PGdP

- Switch
- Felder
- Abkürzungen
- Methoden
- Syntax
- P-Aufgaben

Folien: go.tum.de/904005

Switch

Mehrere Bedingungen prüfen

```
int a = readInt();
switch(a) {
  case 3: write("drei"); break;
  case 7: write("sieben");  //kein break
    //Ausführung für 7 geht bei 0 weiter
  case 0: write("null"); break;
  default: write("something");
    break;
}
```

break am Ende, default Case obligatorisch

Felder

Deklaration

```
int[] a = new int[length];
```

Länge eines Arrays

```
a.length
```

For-Schleife

```
for(init; cond; modify) {stmt}
```

```
for(int i=0; i<a.length; i++)
```

Startwert: `int i=0;`

Endwert: `i<a.length;`

Schrittweite: `i++`

Arrays werden von 0 bis $\text{length} - 1$ indiziert.

Tobias Schamel

Felder

Implizite Erzeugung von Feldern

```
int[] a = {3, 5, 7};
int[][] b = {
   { 3 },
   { 5 },
   { 7, 9 }
};
```

Index out of Bound Exception

```
a[3]; //java.lang.ArrayIndexOutOfBoundsException
```

Felder

Referenzen

```
int[] a; //Zeiger auf Feld für [Integer]
```

Mehrdimensionale Felder:

Felder können auch Referenzen speichern

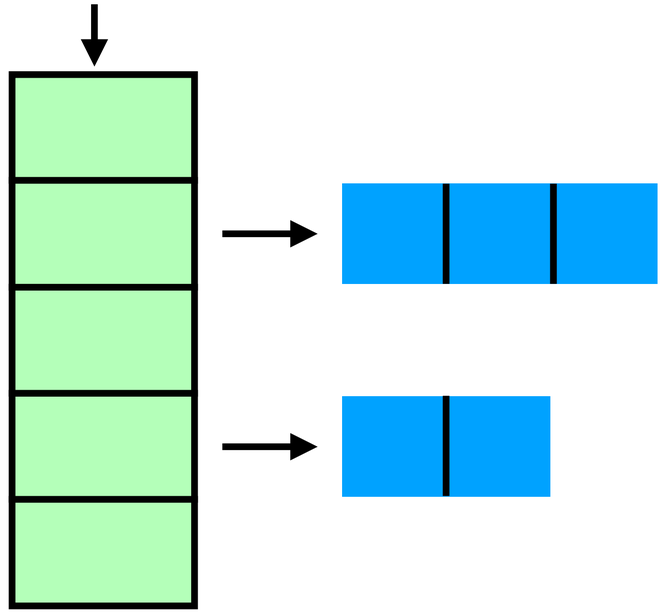
```
int[][] b; /* Zeiger auf Feld für [Zeiger auf
 Feld für [Integer]] */
b = new int[length1][length2]
```

Felder

Mehrdimensionale Felder

```
int[][] b = new int[5][];
```

```
b[1][] = new int[3];
b[3][] = new int[2];
```



Felder

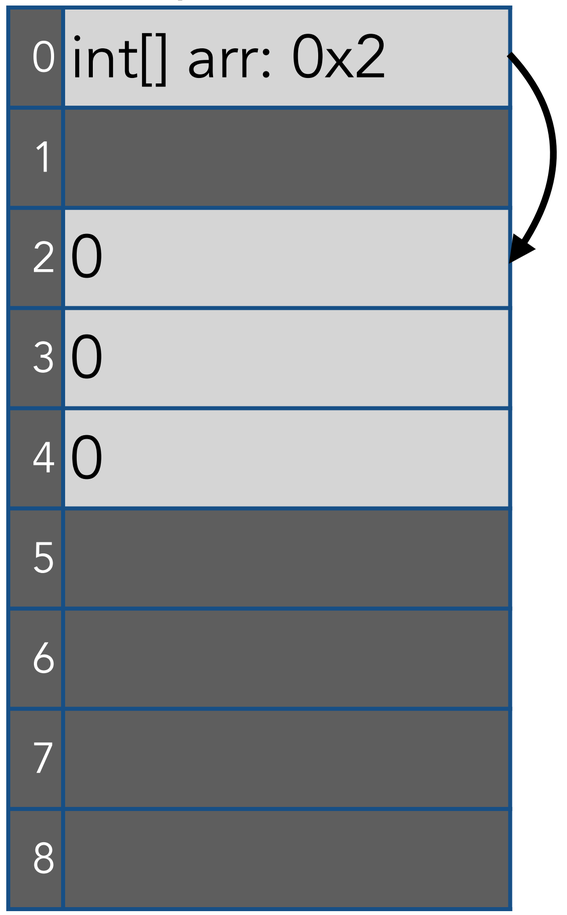
```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7
8   write(arr[1] + arr[2]);
9
10  int[] a2 = arr;
11
12  a2[1] = 7;
13
14  write(arr[1] + arr[2]);
```

