter

- im Methodenkopf (hier x, y)
- sind Variablen der Methode

aktuelle Parameter

- an der Aufrufstelle (hier 100, 2*i)
- können Ausdrücke sein

Parameterübergabe

Aktuelle Parameter werden den formalen Parametern zugewiesen

x = 100; **y** = 2 * i;

- ▷ aktuelle Parameter müssen mit formalen *zuweisungskompatibel* sein
z.B. printMax(shortVal, byteVal); möglich
- ▷ formale Parameter enthalten *Kopien* der aktuellen Parameter

Methodenschnittstelle (Signatur)

```
void printMax (int x, int y)
```

legt fest, wie die Methode aufgerufen werden kann 4

Parameterübergabe von Arrays

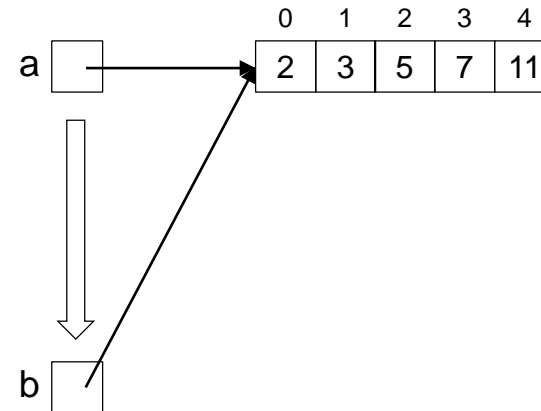


Rufer

```
int[] a = {2, 3, 5, 7, 11};  
...  
Process(a);  
...
```

Gerufene Methode

```
static void Process (int[] b) {  
    ...  
    int x = b[3]; // greift auf a[3] zu  
    ...  
    b[3] = 0;    // ändert auch a[3]!  
}
```



"Alias-Effekt"

Formale Parameter sind normalerweise Kopien der aktuellen Parameter

Bei Arrays wird aber nur die Referenz kopiert, nicht das Array (=> es kommt zu Alias-Effekten)

Variable Anzahl von Parametern

Gewünschter Methodenaufruf

```
int sum1 = SumOf(a, b, c);
int sum2 = SumOf(a, b, c, d, e);
```

variable Anzahl von Parametern
(hier vom Typ *int*)

Vararg-Parameter

bedeutet: 1 oder mehrere *int*-Parameter

```
static void SumOf (int... values) {
    int sum = 0;
    for (int val: values) {
        sum = sum + val;
    }
    return sum;
}
```

oder

wird wie ein Array betrachtet

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

Vararg-Parameter muss der letzte Parameter sein => es darf nicht mehrere Vararg-Parameter geben

```
static void SumOf (int threshold, int... values) {
    int sum = 0;
    for (int val: values) {
        if (val >= threshold) sum = sum + val;
    }
    return sum;
}
```

8. Methoden

8.1 Methoden und Parameter

8.2 Funktionen

8.3 Lokalität, Sichtbarkeit, Lebensdauer

8.4 Überladen von Methoden

8.5 Beispiele

Funktionen



Methoden, die einen Ergebniswert an den Rufer zurückliefern

```
class Sample {  
    static int max (int x, int y) {  
        if (x > y) return x; else return y;  
    }  
  
    public static void main (String[] arg) {  
        ...  
        int result = 3 * max(100, i + j) + 1;  
        ...  
    }  
}
```

- haben Funktionstyp (z.B. *int*) statt *void* (= kein Typ)
- liefern Ergebnis mittels return-Anweisung an den Rufer zurück (muss zuweisungskompatibel mit Funktionstyp sein)
- Werden wie Operanden in einem Ausdruck benutzt

Funktionen Methoden mit Rückgabewert

```
static int max (int x, int y) {...}
```

Prozeduren Methoden ohne Rückgabewert

```
static void printMax (int x, int y) {...}
```


