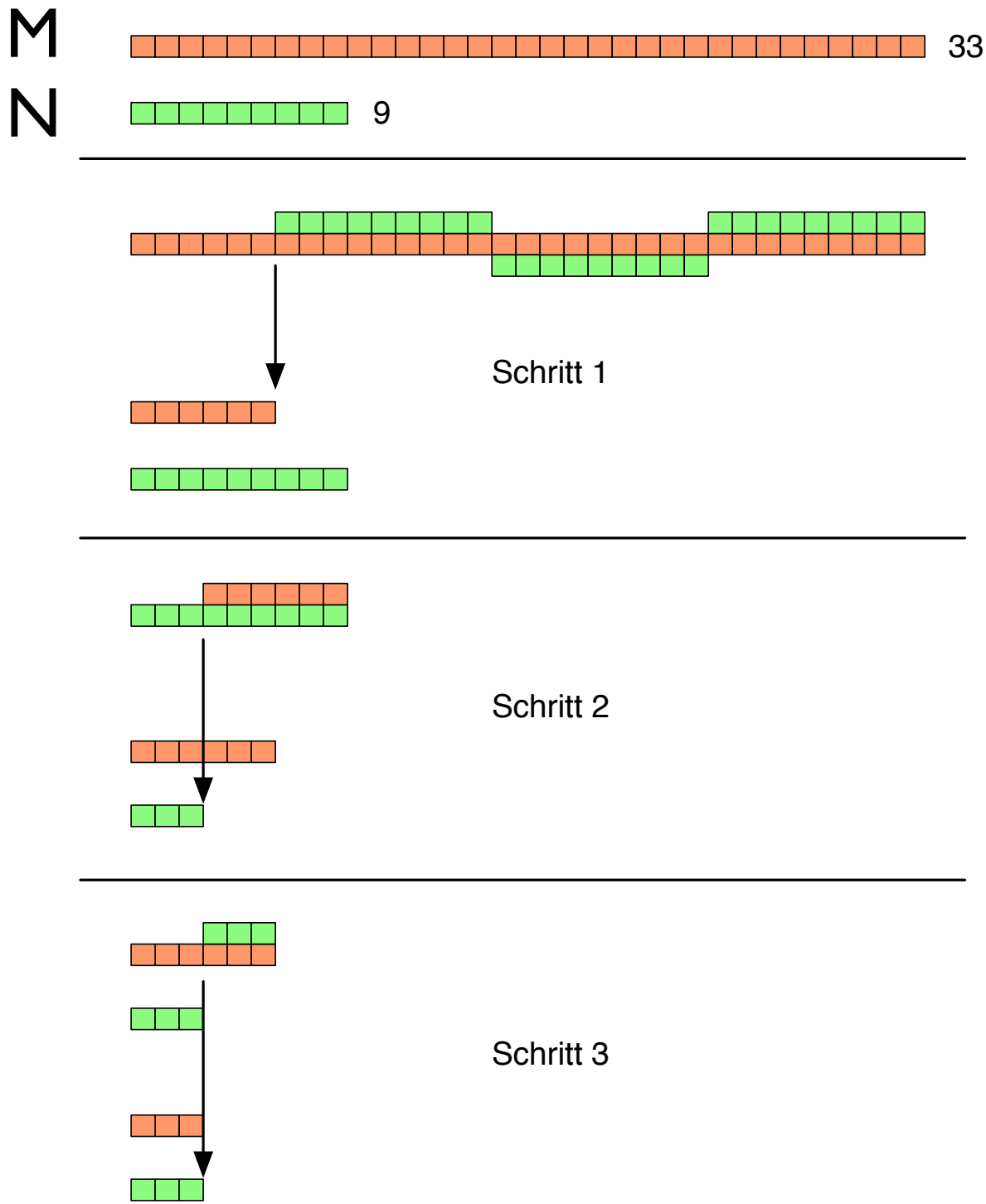
Kapitel 2: Grundlegende Sprachkonzepte

Algorithmus

- ursprüngliche Bedeutung:
  Berechnungsvorschrift in der Mathematik
- moderne Bedeutung:
  eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems (wie ein Kochrezept)

