Lektion 2, Teil 1: Java-Kontrollstrukturen (2)

Programmiersprachen sind formale Sprachen

- Schreibweisen von Elementen einer Programmiersprache müssen exakt definiert sein.
- Programmiersprachen besitzen ähnlich wie natürliche Sprachen Grammatiken.
- Die Grammatiken sind bei Programmiersprachen durch formale Regeln (sogenannte Produktionsregeln) definiert.
- Die Syntax (siehe nächste Folie) wird meist durch eine kontextfreie Grammatik festgelegt.

Syntax, Semantik

- Die Syntax einer (Programmier-) Sprache regelt die erlaubten Kombinationen von Sprachelementen

Fragment aus `NumberOfDigits`

```
//"Zubereitung"
n = reader.readInt();
d = 1;

while(n>9 {
  n = n / 10;
  d = d + 1;
}
System.out.println(d);
```

- Finden Sie den syntaktischen Fehler ?

Syntax, Semantik

- Die Semantik regelt die Bedeutung eines Programmteils

Fragment aus `NumberOfDigits`

```
//"Zubereitung"
n = reader.readInt();
d = 1;

while(n>9); {
  n = n / 10;
  d = d + 1;
}
System.out.println(d);
```

- Finden Sie den semantischen Fehler ?

Syntax, Semantik

- Die Semantik regelt die Bedeutung eines Programmteils

Fragment aus `NumberOfDigits`

```
//"Zubereitung"
n = reader.readInt();
d = 1;

while(n>=9) {
  n = n / 10;
  d = d + 1;
}
System.out.println(d);
```

- Finden Sie den semantischen Fehler ?

Syntaxfehler und Semantikfehler

- Syntaxfehler: Zeigen sich beim Compilieren
- Semantikfehler: Zeigen sich zur Laufzeit des Programmes

Extended Backus-Naur Form (EBNF)

- Die EBNF ist eine Menge von **Produktionen** (Ersetzungregeln), in denen links ein Symbol und rechts Symbolfolgen stehen, durch die das Symbol ersetzt werden darf.
- Die Extended Backus-Naur Form (EBNF) definiert eine Grammatik

lexikalische Struktur: IntLiteral

IntLiteral ⇒ [*Sign*] *Digit*+
Digit ⇒ "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
Sign ⇒ "+" | "-"

```
-107
42
```

Extended Backus-Naur Form (EBNF)

lexikalische Struktur: IntLiteral

IntLiteral ⇒ [*Sign*] *Digit*+

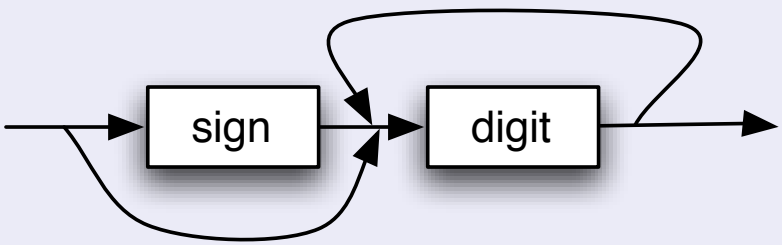
Digit ⇒ "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Sign ⇒ "+" | "-"

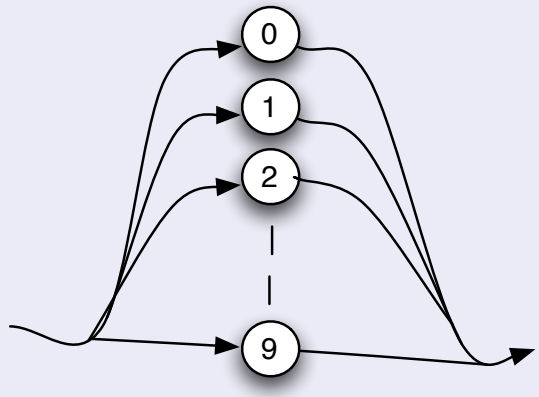
```
-107
42
```

Syntaxdiagramm

IntLiteral



Digit



Sonderzeichen einer EBNF

Wir verwenden folgende Notation[1]

- [...] Symbole zwischen eckigen Klammern sind optional und dürfen einmal vorkommen oder auch weggelassen werden.
- ...$*$ Symbole mit nachfolgendem Stern können beliebig oft wiederholt werden, oder können auch weggelassen werden
- ...$+$ Symbole mit nachfolgendem Pluszeichen müssen mindestens einmal vorkommen und können beliebig oft wiederholt werden
- (...) Gruppieren von Symbolen
- | Trennzeichen für die Alternativen in einer Produktionsregel (siehe oben).

[1] In der Java Sprachbeschreibung auf http://java.sun.com/docs/books/jls/third_edition/html/syntax.html#18.1 wird eine alternative aber gleichmächtige Notation mit " ... " benutzt.

Namen für Variablen (engl. Identifier)

Variablennamen müssen mit einem Buchstaben beginnen und dürfen nur Buchstaben, Zahlen und die Sonderzeichen "$" und "_" enthalten.

Identifier

Letter ⇒ "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i"
| "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r"
| "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
| "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I"
| "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R"
| "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "$" | "_"

Identifier ⇒ *Letter* (*Letter* | *Digit*)*

Produktionsregeln für numerische Literale

Literal	$\Rightarrow$	*IntLiteral* | *DoubleLiteral* | *StringLiteral* | *CharLiteral*
IntLiteral	$\Rightarrow$	[*Sign*] *Digit+*
DoubleLiteral	$\Rightarrow$	[*Sign*] *Digit+* "." *Digit** [*Exponent*] [*DoubleSuffix*]
DoubleLiteral	$\Rightarrow$	[*Sign*] *Digit** "." *Digit+* [*Exponent*] [*DoubleSuffix*]
DoubleLiteral	$\Rightarrow$	[*Sign*] *Digit+* *Exponent* [*DoubleSuffix*]
DoubleLiteral	$\Rightarrow$	[*Sign*] *Digit+* [*DoubleSuffix*]
Exponent	$\Rightarrow$	("E" | "e") [*Sign*] *Digit+*
DoubleSuffix	$\Rightarrow$	"D" | "d"

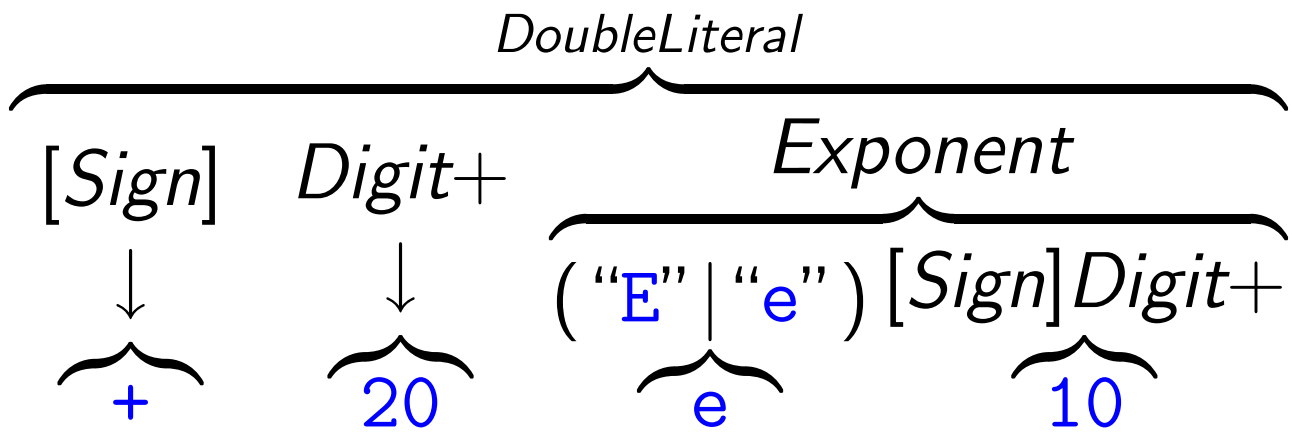
`+20e10`

Beispiel

Literal	gültig?	Literal	gültig?
`10d`	✓	`20.e10d`	✓
`.1e2`	✓	`20.78e-30D`	✓
`10D`	✓	`10e20e-10`	nein
`10e1`	✓	`20e`	nein