Speicher

0	int[] arr: null
1	
2	
3	
4	
5	
6	
7	
8	

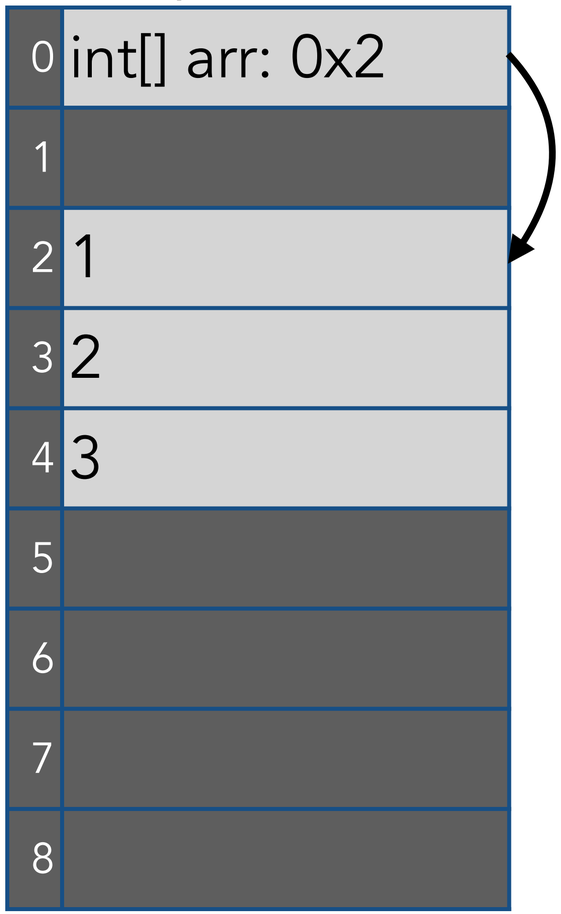
Felder

```
1   int[] arr;
2   arr = new int[3];
3   for(int i = 0; i < 3; i++)
4     arr[i] = i+1;
5   write(arr[1] + arr[2]);
6   int[] a2 = arr;
7   a2[1] = 7;
8   write(arr[1] + arr[2]);
9
10
```



Felder

```
1   int[] arr;
2
3   arr = new int[3];
4   for(int i = 0; i < 3; i++)
5     arr[i] = i+1;
6   write(arr[1] + arr[2]);
7
8   int[] a2 = arr;
9   a2[1] = 7;
10  write(arr[1] + arr[2]);
```