Euklidischer Algorithmus

”...Wenn N Teiler von M ist, ist N, weil auch Teiler von sich selbst, gemeinsamer Teiler. N ist dann auch der größte Teiler, denn größer als N kann ein Teiler von N nicht sein. *Wenn N nicht Teiler von M ist, subtrahiert man, von den beiden Zahlen M und N ausgehend, immer die kleinere von der größeren bis die entstandene Zahl Teiler der ihr vorhergehenden ist, der dann der größte gemeinsame Teiler von M und N ist. . . .*”


M
33
N
9
Schritt 1
Schritt 2
Schritt 3

Euklidischer Algorithmus

Solange M ungleich N ist, wiederhole:

Wenn M größer als N ist, dann:

Ziehe N von M ab und weise das Ergebnis M zu.

Sonst (das heißt, N ist größer als M):

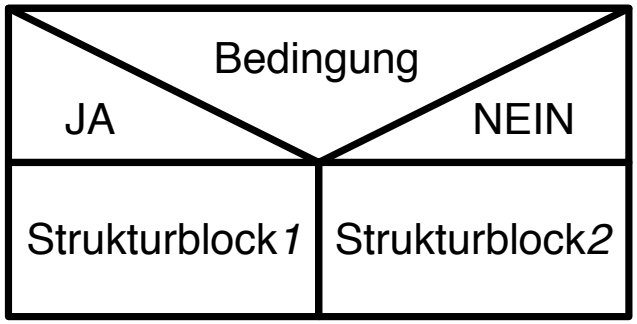
Ziehe M von N ab und weise das Ergebnis N zu.

(Nun ist M gleich N)

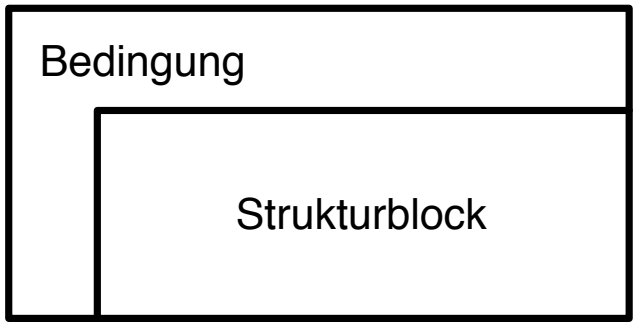
M bzw. N ist der größte gemeinsame Teiler.