EBNF Ableitungsbaum

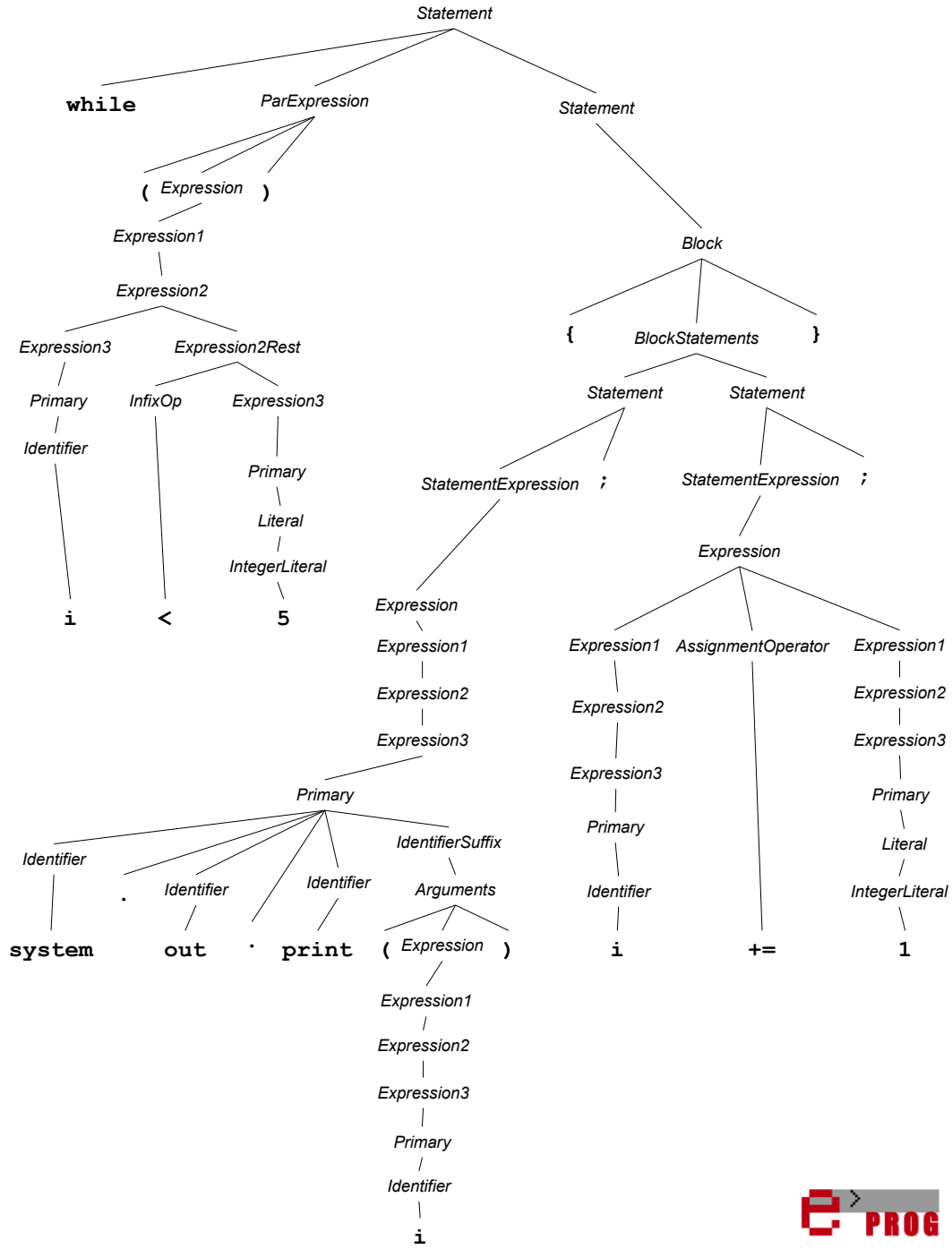
+20e10



Beispiel (EBNF Baum)

Der EBNF Ableitungsbaum einer komplexeren Anweisung ...

```
while ( i <= 5 ) {
  system.out.print(i);
  i+=1;
}
```



EBNF für Anweisungen in Java

Eine Anweisung kann sein:

- Definitionen von Variablen
- Zuweisungen
- ... (folgt später)

Statement	$\Rightarrow$	*Definition* | *Assignment*
Definition	$\Rightarrow$	["`final`"] *Type Identifier* ["`=`" *Expression*] "`;`"
Type	$\Rightarrow$	"`int`" | "`double`" | "`char`" | "`String`"
Assignment	$\Rightarrow$	*Identifier* "`=`" *Expression* "`;`"

Beispiel

```
final double PI = 3.14159;
double radius = 2;
double area;
area = radius * radius * PI;
boolean validRadius = radius >= 0;
```

EBNF für Ausdrücke in Java

Expression	⇒	*DoubleLiteral* | *IntLiteral* | *Identifier*	
Expression	⇒	*Expression BinaryOperator Expression*	`rest + 1`
Expression	⇒	*UnaryOperator Expression*	`-(2+3)`
Expression	⇒	"`(`" *Expression* "`)`"	`(2+3)`
UnaryOperator	⇒	"`+`" | "`-`"	
BinaryOperator	⇒	"`+`" | "`-`" | "`*`" | "`/`" | "`%`"	
IncDecOperator	⇒	"`++`" | "`--`"	
Expression	⇒	*IncDecOperator Identifier*	`++i`
Expression	⇒	*Identifier IncDecOperator*	`i++`
Expression	⇒	"`(`" *Type* "`)`" *Expression*	`(int) 2.5`
Type	⇒	"`int`" | "`double`" | "`char`"	

Wiederholung: Werte und Typen, Ergänzung `boolean`

Definition (Literal)

Ein Literal ist die **Bezeichnung eines Wertes** in einer Programmiersprache

Wert	Literal in Java
10	`10`
7	`7`
21.2	`21.2`
3.1415	`3.1415`
a	`'a'`
programming is fun	`"programming is fun"`
WAHR	`true`
FALSCH	`false`

Boole'sche Ausdrücke

Ausdrücke, die Wahrheitswerte liefern

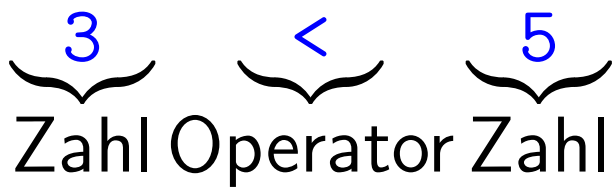
(George Boole, Mathematiker, 1815 – 1864)

Beispiel

Ausdruck	Wert
2<5	WAHR
2>5	FALSCH
2>=2	WAHR

Boole'sche Operatoren

- Operanden: **Zahlenwerte** (numerische Literale, Variablen, Ausdrücke)
- Ergebnis: **Wahrheitswert**



Boole'sche Operatoren

`<`	kleiner
`<=`	kleiner oder gleich
`>`	größer
`>=`	größer oder gleich
`==`	ist gleich
`!=`	ist nicht gleich

Weitere Beispiele für Boole'sche Ausdrücke

```
int i;
char ch;
i = 98;
ch = 'b';
```

Ausdruck	Wert
`'a' < 'b'`	`true`
`ch == 'b'`	`true`
`ch == 'a'`	`false`
`ch == i`	`true`