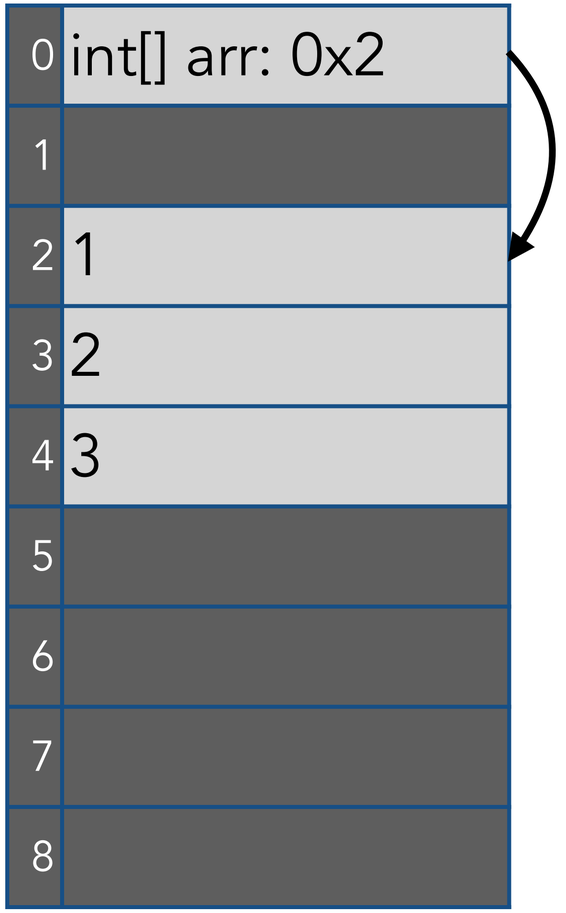


Felder

```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7   write(arr[1] + arr[2]);
8
9   int[] a2 = arr;
10
    a2[1] = 7;
    write(arr[1] + arr[2]);
```

```
> 5
```

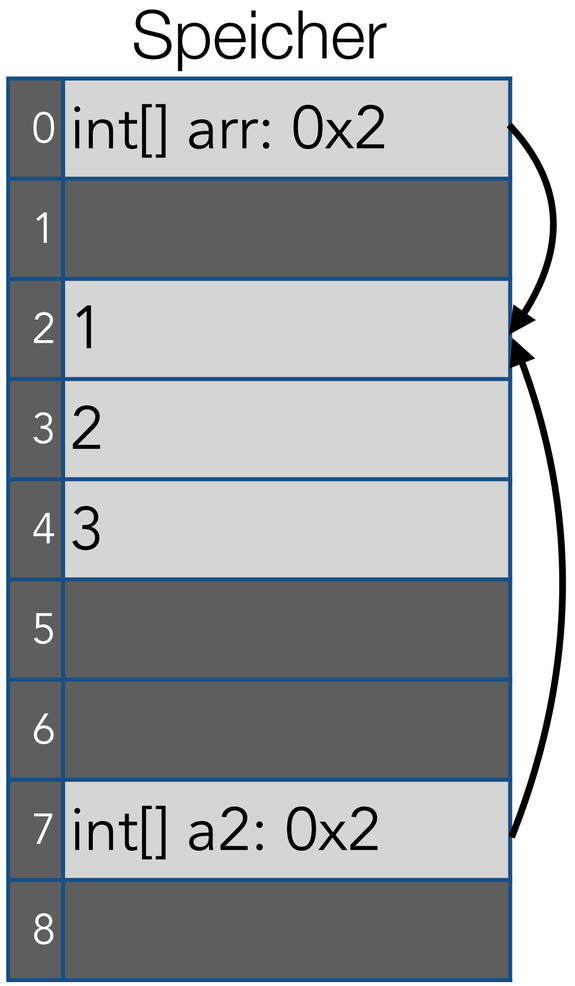
Speicher



Felder

```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7   write(arr[1] + arr[2]);
8   int[] a2 = arr;
9   a2[1] = 7;
10  write(arr[1] + arr[2]);
```

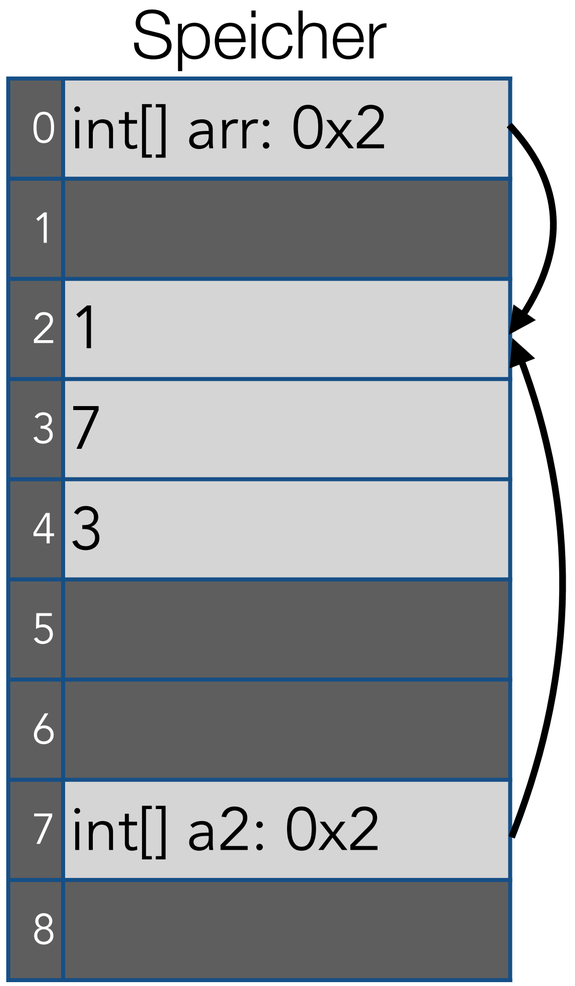
```
> 5
```



Felder

```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7
8   write(arr[1] + arr[2]);
9   int[] a2 = arr;
10  a2[1] = 7;
11  write(arr[1] + arr[2]);
```