Nassi-Schneidermann Diagramm

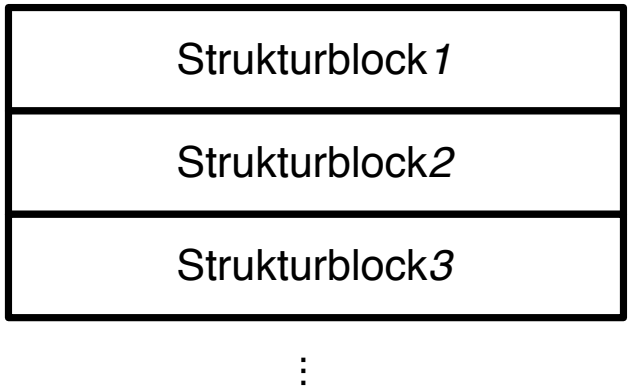
Strukturblock: Verzweigung



Strukturblock: Wiederholung



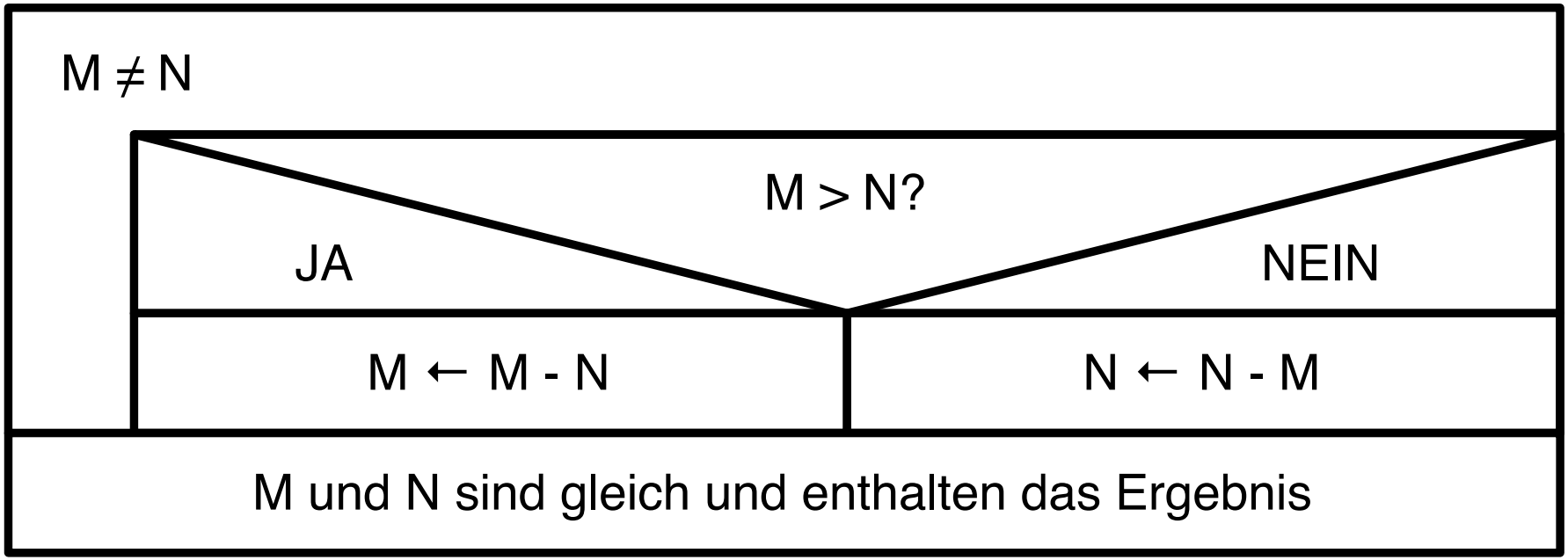
Strukturblock: Folge



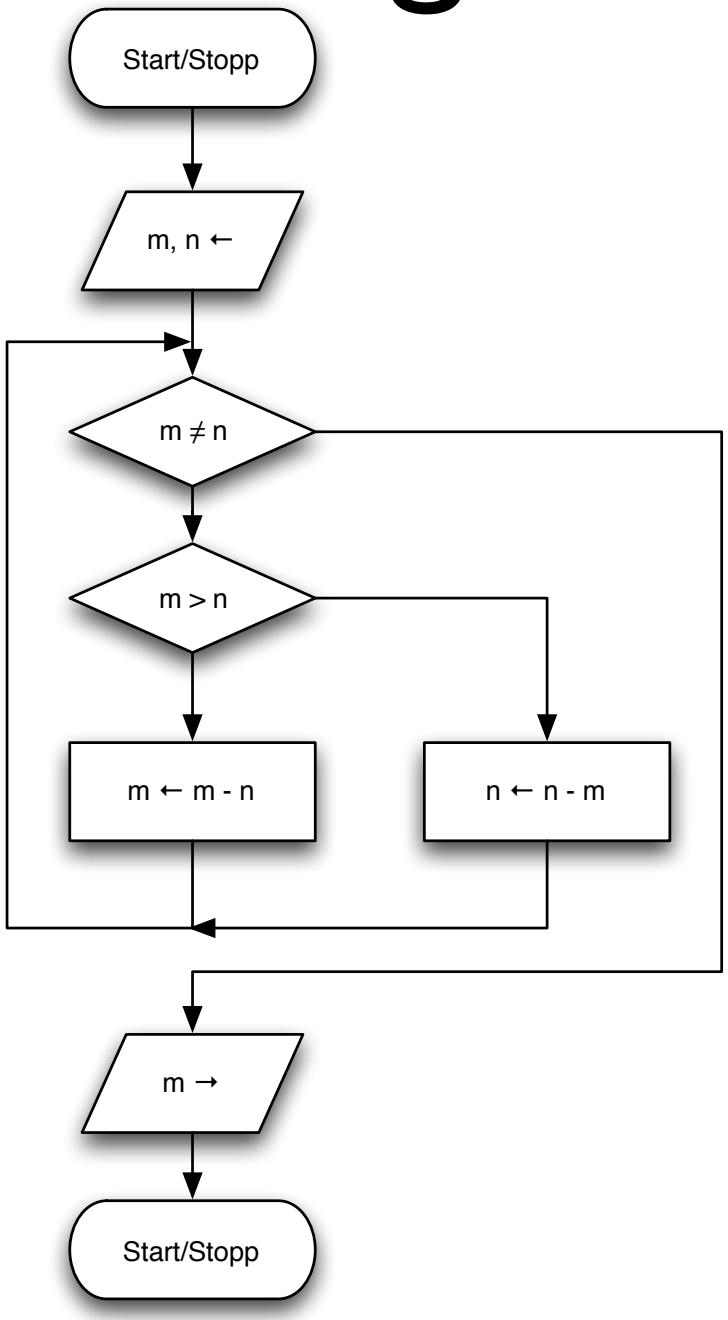
⋮

Strukturblock: Anweisung

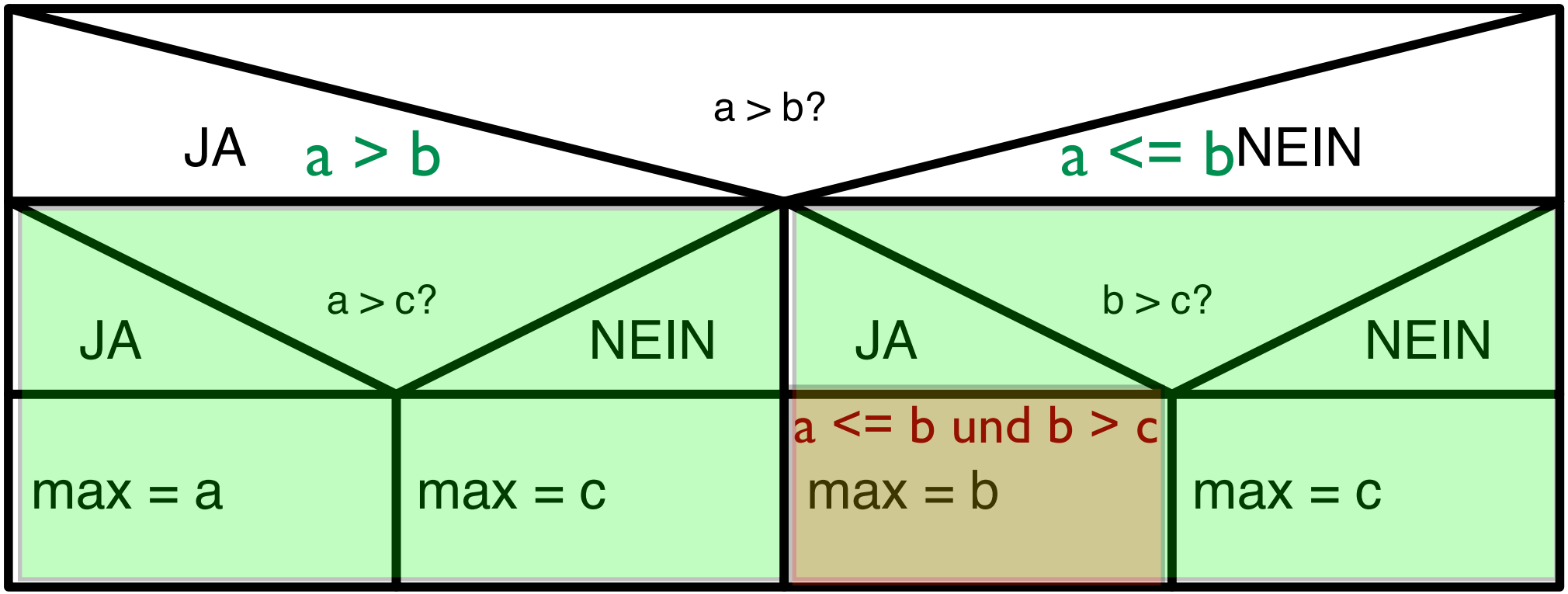