Priorität von Boole'schen Operatoren

- Boole'sche Operatoren binden schwächer als arithmetische Operatoren

Ausdruck	Wert
`4 + 3 < 8`	`true`
`14 != 20 - 6`	`false`

Variablen vom Typ `boolean`

Einer `boolean`-Variable kann ein `boolean`-Wert zugewiesen werden

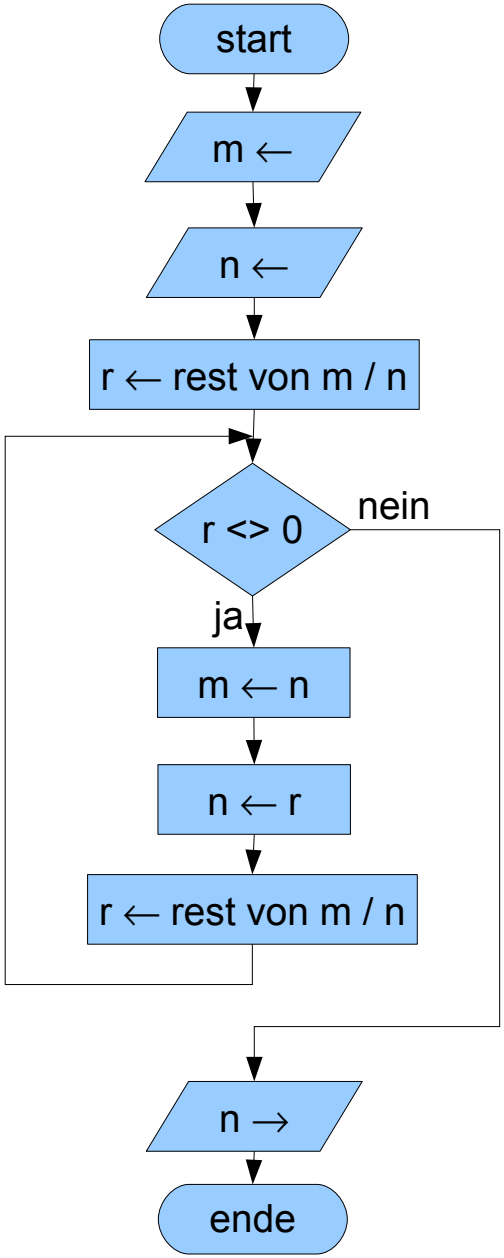
Beispiel

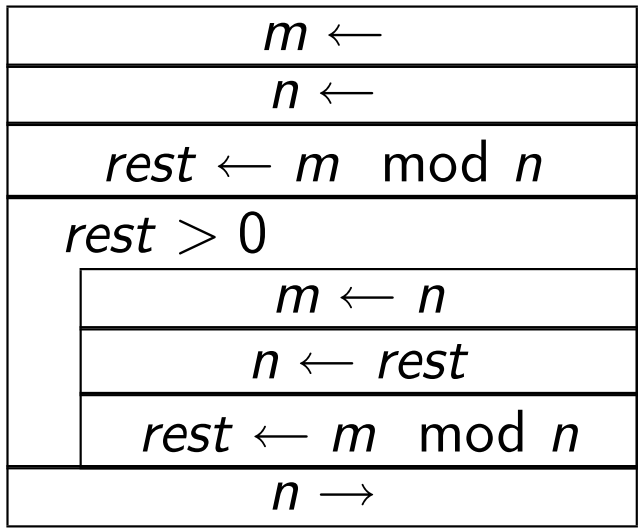
```
boolean angemeldet;
angemeldet = true;

boolean validTemperature = celcius > -273.16;
```

Kontrollstrukturen in Java

- Verzweigungen: `if`-**Anweisung**
- **Anweisungsblöcke**
- Wiederholungen (Schleifen):
  - `while`-**Anweisung**
  - `for`-**Anweisung**
- ...

Wiederholung: GGT - Euklidischer Algorithmus

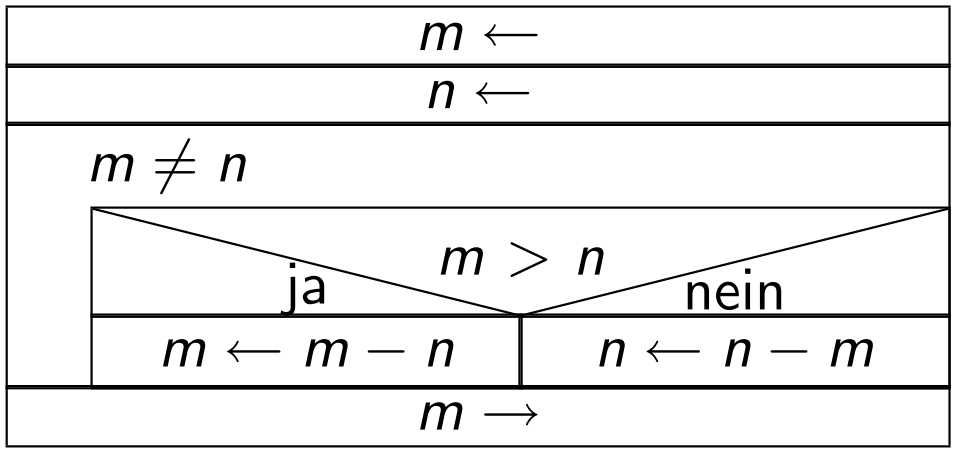




m	n	r
28	20	8
20	8	4
8	4	0

m	n	r
100	45	10
45	10	5
10	5	0

Beispiel: GGT Differenz Algorithmus



m	n
28	20
8	20
8	12
8	4
4	4

m	n
100	45
55	45
10	45
10	35
10	25
10	15
10	5
5	5

- **Äquivalent** zum Euklidischen Alg.: gleiches Ein-/Ausgabeverhalten
- **Effizienz**: Euklidischer Alg. ist effizienter (weniger Schritte notwendig, unabhängig von Programmiersprache)

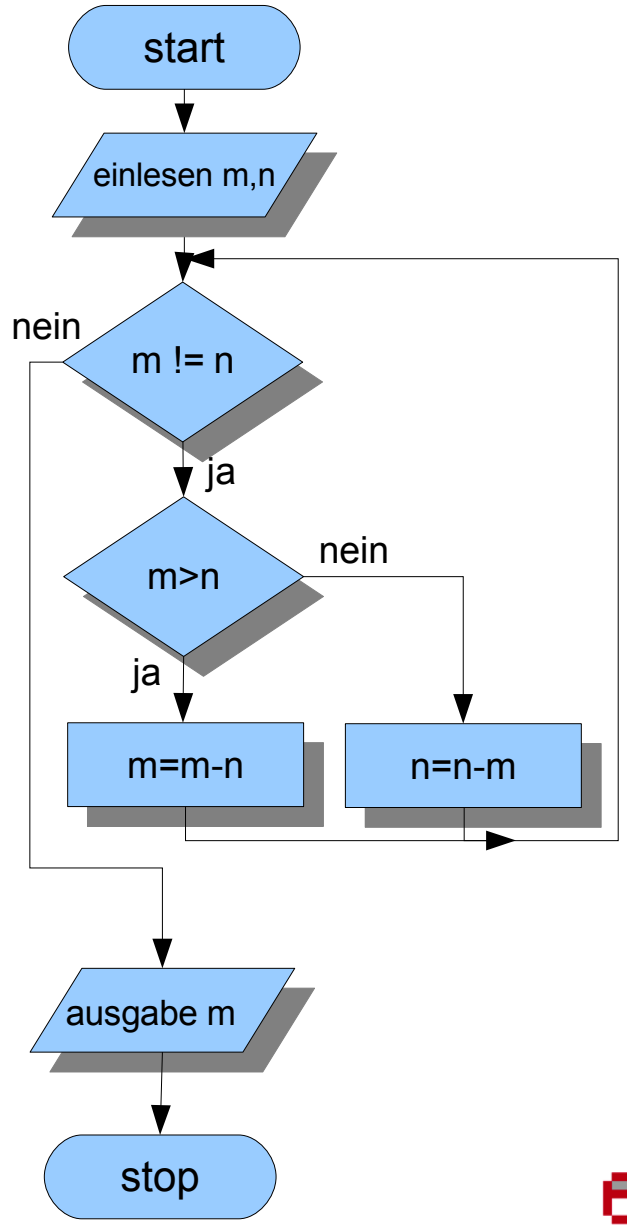
GGT - Differenz Algorithmus

$m \leftarrow$

$n \leftarrow$

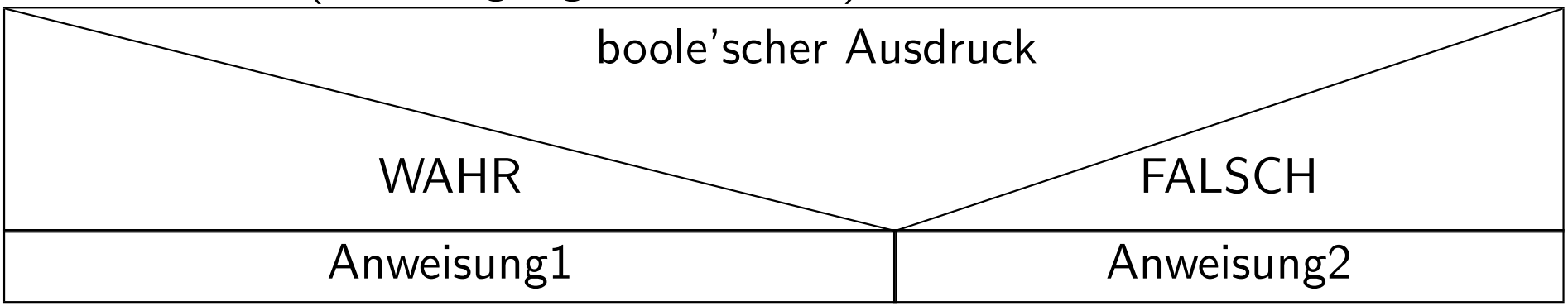
```
while (m!=n)
    if (m>n)
        m = m - n;
    else
        n = n - m;
```