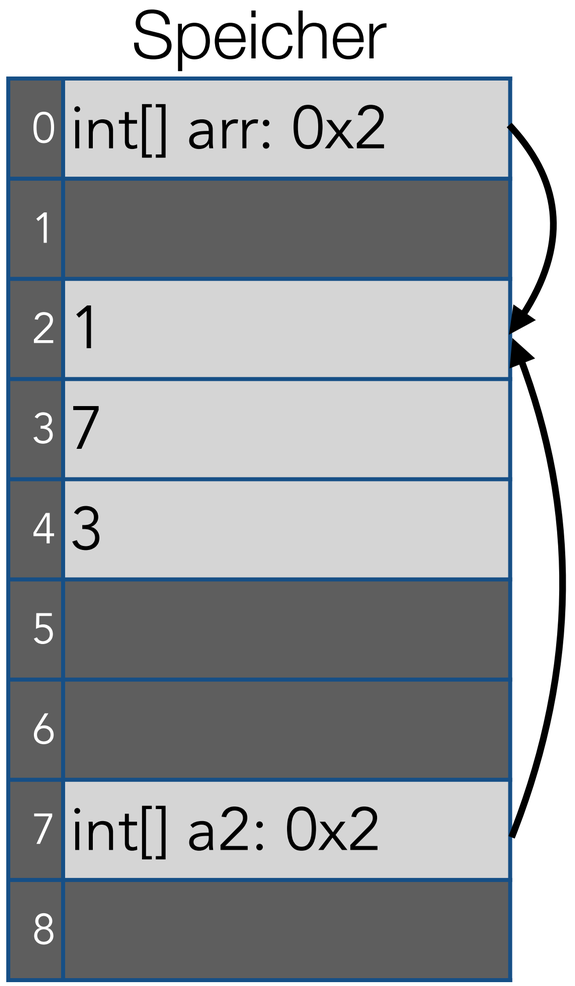
```
> 5
```



Felder

```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7
8   write(arr[1] + arr[2]);
9   int[] a2 = arr;
10  a2[1] = 7;
    write(arr[1] + arr[2]);
```

```
> 5
> 10
```