Euklidischer Algorithmus

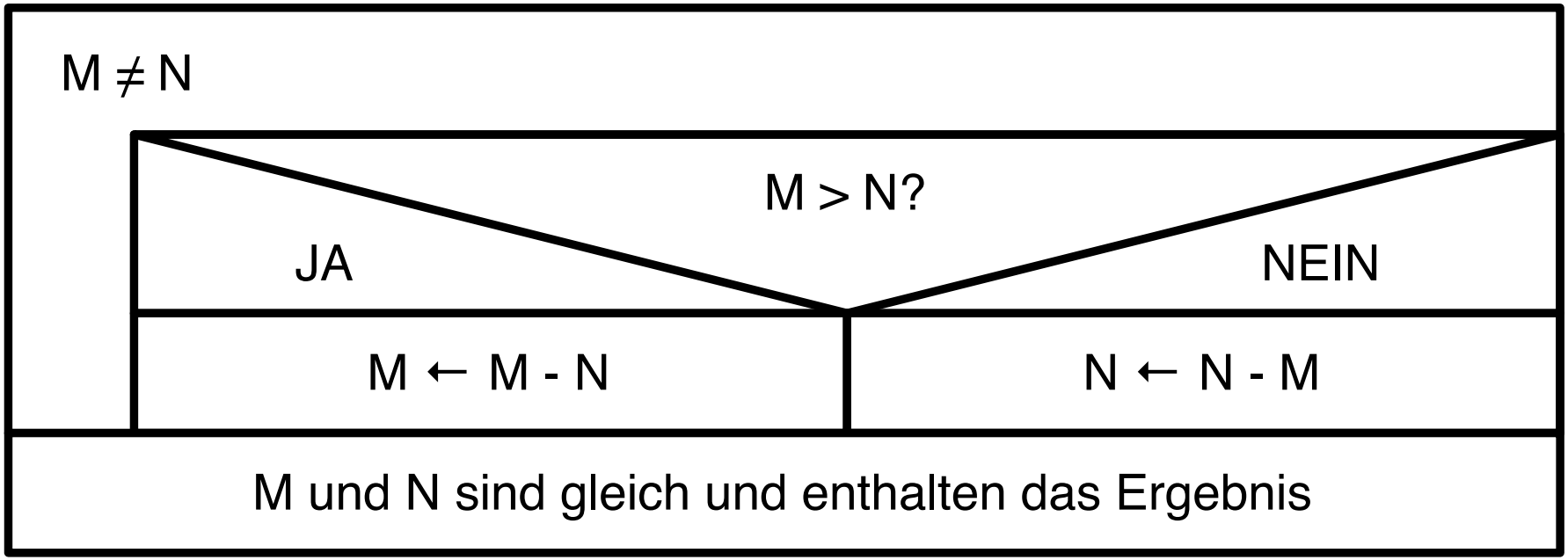


Flussdiagramm



Maximum von 3 Zahlen






```
while (m != n)
   if (m > n)
      m = m - n;
   else  // es gilt n > m
      n = n - m;

// nun gilt m == n
System.out.println(m);
```

Variable

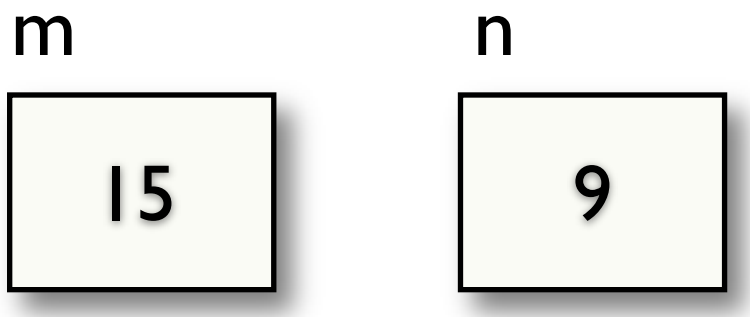
(steht für einen änderbaren Wert)

ist ein benannter Speicherbereich, der genau einen Wert speichert

Wertzuweisung mit „=“

```
m = 33;
n = 9;
m = m - n;
m = m - n;
```

Ausdruck, liefert Wert



Welchen Wert hat a nach dieser Anweisungsfolge?

```
b = 0;
a = 1;
b = 2 + a;
a = a + b;
```