$m \rightarrow$



`if`-Anweisung mit `else`-Teil

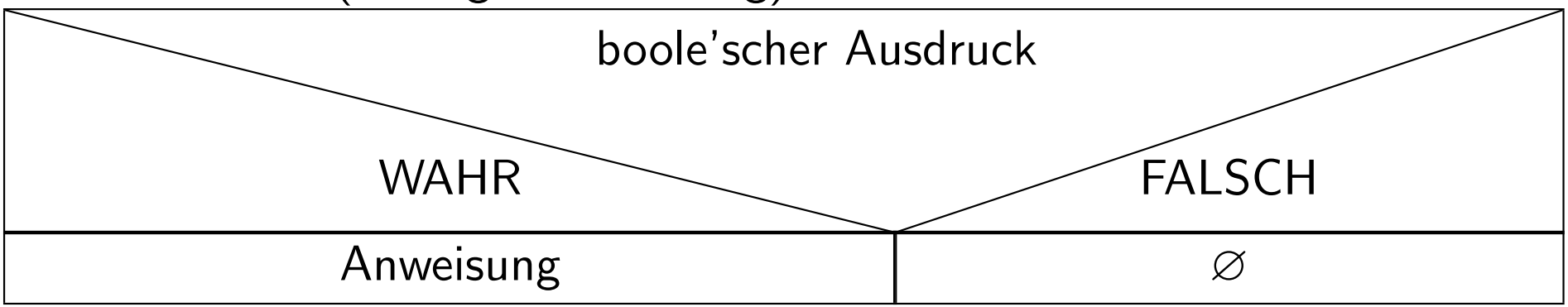
mit `else`-Teil (Verzweigung, Alternative):




```
if ( boole'scher Ausdruck )
    Anweisung1
else
    Anweisung2
```

`if`-Anweisung ohne `else`-Teil

ohne `else`-Teil (bedingte Anweisung)



`if` `(` *boole'scher Ausdruck* `)`
    Anweisung

if-Anweisung: Beispiel

In einer Firma soll für Überstunden 1.5 mal soviel Lohn bezahlt werden...

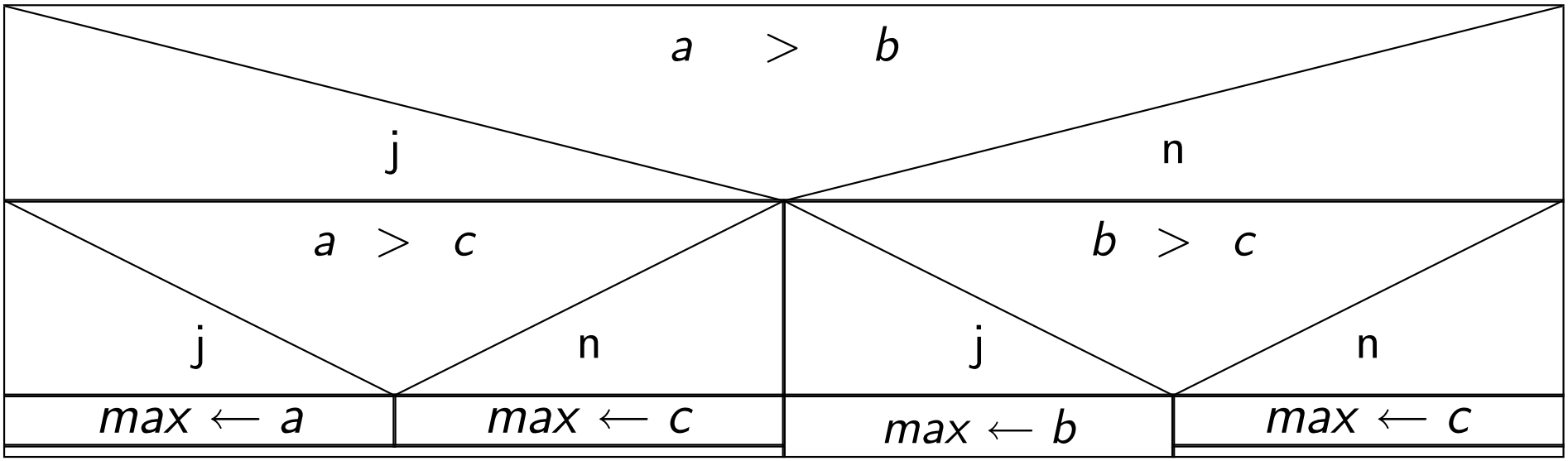
```
if (wochenstunden > 40)
   gehalt = stundenlohn*40
            + 1.5*stundenlohn*(wochenstunden - 40);
else
   gehalt = stundenlohn*wochenstunden;
```

if-Anweisung: Beispiel

Die if-Anweisung muss nicht immer einen else-Teil haben:

```
bonus = 0;
gehalt = wochenstunden * stundenlohn;
if (wochenstunden > 40)
    bonus = 0.5*stundenlohn*(wochenstunden - 40);
gehalt = gehalt + bonus;
```

Beispiel: Maximum von 3 Werten



Beispiel: Maximum von 3 Werten

```
if(a > b)
    if(a > c)
        max = a;
    else
        max = c;
else
    if(b > c)
        max = b;
    else
        max = c;
```

Anweisungsblöcke

- Fassen eine Sequenz von Anweisungen mit { und } zusammen
- Können eingesetzt werden, wo eine einzelne Anweisung erwartet wird

Statement ⇒ "{" *Statement* "}"

```
if (  bool'scher Ausdruck )
     Anweisung1
else
     Anweisung2
```

```
if ( radius >= 0 ) {
    area = radius * radius * PI;
    System.out.print("Die Fläche des Kreises mit Radius ");
    System.out.print(radius);
    System.out.print(" ist ");
    System.out.println(area);
} else
        System.out.println("Eingabefehler!");
```

Problem des "dangling-else"

Zu welchem `if` gehört das `else` in diesem Beispiel?

`if` `(` *Bedingung 1* `)` `if` `(`*Bedingung 2* `)` *Anweisung 1* `else` *Anweisung 2*

```
if ( Bedingung 1 )
    if ( Bedingung 2 )
        Anweisung 1
    else
        Anweisung 2
```

✓

```
if ( Bedingung 1 )
    if ( Bedingung 2 )
        Anweisung 1
else
    Anweisung 2
```

Einrückung entspricht hier NICHT der Interpretation durch den Compiler

- `else` bezieht sich immer auf das **textuell letzte freie** `if`
- Einrückungen dienen der Übersichtlichkeit für Menschen, werden aber vom Compiler **nicht berücksichtigt**

Explizite Zuordnung durch Block-Klammern

```
if (   Bedingung 1 ) {
    if (   Bedingung 2 )
           Anweisung 1
    else
           Anweisung 2
}
```

```
if (   Bedingung 1 ) {
    if (   Bedingung 2 )
           Anweisung 1
}
else
       Anweisung 2
```

Logische Operatoren

- akzeptieren `boolean`-Werte (Ausdrücke, Variablen, Wahrheitsliterale) als Operanden
- liefern als Ergebnis einen `boolean`-Wert

Logische Operatoren

`&&`	logisches Und
`||`	logisches Oder
`^`	logisches exklusives Oder
`!`	logisches Nicht

Logische Operatoren

And

```
false  &&  false  →  false
false  &&  true   →  false
true   &&  false  →  false
true   &&  true   →  true
```

Xor

```
false  ^  false  →  false
false  ^  true   →  true
true   ^  false  →  true
true   ^  true   →  false
```

Or

```
false  ||  false  →  false
false  ||  true   →  true
true   ||  false  →  true
true   ||  true   →  true
```

Not

```
!false  →  true
!true   →  false
```

Zusammengesetzte Bedingungen

```
if ( month == 4 || month == 6 || month == 9 || month == 11 )
    days = 30;
else
    if ( month == 2 )
        days = 28;
    else
        days = 31;
```

Beispiel: Maximum von 3 Werten

```
if(a > b)
    if(a > c)
        max = a;
    else
        max = c;
else
    if(b > c)
        max = b;
    else
        max = c;
```

Beispiel: Maximum von 3 Werten

```
if(a > b && a > c)
        max = a;
if(a > b && a <= c)
        max = c;
if(a <= b && b < c)
        max = c;
...
```