Abkürzungen

Post- und Preinkrement, Post- und Predekrement

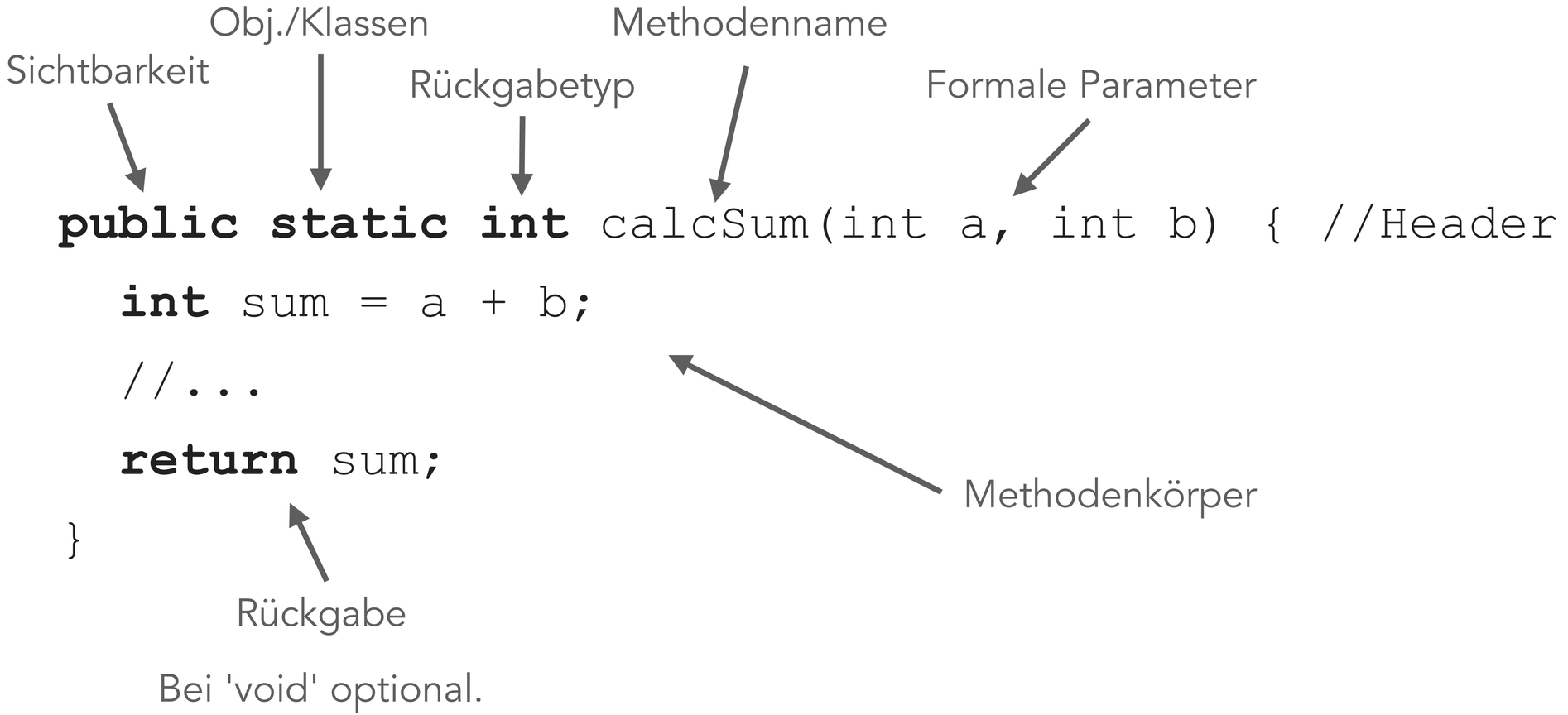
```
a[i++] = m;   ≡   a[i] = m; i = i + 1;
a[++i] = m;   ≡   i = i + 1; a[i] = m;
```

Kombinierte Zuweisung

```
i += 5;   ≡   i = i + 5;
i %= 4;   ≡   i = i % 4;
```

Methoden

Teilprobleme separat lösen, Lösungen mehrfach verwenden

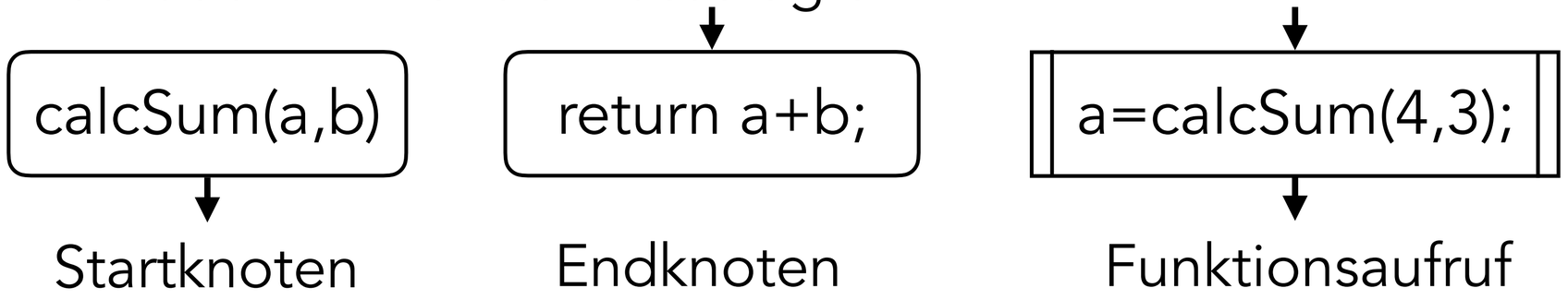


Methoden

Aufruf einer Methode

```
public static void main(String[] args) {
  int a = calcSum(4, 3);
  write(calcSum(readInt(), readInt()));
}
```

Methoden im Kontrollflussdiagramm



Syntax

Abgrenzung Semantik & Syntax:

Die **Syntax** einer Sprache (eines Zeichensystems) beschreibt die Regeln, nach denen die Sprachkonstrukte (Zeichen des Zeichensystems) gebildet werden.