4

Welchen Wert hat a nach dieser Anweisungsfolge?

```
b = 0;
a = 1;
a = a + b;
b = 2 + a;
```

1

Datentyp

bestimmt eine Menge von Werten (o. Objekten) mit darauf anwendbaren Operationen

Typbezeichnung	Beschreibung	Bereich	Operationen	Literale (Beispiel)
`int`	ganze Zahlen	-2^{31} bis $2^{31}-1$	+ - * / % etc.	23 , 15 , 4
`char`	Zeichen	alle 16-bit Unicode Zeichen	+ - * / % etc.	'a','E','3' '+'
`double`	Fließkommazahlen	ca. -1.8E308 bis 1.8E308	+ - * / % etc.	31.45 0.3145E2
`boolean`	Wahrheitswert	WAHR / FALSCH	boolesche, logische	true false
`String`	Zeichenketten		+ (Verkettung) etc.	"true" "hello\nbye"

(einige Beispiele für Datentypen in Java)

Instanz

Ein Element der Wertemenge eines Datentyps

Beispiel:

Die Zahl `31.45` ist Instanz des Datentyps `double`

Das Zeichen `'a'` ist Instanz des Datentyps `char`

Literal

Symbol zur direkten Darstellung eines Werts
(einer Instanz eines Basistyps)

```
23          31.45          'a'     "first things first"
15        0.3145E2         'X'     "hello!\nbye!"
 4         .3145E2         '+'     "23"
           3145E-2         'ß'
              14D          '\n'
              14d          '<'
```

arithmetische Ausdrücke

liefern einen *Rückgabewert* als Ergebnis
ihrer *Auswertung*

`4`	`4`
`3+4`	`7`
`3-4+1`	`0`
`2.5 * 5D + 7.0 * 2.1`	`27.200000000000003`
`a - b + 2`	`a - b + 2`

arithmetische Operatoren

Operator

3+4

Operanden

Zweistelliger Operator (hat 2 Operanden)

Infix Operator (Operator steht zwischen den Operanden)

arithmetische Operatoren

`+`	Addition	`1+5` → `6`
`-`	Subtraktion	`4-2` → `2`
`*`	Multiplikation	`3*4` → `12`
`/`	Division	`5/2` → `2`
`%`	Restwertoperator (Modulus)	`5%2` → `1`

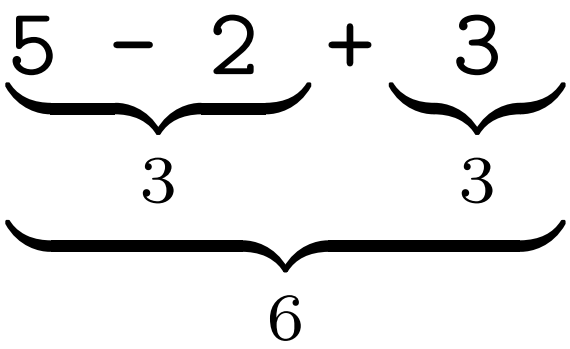
Priorität von arithm. Operatoren

„Punktrechnung geht vor Strichrechnung“

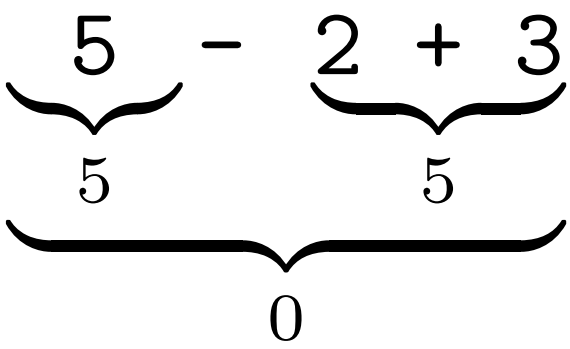
```
2.5 * 5D + 7.0 * 2.1
```

1. 2. 1.