Priorität von logischen Operatoren

- logische Operatoren binden schwächer als Boole'sche Operatoren

Ausdruck	Wert
`4 + 3 < 8 || false`	`true`
`14 != 20 - 6 || 2 < 2`	`false`

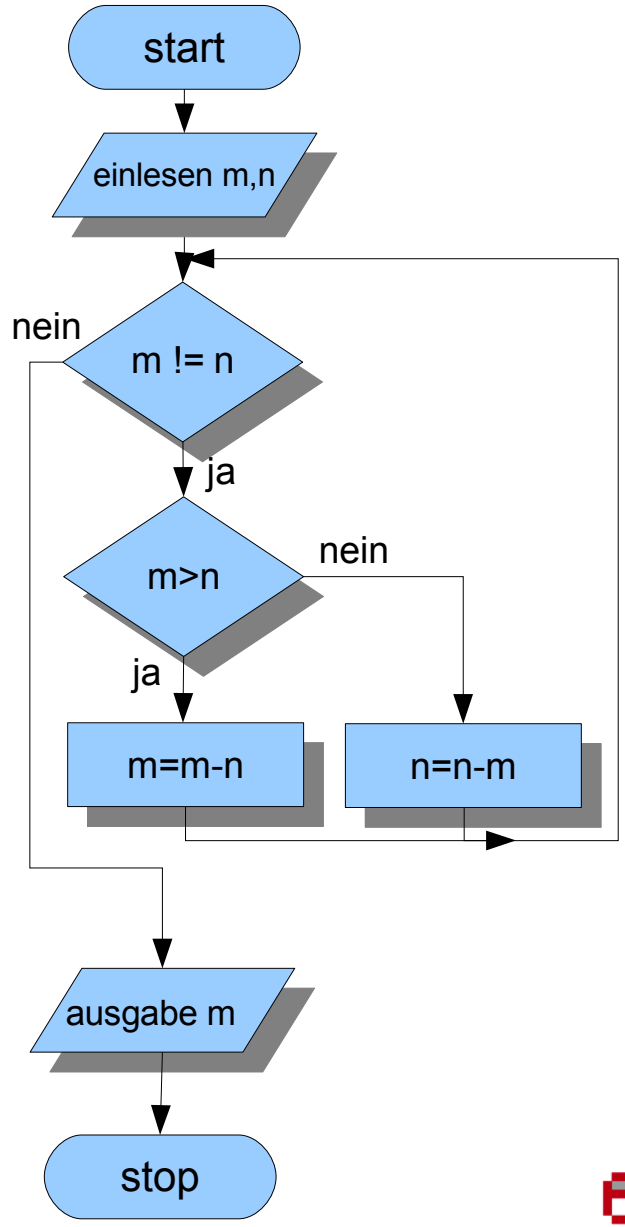
GGT - Differenz Algorithmus

$m \leftarrow$

$n \leftarrow$

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

$m \rightarrow$



Kontrollstrukturen in Java

- Verzweigungen: `if`-**Anweisung**
- **Anweisungsblöcke**
- Wiederholungen (Schleifen):
  - `while`-**Anweisung**
  - `for`-**Anweisung**
- ...

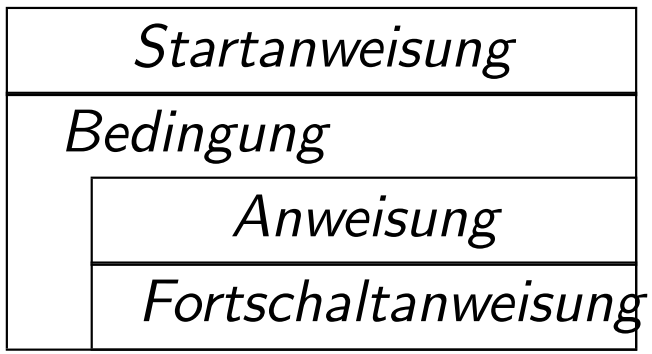
Syntax der `while`-Anweisung

`while` `(` *boole'scher Ausdruck* `)` *Anweisung*

Syntax der `for`-Anweisung

Die `for`-Anweisung

- lässt sich auch in eine `while`-Anweisung umschreiben
- eignet sich aber oft wegen der kompakten Schreibweise...




```
for ( Startanweisung;
      Bedingung;
      Fortschaltanweisung )
          Anweisung
```

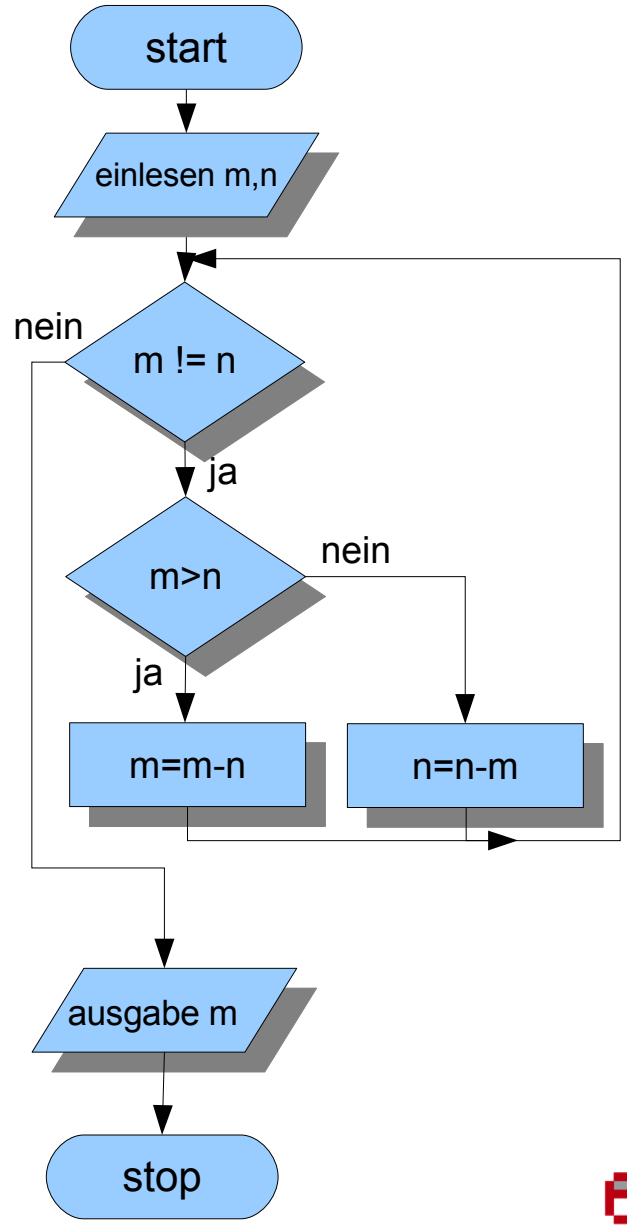
Schleifen: Die `for`-Schleife

```
int summe = 0;
int i = 1;
while (i <= n) {
    summe = summe + i;
    i = i + 1;
}
```

```
int summe = 0;
for (int i = 1; i <= n; i++) {
    summe = summe + i;
}
```

GGT: Differenz Algorithmus

```
m = reader.nextInt();
n = reader.nextInt();
while (m!=n)
    if (m>n)
        m = m - n;
    else
        n = n - m;
System.out.println(m);
```



Zusammenfassung Lektion 2

- Strukturebenen einer Programmiersprache:
  - Syntax (Beschreibung mit EBNF)
  - Semantik
- Bool'sche Werte, Variablen, Operatoren, Ausdrücke
- Logische Operatoren
- Kontrollstrukturen in Java
  - `if`,`while`,`for`,`switch`

Lektion 2, Teil 2 (Repetitorium): Java-Programme, Bibliotheksmethoden, Scanner

Bibliotheksmethoden

- `System.out.println`: zum Erzeugen von Ausgaben am Bildschirm mit nachfolgendem Zeilenvorschub
- `System.out.print`: zum Erzeugen von Ausgaben am Bildschirm ohne Zeilenvorschub
- `Math.sqrt`: Berechnen der Wurzel. z.B. `Math.sqrt(4)` wertet zu 2.0 (Typ `double`) aus.
- `Math.sqrt` liefert im Gegensatz zu den oberen 2 Methoden einen Wert zurück
- ...

Der Funktionsumfang der Laufzeitbibliothek wird als "Java Application Programming Interface" bezeichnet (**Java-API**). Interface = Schnittstelle.

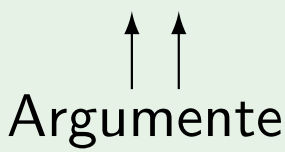
Mathematische Bibliotheksmethoden

- So wie die Operatoren eine bestimmte Anzahl an Operanden benötigen, erwarten die Methoden eine bestimmte Anzahl an **Argumenten**.
- mehrere Argumente werden durch Beistriche getrennt
- die mathematischen Methoden werden so wie Ausdrücke ausgewertet und liefern einen **Funktionswert** (meist `double`) zurück.