Die **Semantik** einer Sprache (eines Zeichensystems) beschreibt die Bedeutung der Sprachkonstrukte (Zeichen des Zeichensystems).

Syntax

Syntax ≡ Grammatik

```
| Alternative
* Iteration + Konkatenation    (0 – ∞)
? Option + Konkatenation       (0,1)
```

Beispiel Postleitzahl mit opt. Länderkennung

```
lkennung ::= letter? letter? letter -
nummer ::= digit digit digit digit digit
plz ::= lkennung? nummer
```

Beispiel Nachname Maier/Mayer/… & Jannik/Yannic/…

```
M(e|a)(i|y)er      (J|Y)an(n)?i(ck|k|c)
```

Syntax

Syntaxbäume

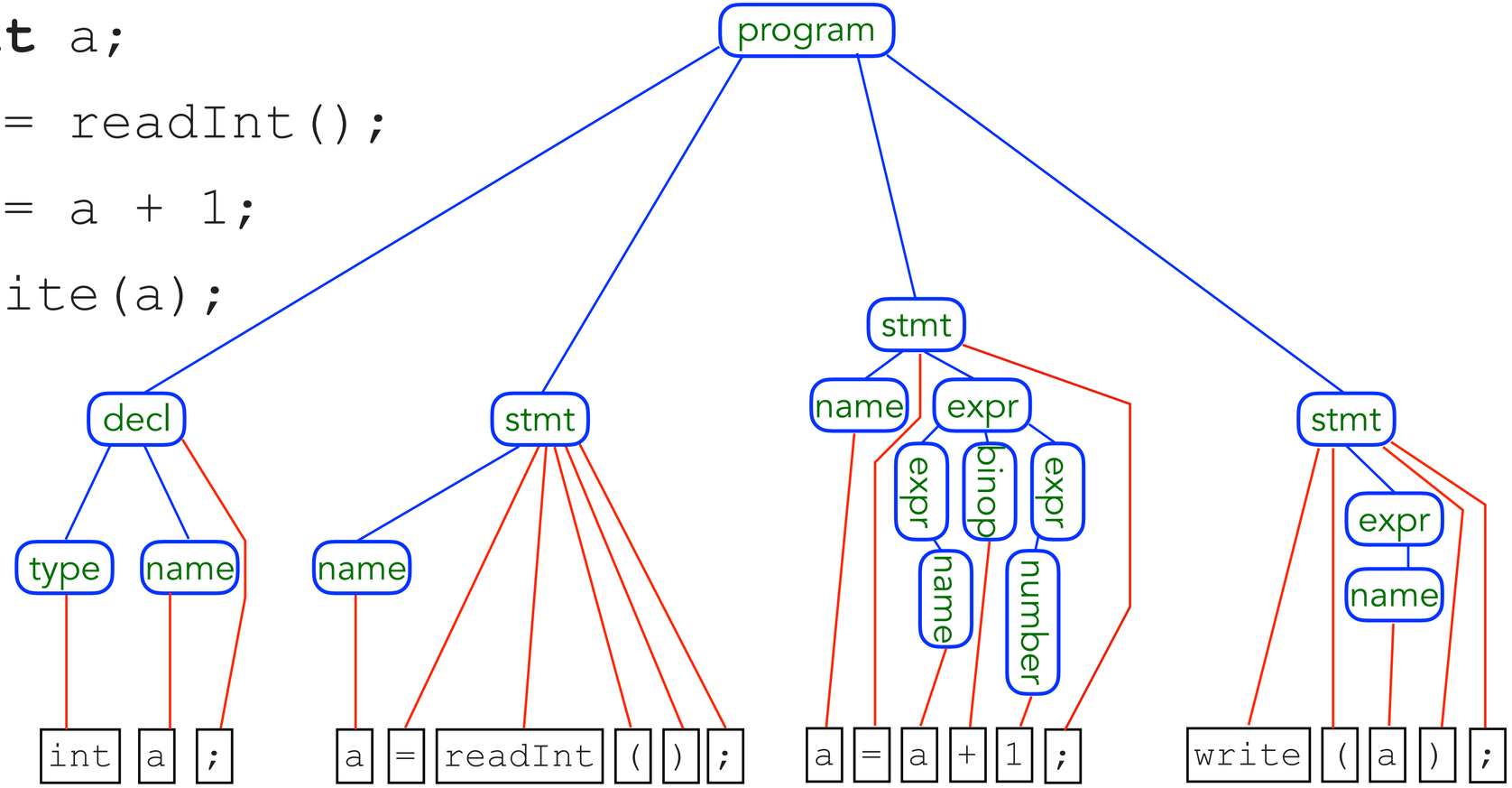
```
program   ::=  decl* stmt*
decl      ::=  type name ( , name )* ;
type      ::=  int
stmt      ::=  ; | { stmt* } |
               name = expr; | name = readInt(); | write( expr ); |
               if ( cond ) stmt |
               if ( cond ) stmt else stmt |
               while ( cond ) stmt
expr      ::=  number | name | ( expr ) |
               unop expr | expr binop expr
unop      ::=  -
binop     ::=  - | + | * | / | %
```

```
cond      ::=  true | false | ( cond ) |
               expr comp expr |
               bunop cond | cond bbinop cond
comp      ::=  == | != | <= | < | >= | >
bunop     ::=  !
bbinop    ::=  && | ||
```