Weiteres Beispiel einer Funktion

Ganzzahliger Zweierlogarithmus

```
class Sample {  
    static int log2 (int n) { // assert: n > 0  
        int res = 0;  
        while (n > 1) {n = n / 2; res++;}  
        return res;  
    }  
  
    public static void main (String[] arg) {  
        int x = log2(17); // x == 4  
        ...  
    }  
}
```

n	res
17	0
8	1
4	2
2	3
1	4

Return in Prozeduren



Auch Prozeduren können vorzeitig mit *return* abgebrochen werden

```
class ReturnDemo {  
  
    static void printLog2 (int n) {  
        if (n <= 0) return;           // kehrt zum Rufer zurück  
        int res = 0;  
        while (n > 1) {n = n / 2; res++;}  
        Out.println(res);  
    }  
  
    public static void main (String[] arg) {  
        int x = In.readInt();  
        if (!In.done()) return;      // beendet das Programm  
        printLog2(x);  
        ...  
    }  
}
```

Funktionen müssen mit return beendet werden

Prozeduren können mit return beendet werden

8. Methoden

8.1 Methoden und Parameter

8.2 Funktionen

8.3 Lokalität, Sichtbarkeit, Lebensdauer

8.4 Überladen von Methoden

8.5 Beispiele

Lokale und globale Variablen



Sichtbarkeit

Globale (statische) Variablen

auf Klassenebene mit *static* deklariert;
in allen Methoden dieser Klasse sichtbar

Lokale Variablen

in einer Methode ohne *static* deklariert
(nur in dieser Methode sichtbar)

```
class C {  
    static int a, b;  
    static void P() {  
        int x, y;  
        ...  
    }  
    ...  
}
```

A diagram illustrating variable visibility. A line from the 'Globale (statische) Variablen' section points to the line 'static int a, b;'. Another line from the 'Lokale Variablen' section points to the line 'int x, y;'. The code is highlighted in a yellow box.

Lebensdauer

Globale (statische) Variablen

am Programmbeginn angelegt
am Programmende wieder freigegeben
("überleben" Methodenaufrufe)

Lokale Variablen

bei jedem Aufruf der Methode neu angelegt
am Ende der Methode wieder freigegeben

Beispiel: Summe einer Zahlenfolge



falsch!

```
class Wrong {  
    static void add (int x) {  
        int sum = 0;  
        sum = sum + x;  
    }  
  
    public static void main (String[] arg) {  
        add(1); add(2); add(3);  
        Out.println("sum = " + sum);  
    }  
}
```

richtig!

```
class Correct {  
    static int sum = 0;  
  
    static void add (int x) {  
        sum = sum + x;  
    }  
  
    public static void main (String[] arg) {  
        add(1); add(2); add(3);  
        Out.println("sum = " + sum);  
    }  
}
```

- *sum* ist in *main* nicht sichtbar
- *sum* wird bei jedem Aufruf von *add* neu angelegt (alter Wert geht verloren)

Variablen möglichst lokal deklarieren, nicht als statische Variablen.

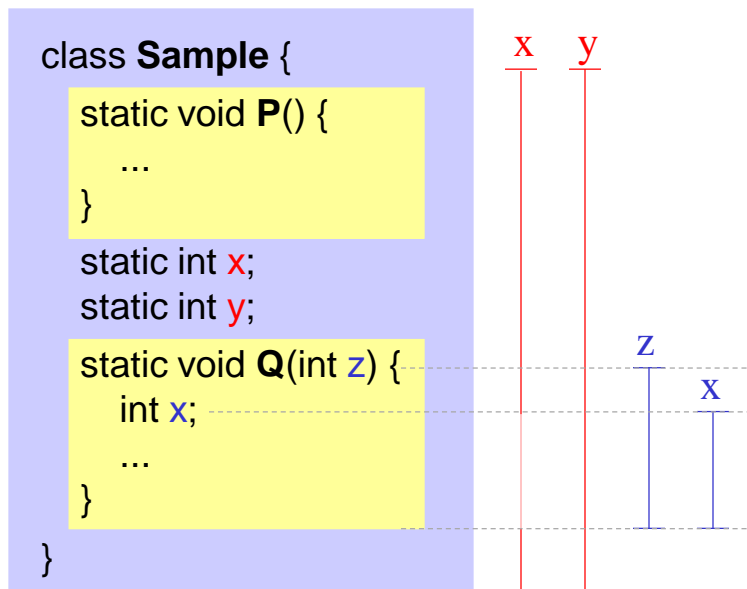
Vorteile

- **Übersichtlichkeit**
Deklaration und Benutzung nahe beisammen
- **Sicherheit**
Lokale Variablen können nicht durch andere Methoden zerstört werden
- **Effizienz**
Zugriff auf lokale Variable ist oft schneller als Zugriff auf statische Variable