Beispiel (Berechnung einer Potenz)

Der `double`-Variablen `x` soll der Wert 3^7 zugewiesen werden:

```
x = Math.pow(3,7);
```



Mathematische Bibliotheksmethoden

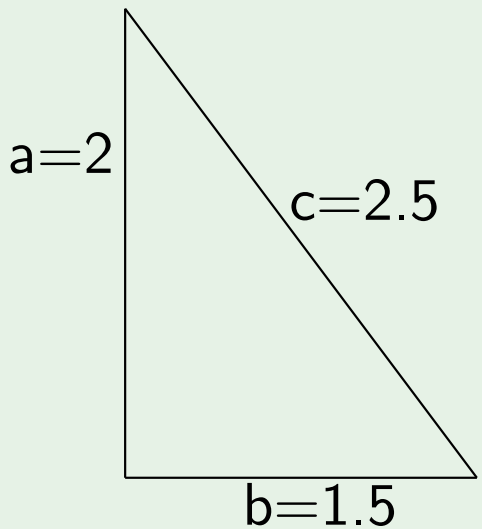
Beispiel (Verschachtelungen)

berechne $x = 3^{3^3}$: `x = Math.pow(3, Math.pow(3,3));`

berechne $x = (3^3)^3$: `x = Math.pow(Math.pow(3,3), 3);`

Bibliotheksmethode `Math.hypot()`

Beispiel (`Math.hypot()`)



Pythagoras:

$$\begin{aligned} c^2 &= a^2 + b^2 \\ c &= \sqrt{a^2 + b^2} \end{aligned}$$

Bibliotheksmethode `Math.hypot()`

Berechnung der Länge der Hypotenuse, wenn in `a,b` die Längen der Katheten gespeichert sind:

```
double a = 2;
double b = 1.5;
double c = Math.hypot(a,b);
```

das gleiche Ergebnis erhält man mit

```
double a = 2;
double b = 1.5;
double c = Math.sqrt(Math.pow(a,2)+Math.pow(b,2));
```

Mathematische Bibliotheksmethoden: Übersicht

Methodenname	Bedeutung	Argumente	Beispiel
`Math.sin`	Sinus	`double`	`Math.sin(3.1415)`
`Math.cos`	Cosinus	`double`	`Math.cos(3.1415)`
`Math.hypot`	Hypotenuse	`double , double`	`Math.hypot(2,1.5)`
`Math.exp`	e-Funktion	`double`	`Math.exp(1.0)`
`Math.pow`	Potenz	`double , double`	`Math.pow(3,7)`
`Math.max`	Maximum	`double , double`	`Math.max(3.0,5)`
`Math.min`	Minimum	`double , double`	`Math.min(3,5)`

Mathematische Bibliotheksmethoden: Übersicht

Mathematik	Java	Wert
$\sin(\pi)$	`Math.sin(3.1415)`	`9.265358966049026E-5`
$\cos(\pi)$	`Math.cos(3.1415)`	`-0.9999999957076562`
$\sqrt{2^2 + \left(\frac{3}{2}\right)^2}$	`Math.hypot(2,1.5)`	`2.5`
e^1	`Math.exp(1.0)`	`2.718281828459045`
3^7	`Math.pow(3,7)`	`6561.0`
$\max(3, 5)$	`Math.max(3.0,5)`	`5.0`
$\min(3, 5)$	`Math.min(3,5)`	`3`

Vordefinierte Variablen

Mathematik	Java	Beispiel	Wert
π	`Math.PI`	`Math.cos(Math.PI)`	-1.0
e	`Math.E`	`Math.log(Math.E)`	1.0

Bibliotheksmethoden: Werte am Bildschirm ausgeben

- `System.out.println` kann auch Zahlenwerte ausgeben:

```
double radius = 5.5;
double area = radius * radius * Math.PI;
System.out.println("Die Fläche des Kreises mit Radius ");
System.out.println(radius);
System.out.println(" ist ");
System.out.println(area);
```

erzeugt am Bildschirm

```
Die Fläche des Kreises mit Radius
5.5
 ist
95.03317777109125
```

Bibliotheksmethoden: Ausgabe mit oder ohne Zeilenvorschub

- `System.out.println`
  - mit abschliessendem Zeilenvorschub
- `System.out.print`
  - ohne abschliessenden Zeilenvorschub

Bibliotheksmethoden: Ausgabe mit oder ohne Zeilenvorschub

```
double radius = 5.5;
double area = radius * radius * Math.PI;
System.out.print("Die Fläche des Kreises mit Radius ");
System.out.print(radius);
System.out.print(" ist ");
System.out.println(area);
```

erzeugt am Bildschirm

```
Die Fläche des Kreises mit Radius 5.5 ist 95.03317777109125
```

Bibliotheksmethoden: Werte oder Text einlesen mit Scanner

```
//die folgende Zeile muss im Programmcode stehen
Scanner reader = new Scanner(System.in);

/* Echonumber: Eine Zahl wird eingelesen und am
   Bildschirm ausgegeben */
System.out.print("Bitte eine Zahl eingeben: ");
int input = reader.nextInt();

System.out.print("Sie haben die Zahl ");
System.out.print(input);
System.out.println(" eingegeben.");
```

Methoden: Werte oder Text einlesen mit `Scanner`

Der Wert 7 wird beim Ablauf des Programmes vom Benutzer eingegeben...

```
Bitte eine Zahl eingeben: 7
Sie haben die Zahl 7 eingegeben.
```

7 ist Eingabe des Benutzers

String: + Operator zum verketten

```
String output = "Sie haben die Zahl " + input + " eingegeben";
System.out.print(output);
```

erzeugt

```
Sie haben die Zahl 7 eingegeben.
```

Achtung:

```
System.out.print('a'+'b');
```

erzeugt

```
195
```

String: Zeilenumbrüche

```
String output = "eins\nzwei\ndrei";
System.out.print(output);
```

erzeugt

```
eins
zwei
drei
```

String: Beispiel für Verkettung mit Zeilenumbrüchen

```
//Draw a triangle
int zeilen = 4;
String output = "";
for (int i = 0; i<zeilen; i++) {
    for (int j = 0; j<=i; j++) {
        output += "*";
    }
    output += "\n";
}
System.out.print(output);
```

```
*
**
***
****
```

String: Länge einer Zeichenkette mit `length()`

- Die Länge einer Zeichenkette kann mit der Methode `length()` bestimmt werden.

```
String output = "eins\nzwei\ndrei";
int len = output.length();
System.out.print(len);
```

erzeugt

```
14
```

Vergleich von Zeichenketten (`String`)

- Zeichenketten sollten **nicht** mit dem Operator `==` auf Gleichheit überprüft werden.
- stattdessen sollte `equals` verwendet werden.
- Grund: String ist Referenztyp (siehe später)

```
String input;
input = reader.next();
if (input.equals("ja")) {
    ...
}
```

✓

Scanner-Methoden

Methode	zum Einlesen von
`nextInt()`	ganzen Zahlen
`nextDouble()`	Kommazahlen
`next()`	einer Zeichenkette
?	eines einzelnen Zeichens
⋮	

Scanner-Methoden: Beispiel Punkteschema

Gegeben sei folgendes Notenschema:

Punkte	[0 - 50[	[50 - 68[	[68 - 80[	[80 - 90[	[90 - 100]
Note	5	4	3	2	1

Scanner-Methoden: Einlesen von int-Werten

```
String grade;
Scanner reader = new Scanner(System.in);
System.out.print("Bitte Punktestand eingeben: ");
int points = reader.nextInt();
if (points >= 90)
    grade = "sehr gut";
else if (points >= 80)
    grade = "gut";
else if (points >= 68)
    grade = "befriedigend";
else if (points >= 50)
    grade = "genuegend";
else grade = "nicht genuegend";
System.out.println("Note: " + grade);
```

Scanner-Methoden: Einlesen von Strings

- `next()` liest ein Wort bis **exklusive** zum nächsten Leerzeichen oder Zeilenvorschub ein
- `nextLine()` liest bis zum Zeilenende (inkl. Zeilenvorschub) ein

H	a	l	l	o		W	e	l	t	\n

```
String a = reader.next();
String b = reader.nextLine();
```

Variable	bekommt "Wert"
a	"Hallo"
b	" Welt"