Assoziativität von Operatoren



linksassoziativ



rechtsassoziativ

++ und --

Einstellige Präfix oder Postfix Operatoren, die den Wert einer Variable im eins erhöhen bzw. veringern

```
i = 5;
j = i++; //i == 6, j == 5
```

Anweisung und Ausdruck zugleich

```
i = 5;
j = ++i; //i == 6, j == 6
```

```
i = 5;
j = i--; //i == 4, j == 5
```

Auswertungsreihenfolge

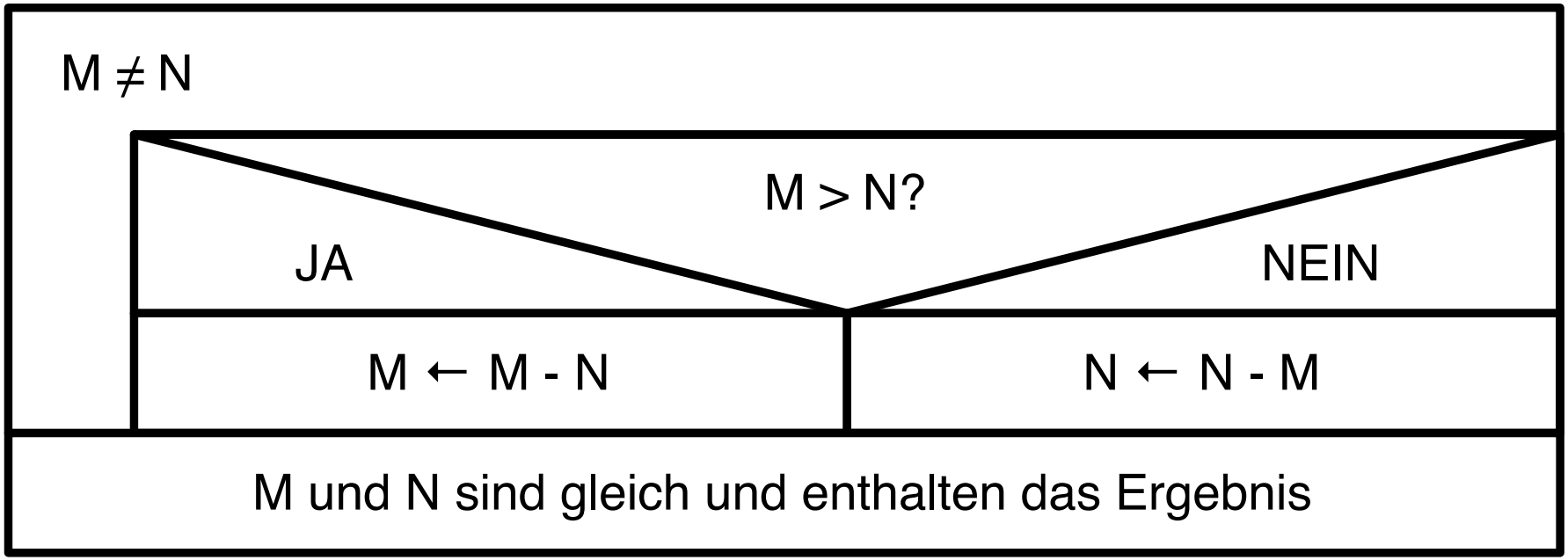
(bezieht sich auf die Operanden eines Operators)

➪ generell von links nach rechts

Welchen Wert liefert folgender Ausdruck?

```
i++ - i
```

Euklidischer Algorithmus



Boolesche Operatoren

Operanden: Zahlen
Rückgabewert: Wahrheitswert

Zweistellige Infix Operatoren:

`>`	größer	`1>5`	→	`false`
`>=`	größergleich	`4>=4`	→	`true`
`<`	kleiner	`3<4`	→	`true`
`<=`	kleinergleich	`1<=2`	→	`true`
`==`	ist gleich	`5==2`	→	`false`
`!=`	ist nicht gleich	`5!=2`	→	`true`

Operatoren	Priorität	Assoziativität	Bedeutung
`[]`	1	links	Arrayzugriff
`()`	1	links	Methodenaufruf
`.`	1	links	Komponentenzugriff
`++, --`	1	links	Postinkrement, Postdekrement
`++, --`	2	rechts	Präinkrement, Prädekrement
`+, -`	2	rechts	unäres Plus und Minus
`~`	2	rechts	bitweises Komplement
`!`	2	rechts	logisches Komplement
`(Typ)`	3	rechts	Cast
`new`	3	rechts	Erzeugung (siehe Kapitel 3)
`*, /, %`	4	links	Multiplikation, Division, Rest
`+, -`	5	links	Addition und Subtraktion
`+`	5	links	Stringverkettung
`<<`	6	links	Linksshift
`>>`	6	links	Rechtsshift mit Vorzeichenerweiterung
`>>>`	6	links	Rechtsshift ohne Vorzeichenerw.
`<, <=, >, >=`	7	links	numerische Vergleiche
`instanceof`	7	links	Typvergleich (siehe Kapitel 3)
`==, !=`	8	links	Gleich-/Ungleichheit
`&`	9	links	bitweises/logisches UND
`^`	10	links	bitweises/logisches exklusives ODER
`|`	11	links	bitweises/logisches ODER
`&&`	12	links	logisches konditionales UND
`||`	13	links	logisches konditionales ODER
`?:`	14	rechts	Bedingungsoperator
`=`	15	rechts	Zuweisung
`*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=`	16	rechts	kombinierter Zuweisungoperator

Aufgabe

Entwerfen Sie einen Algorithmus mit folgender Spezifikation:

Eingabe: Zwei Zahlen `a` und `b`
Ausgabe:

- Ergebnis der ganzzahligen Division `a/b`
- Restwert `a%b`

Bedingungen: Der Algorithmus soll ohne die Operatoren `/` und `%` auskommen.

Aufgabe

Entwerfen Sie einen Algorithmus mit folgender Spezifikation:

Eingabe: Zwei Zahlen `a` und `b`
Ausgabe:

- Produkt `a*b`

Bedingungen: Der Algorithmus soll ohne die Operatoren `*`,`/` und `%` auskommen.

Scanner

```
Scanner sc = new Scanner(System.in);

int ganzzahlEingabe;
String textEingabe;
double x;

ganzzahlEingabe = sc.nextInt();
textEingabe = sc.next();
x = sc.nextDouble();
```

Einlesen von Zeichenketten

`next()` liest ein Wort bis zum nächsten Trennzeichen (Leerraum) ein, ohne dieses zu „verbrauchen“

`nextLine()` liest ein Wort bis zum nächsten Zeilenumbruchszeichen ein und verbraucht dieses

Eingabe: `Hello world!\n` ←Ende der Eingabe

```
String a = sc.next();          //“Hello“
String b = sc.nextLine();      //“ world!“
```

Einlesen von Zeichenketten

`next()` liest ein Wort bis zum nächsten Trennzeichen (Leerraum) ein, ohne dieses zu „verbrauchen“

`nextLine()` liest ein Wort bis zum nächsten Zeilenumbruchszeichen ein und verbraucht dieses

Eingabe: `Hello\nworld!\n`

```
String a = sc.next();          //“Hello“
String b = sc.next();          //“world!“
```

Einlesen von Zeichenketten

`next()` liest ein Wort bis zum nächsten Trennzeichen (Leerraum) ein, ohne dieses zu „verbrauchen“

`nextLine()` liest ein Wort bis zum nächsten Zeilenumbruchszeichen ein und verbraucht dieses

Eingabe: `Hello\nworld!\n`

```
String a = sc.next();        //“Hello“
String b = sc.nextLine();    //““
```