Syntax

Syntaxbäume

```
int a;
a = readInt();
a = a + 1;
write(a);
```



P03.1

Syntaxbaum entsprechend der MiniJava-Grammatik

```
1  int x,r;
2  int n;
3  r = 1;
4  n = 1;
5  x = readInt();
6
7
8
9
```

```
10 while (n < x) {
11     if (r % 1 == 0)
12         r = r * n;
13     else {
14         r = r * (-n);
15     }
16     n = n + 1;
17     write (r);
18 }
```

P03.2

Dezimale, Binäre, Oktale & Hexadezimale Zahlen

Dez.	10^2	10^1	10^0	Calc
1	0	0	1	0•100+0•10+1•1
682	6	8	2	6•100+8•10+2•1

Hex.	16^2	16^1	16^0	Calc
1	0	0	1	0•256+0•16+1•1
682	2	A	A	2•256+A•16+A•1

Okt	8^3	8^2	8^1	8^0	Calc
1	0	0	0	1	0•512+0•64+0•8+1•1
682	1	2	5	2	1•512+2•64+5•8+2•1

Hex.	Dez.
1	1
…	…
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Bin.	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^2	2^1	2^0
1	0	0	0	0	0	0	0	0	0	0
682	1	0	1	0	1	0	1	0	1	0