Scanner-Methoden: Einlesen von Strings

- `next()` liest ein Wort bis **exklusive** zum nächsten Leerzeichen oder Zeilenvorschub ein
- `nextLine()` liest bis zum Zeilenende (Zeilenvorschub) ein

H	a	l	l	o	\n

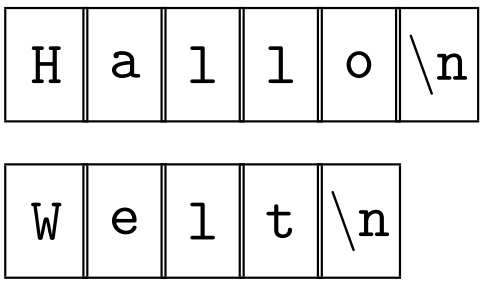
W	e	l	t	\n

```
String a = reader.nextLine();
String b = reader.next();
```

Variable	bekommt "Wert"
a	"Hallo"
b	"Welt"

Scanner-Methoden: Einlesen von Strings

- `next()` liest ein Wort bis **exklusive** zum nächsten Leerzeichen oder Zeilenvorschub ein
- `nextLine()` liest bis zum Zeilenende (inkl. Zeilenvorschub) ein




```
String a = reader.next();
String b = reader.nextLine();
```

Variable	bekommt "Wert"
`a`	`"Hallo"`
`b`	`""`

Scanner-Methoden: Einlesen von Strings

- `next()` liest ein Wort bis **exklusive** zum nächsten Leerzeichen oder Zeilenvorschub ein
- `nextLine()` liest bis zum Zeilenende (Zeilenvorschub) ein

H	a	l	l	o		W	e	l	t	\n

```
String a = reader.nextLine();
String b = reader.next();  //blockiert(wartet)
```

Variable	bekommt "Wert"
a	"Hallo Welt"
b	

Scanner-Methoden: Einlesen von einzelnen Zeichen

- Es gibt keine Methode zum Einlesen eines Zeichens.
- Folgender Befehl liefert aber als Abhilfe das erste Zeichen eines eingelesenen Wortes:

```
Scanner reader = new Scanner(System.in);
char answer;

answer = reader.next().charAt(0);
if (answer == 'j')
    System.out.println("JA");
else
    System.out.println("NEIN");
```

Allg. Form des Aufrufs von Scanner-Methoden

1. Ein Scanner-Objekt erzeugen (Erklärung folgt später)...

   `Scanner` *objektname* `= new Scanner(System.in);`

   Diese Anweisung sollte zusammen mit den Variablendefinitionen (am Anfang des Programms) stehen.
2. Wert einlesen (Scanner-Methoden verwenden)
   variablenname `=` *objektname*`.nextInt();`

GGT: Differenz Algorithmus

```
m = reader.nextInt();
```

```
n = reader.nextInt();
```

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

```
System.out.println(m);
```

"Programmrahmen"

```
class GGTDiff {
    public static void main (String[] args) {

      Hier kommt der obige Programmcode hinein...

    }
}
```

```java
import java.util.Scanner;

public class GGTDiff {
    public static void main (String[] args) {
        int n;
        int m;

        Scanner reader = new Scanner(System.in);

        n = reader.nextInt();
        m = reader.nextInt();

        while ( n != m ) {
            if ( m > n ) {
                m = m - n;
            } else {
                n = n - m;
            }
        }
        System.out.println(m);
    }
}
```

"Programmrahmen": allgemeine Form

```
public class   programmname {
    public static void main (String[] args) {

      Hier kommt der Programmcode hinein...

    }
}
```

Scanner: Überprüfung der Eingaben

- Folgende Methoden stehen zur Überprüfung der Eingaben zur Verfügung
- liefern Information über die Eingabe in Form von `boolean`-Werten
  - `hasNext()`: liefert `false` wenn das Ende der Eingabe ("End of File") erreicht wurde, sonst `true`
  - `hasNextInt()`: liefert `false` wenn das nächste Wort der Eingabe nicht als `int`-Wert gelesen werden kann, sonst `true`
  - `hasNextDouble()`: liefert `false` wenn das nächste Wort der Eingabe nicht als `double`-Wert gelesen werden kann, sonst `true`
- Weitere Info: http://java.sun.com/javase/6/docs/api/?java/util/Scanner.html

Gültigkeitsbereiche von Variablen

Eine Variable ist nur innerhalb des Blocks in dem sie definiert wurde gültig...

```
public class Test {
    public static void main(String[] args) {
        {
            int i = 2;
        }
        System.out.print(i); //Fehler i nicht definiert
    }
}
```

Gültigkeitsbereiche von Variablen

Eine Variable ist nur innerhalb des Blocks in dem sie definiert wurde gültig...

```
public class Test {
    public static void main(String[] args) {
        int i = 2;
        {
            System.out.print(i); //Ausgabe: 2
        }
    }
}
```

Lebensdauer von Variablen

- Beim Verlassen eines Programmblocks werden alle darin definierten Variablen verworfen
- In diesem Beispiel wird bei jedem Schleifendurchlauf eine Variable square erzeugt und wieder verworfen

```
// Bildet die Summe der Quadrate von 1 bis n
int n = 10;
int sum = 0;
int i = 0;
while (i <= n) {
    i++;
    int square = i*i;
    sum += square;
}
```

Namenskollision

- Namenskollision: Eine definierte Variable darf nicht erneut definiert werden (solange Sie noch gültig ist)

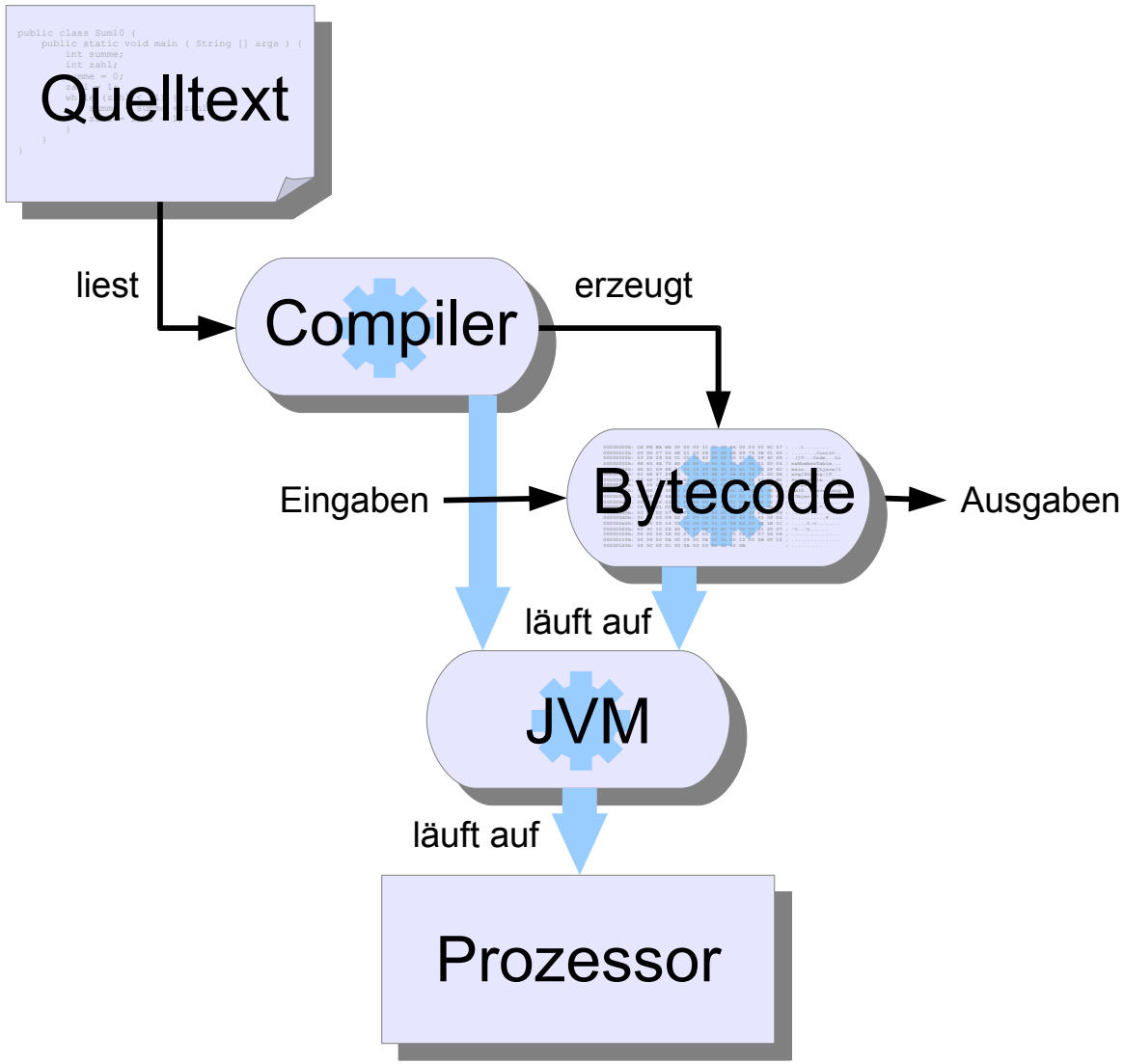
```
// Bildet die Summe der Quadrate von 1 bis n
int n = 10;
int sum = 0;
int i = 0;
while (i <= n) {
    i++;
    int n = i*i; // Fehler: n ist bereits definiert
    sum += n;
}
```