Sichtbarkeitsbereich von Namen



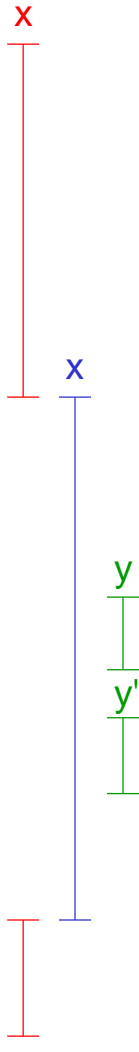
Programmstück, in dem auf diesen Namen zugegriffen werden kann
(auch *Gültigkeitsbereich* oder *Scope* des Namens genannt)



Sichtbarkeitsregeln

1. Ein Name darf in einem Block nicht mehrmals deklariert werden (auch nicht in geschachtelten Anweisungsblöcken).
2. Lokale Namen verdecken Namen, die auf Klassenebene deklariert sind.
3. Der Sichtbarkeitsbereich eines lokalen Namens beginnt bei seiner Deklaration und geht bis zum Ende der Methode.
4. Auf Klassenebene deklarierte Namen sind in allen Methoden der Klasse sichtbar.

Beispiel zu den Sichtbarkeitsregeln



The diagram illustrates the visibility of variables x, y, and y' across the code blocks. A red vertical line with a horizontal tick at the top is labeled 'x'. A blue vertical line with horizontal ticks at the top and bottom is labeled 'x'. A green vertical line with horizontal ticks at the top and bottom is labeled 'y'. A green vertical line with horizontal ticks at the top and bottom is labeled 'y'.

```
class Sample {  
    static void P() {  
        Out.println(x);           // gibt 0 aus  
    }  
  
    static int x = 0;              // statisches x  
  
    public static void main (String[] arg) {  
        Out.println(x);           // gibt 0 aus  
        int x = 1;                // lokales x: verdeckt statisches x  
        Out.println(x);           // gibt 1 aus  
        P();  
        if (x > 0) {  
            int x;                // Fehler: x ist in main bereits deklariert  
            int y;  
            ...  
        } else {  
            int y;  
            ...  
        }  
        for (int i = 0; ...) {...}  
        for (int i = 1; ...) {...} // ok, kein Konflikt mit i aus letzter Schleife  
    }  
    ...  
    ...  
}
```


Lebensdauer von Variablen

```
class LifenessDemo {
    static int g;

    static void A() {
        int a;
        f..
    }

    static void B() {
        int b;
        ... A(); „.. A(); ...
    }

    public static void main(String[] arg) {
        int m;
        ... B(); †.
    }
}
```

Lokale Variablen

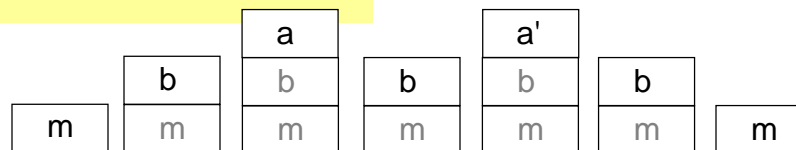
leben nur während der Ausführung ihrer Methode

Statische Variablen

leben während der gesamten Programmausführung

Sichtbarkeit ¹ Lebensdauer

lokale Variablen
(Methoden-Stack)



statische Var.