P03.2

Codebeispiel

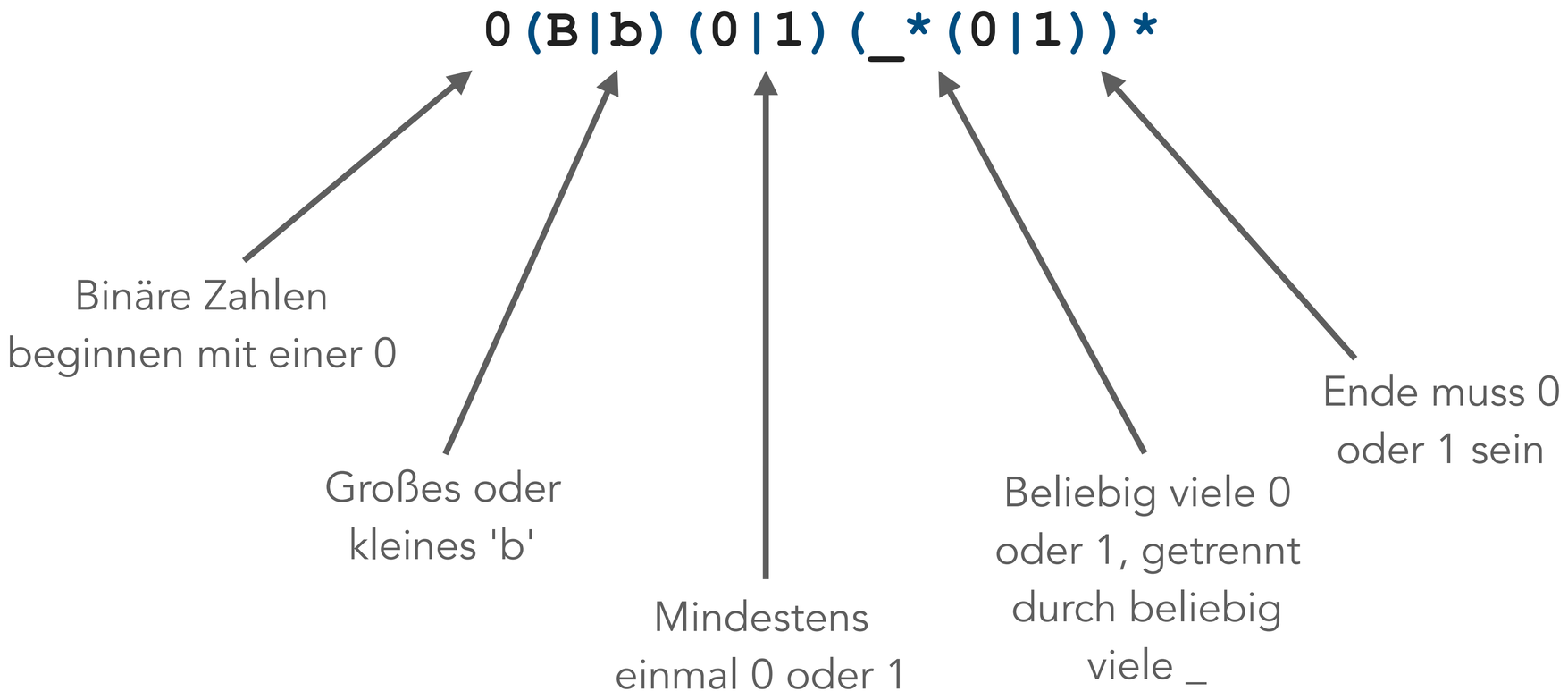
```
int dez = 42;
int bin = 0b10_1010;  //0B10_1010
int hex = 0x2A; int okt = 052;
```

Gebe reguläre Ausdrücke an für Binär, Hexadezimal, Oktal

- Alternativen mit |
- Iteration mit * (beliebig oft, inklusive 0-mal)
- Konkatenation *<durch hintereinander aufschreiben>*
- Optionen mit ?

P03.2

Binärzahl:



P03.2

Oktalzahl:

```
0(_|0|1|2|3|4|5|6|7)*(0|1|2|3|4|5|6|7)
```

Hexadezimalzahl:

```
0(x|X)(hNumbers)(_*(hNumbers))*
hNumbers ::= (0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F)
```

P03.3

Palindrom: sinnvolle Folge von Buchstaben, Wörtern oder Zahlen, die vorwärts- wie rückwärtsgelesen [den gleichen] Sinn ergeben.

1. Länge der Zahl bestimmen
2. Zahl in Array überführen
3. Auf Palindrom testen

P03.3

Länge der Zahl bestimmen:

Plausibilitätsabfrage fehlt

```
int n = readInt();
int factor = 1; int length = 0;
boolean found = false;
while(!found) {
  if(n/factor < 1)
    found = true;
  else {
    factor=factor*10;
    length=length+1;
  } } //⇐ kein guter Stil
```

P03.3

Zahl in Array überführen (v1):

```
int[] palindrom = new int[length];
factor = 10;
int pos = length-1;
while(pos >= 0) {
  palindrom[pos--] = (n % factor)/(factor/10);
  n = n-(n%factor);
  factor=factor*10;
}
```

P03.3

Zahl in Array überführen (v2, in Übung besprochen):

```
int[] palindrom = new int[length];
int pos = length - 1;
factor /= 10;
while(factor >= 1) {
  palindrom[pos] = n/factor;
  n -= palindrom[pos]*factor;
  factor/=10;
  pos--;
}
```

P03.3

Auf Palindrom testen:

```
boolean isPalindrom = true;
int front = 0;
int back = length-1;
while(front < back) {
  if(palindrom[front] != palindrom[back])
    isPalindrom = false;
  front++; back--;
}
write(""+isPalindrom);  //gibt nur true/false aus
```

P03.4

Pascal'sches Dreieck

- Die Anzahl der Elemente von Zeile *n* ist *n* + 1.
- Die erste und letzte Zahl jeder Zeile ist stets die 1.
- Das *i*-te Element der Zeile n entspricht der Summe des *i*-ten und des (*i* - 1)-ten Elements der Zeile (*n* - 1).