• , f „ f ... †

8. Methoden

- 8.1 Methoden und Parameter
- 8.2 Funktionen
- 8.3 Lokalität, Sichtbarkeit, Lebensdauer
- 8.4 Überladen von Methoden
- 8.5 Beispiele

Überladen von Methoden



Methoden mit gleichem Namen aber verschiedenen Parameterlisten können in derselben Klasse deklariert werden

```
static void write (int i) {...}  
static void write (float f) {...}  
static void write (int i, int width) {...}
```

müssen sich in *Anzahl* und/oder *Typ* der Parameter unterscheiden

Beim Aufruf wird diejenige Methode gewählt, die am besten zu den aktuellen Parametern passt

```
write(100);    ➤ write (int i)  
write(3.14f);  ➤ write (float f)  
write(3.14);   ➤ Fehler! Es gibt keine Methode mit double-Parameter  
write(100, 5); ➤ write (int i, int width)  
short s = 17;  
write(s);      ➤ write (int i);
```

8. Methoden

- 8.1 Methoden und Parameter
- 8.2 Funktionen
- 8.3 Lokalität, Sichtbarkeit, Lebensdauer
- 8.4 Überladen von Methoden
- 8.5 Beispiele

Beispiele



Größter gemeinsamer Teiler nach Euklid

```
static int ggt (int x, int y) {  
    int rest = x % y;  
    while (rest != 0) {  
        x = y; y = rest; rest = x % y;  
    }  
    return y;  
}
```

`ggt(8, 12)` \models 4

Kürzen eines Bruchs

```
static void reduce (int z, int n) {  
    int x = ggt(z, n);  
    Out.println((z / x) + "/" + (n / x));  
}
```

`reduce(8, 12);` \models 2/3

Beispiele



Prüfe, ob x eine Primzahl ist

```
static boolean isPrime (int x) {  
    if (x == 2) return true;  
    if (x % 2 == 0) return false;  
    int i = 3;  
    while (i * i <= x) {  
        if (x % i == 0) return false;  
        i = i + 2;  
    }  
    // i > Öx und x ist durch keine Zahl  
    // zwischen 2 und i teilbar  
    return true;  
}
```

```
for (int i = 3; i * i <= x; i += 2)  
    if (x % i == 0) return false;
```

Beispiele



Berechne x^n

```
static long power (int x, int n) {  
    long res = 1;  
    for (int i = 1; i <= n; i++) res = res * x;  
    return res;  
}
```

$\underbrace{x * x * x * \dots * x}_{n \text{ mal}}$

Dasselbe effizienter

```
static long power (int x, int n) {  
    long res = 1;  
    while (n > 0) {  
        if (n % 2 == 0) {  
            x = x * x; n = n / 2;    //  $x^{2i} = (x^2)^i$   
        } else {  
            res = res * x; n--;      //  $x^{i+1} = x^i * x$   
        }  
    }  
    return res;  
}
```

x	n	res
2	5	1
	4	2
4	2	
	1	
16	0	32

Beispiel: Nachbauen von readInt



```
static int readInt() {  
    int val = 0;  
    char ch = In.read(); // liest ein einzelnes Zeichen  
    while (In.done() && '0' <= ch && ch <= '9') {  
        val = 10 * val + (ch - '0');  
        ch = In.read();  
    }  
    // ! In.done() || ch < '0' || ch > '9'  
    return val;  
}
```

'0' ... 48
'1' ... 49
'2' ... 50
'3' ... 51
'4' ... 52
'5' ... 53
'6' ... 54
'7' ... 55
'8' ... 56
'9' ... 57

Schreibtischtest: Eingabe 123+

val	ch	
0	'1' (49)	'0' = 48
1	'2' (50)	
12	'3' (51)	
123	'+' (43)	

"Horner-Schema"

Beispiel: Nachbauen von toString(int)



```
static String toString(int n) {  
    StringBuilder b = new StringBuilder();  
    do {  
        int lastDigit = n % 10;  
        b.insert(0, (char)('0' + lastDigit));  
        n = n / 10;  
    } while (n > 0);  
    return b.toString();  
}
```

'0' ... 48
'1' ... 49
'2' ... 50
'3' ... 51
'4' ... 52
'5' ... 53
'6' ... 54
'7' ... 55
'8' ... 56
'9' ... 57

Schreibtischtest: n == 123

n	lastDigit	b
123		""
12	3	"3"
1	2	"23"
0	1	"123"