Quelltext

Definition

- **Quelltext, Code, Quellcode** (engl. source code): = vom Programmierer verfasster Text in einer formalen Sprache (Programmiersprache wie z.B. Java). Dient als "Quelle" zur Weiterverarbeitung.
- Zum Schreiben eines Quelltexts wird ein **Texteditor** benutzt der den Quelltext in einer **Quelldatei** speichert.

Übersetzer, Virtuelle Maschine



Java Quellcode-Datei: NumberOfDigits.java

```
public class NumberOfDigits {
  public static void main (String [] args) {
    //"Zutaten"
    int n;
    int d;

    //"Zubereitung"
    n = Integer.parseInt(args[0]);
    d = 1;

    while(n>9) {
      n = n / 10;
      d = d + 1;
    }
    System.out.println(d);
  }
}
```

Java Bytecode erstellen

1. Quelltext wird mit einem Editor bearbeitet
2. und in der Datei `NumberOfDigits.java` gespeichert
3. **Erzeugen von ausführbarem Bytecode:**

```
C:\>  javac  NumberOfDigits.java
```

Aufruf des Compilers

dem Compiler sagen, welche Datei

erzeugt die Datei `NumberOfDigits.class`

Java Bytecode-Datei: **NumberOfDigits.class**

mit einem Hex-Editor betrachtet...

```
00000000h: CA FE BA BE 00 00 00 31 00 21 0A 00 06 00 0F 0A ; ...1.!......
00000010h: 00 10 00 11 09 00 12 00 13 0A 00 14 00 15 07 00 ; ................
00000020h: 16 07 00 17 01 00 06 3C 69 6E 69 74 3E 01 00 03 ; .......<init>...
00000030h: 28 29 56 01 00 04 43 6F 64 65 01 00 0F 4C 69 6E ; ()V...Code...Lin
00000040h: 65 4E 75 6D 62 65 72 54 61 62 6C 65 01 00 04 6D ; eNumberTable...m
00000050h: 61 69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61 ; ain...([Ljava/la
00000060h: 6E 67 2F 53 74 72 69 6E 67 3B 29 56 01 00 0A 53 ; ng/String;)V...S
00000070h: 6F 75 72 63 65 46 69 6C 65 01 00 13 4E 75 6D 62 ; ourceFile...Numb
00000080h: 65 72 4F 66 44 69 67 69 74 73 2E 6A 61 76 61 0C ; erOfDigits.java.
00000090h: 00 07 00 08 07 00 18 0C 00 19 00 1A 07 00 1B 0C ; ................
000000a0h: 00 1C 00 1D 07 00 1E 0C 00 1F 00 20 01 00 0E 4E ; ........... ...N
000000b0h: 75 6D 62 65 72 4F 66 44 69 67 69 74 73 01 00 10 ; umberOfDigits...
000000c0h: 6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 ; java/lang/Object
000000d0h: 01 00 11 6A 61 76 61 2F 6C 61 6E 67 2F 49 6E 74 ; ...java/lang/Int
000000e0h: 65 67 65 72 01 00 08 70 61 72 73 65 49 6E 74 01 ; eger...parseInt.
000000f0h: 00 15 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 ; ..(Ljava/lang/St
00000100h: 72 69 6E 67 3B 29 49 01 00 10 6A 61 76 61 2F 6C ; ring;)I...java/l
00000110h: 61 6E 67 2F 53 79 73 74 65 6D 01 00 03 6F 75 74 ; ang/System...out
00000120h: 01 00 15 4C 6A 61 76 61 2F 69 6F 2F 50 72 69 6E ; ...Ljava/io/Prin
00000130h: 74 53 74 72 65 61 6D 3B 01 00 13 6A 61 76 61 2F ; tStream;...java/
00000140h: 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D 01 00 ; io/PrintStream..
00000150h: 07 70 72 69 6E 74 6C 6E 01 00 04 28 49 29 56 00 ; .println...(I)V.
00000160h: 21 00 05 00 06 00 00 00 00 00 02 00 01 00 07 00 ; !...............
00000170h: 08 00 01 00 09 00 00 00 1D 00 01 00 01 00 00 00 ; ................
00000180h: 05 2A B7 00 01 B1 00 00 00 01 00 0A 00 00 00 06 ; .*............
00000190h: 00 01 00 00 00 01 00 09 00 0B 00 0C 00 01 00 09 ; ................
000001a0h: 00 00 00 57 00 02 00 03 00 00 00 23 2A 03 32 B8 ; ...W.......#*.2
000001b0h: 00 02 3C 04 3D 1B 10 09 A4 00 0F 1B 10 0A 6C 3C ; ..<.=........l<
000001c0h: 1C 04 60 3D A7 FF F1 B2 00 03 1C B6 00 04 B1 00 ; ..`=.......
000001d0h: 00 00 01 00 0A 00 00 00 22 00 08 00 00 00 08 00 ; ........".......
000001e0h: 07 00 09 00 09 00 0B 00 0F 00 0C 00 14 00 0D 00 ; ................
000001f0h: 18 00 0E 00 1B 00 0F 00 22 00 10 00 01 00 0D 00 ; ........".......

00000200h: 00 00 02 00 0E ; .....
```

Java Bytecode: Auschnitt aus **NumberOfDigits.class**

Befehl Nr	Befehl	Code des Befehls
6:	`istore_1`	3C
7:	`iconst_1`	04
8:	`istore_2`	3D
9:	`iload_1`	1B
10:	`bipush 9`	10 09
12:	`if_icmple 27`	A4 00 0F
15:	`iload_1`	1B
16:	`bipush 10`	10 0A
18:	`idiv`	6C
19:	`istore_1`	3C
20:	`iload_2`	1C
21:	`iconst_1`	04
22:	`iadd`	60
23:	`istore_2`	3D
24:	`goto 9`	A7

Übung: Beispiele aus der Vorlesung

- Die folgenden Beispiele (GGTDiff, SumN) wurden in der Vorlesung besprochen
- Es gibt je 2 Varianten
- Als Übung für Zuhause: Vergleichen Sie die Varianten
- Kopieren Sie den Quellcode in einen Editor, speichern Sie unter dem entsprechenden Namen, übersetzen Sie diese und führen Sie den Bytecode aus.

Berücksichtigung von falschen Eingaben

```
import java.util.Scanner;

public class GGTDiff {
        public static void main (String[] args) {
                int n = 0;
                int m = 0;
                boolean ok = true;
                Scanner reader = new Scanner(System.in);

                if (reader.hasNextInt()) {
                        n = reader.nextInt();
                        if (reader.hasNextInt()) {
                            m = reader.nextInt();
                        } else {
                            ok = false;
                        }
                } else {
                        ok = false;
                }

                if (ok) {
                        if (n == 0 || m == 0) {
                                System.out.println(m+n);
                        } else {
                                while ( n != m ) {
                                        if ( m > n ) {
                                                m = m - n;
                                        } else {
                                                n = n - m;
                                        }
                                }
                                System.out.println(m);
                        }
                } else {
                        System.out.println("FALSCHE EINGABE");
                }
        }
}
```

Berücksichtigung von falschen Eingaben

```
import java.util.Scanner;

public class GGTDiff {
        public static void main (String[] args) {
                int n = 0;
                int m = 0;
                boolean ok = false;
                Scanner reader = new Scanner(System.in);

                if (reader.hasNextInt()) {
                        n = reader.nextInt();
                        if (reader.hasNextInt()) {
                                m = reader.nextInt();
                                ok = true;
                        }
                }

                if (ok) {
                        if (n == 0 || m == 0) {
                                System.out.println(m+n);
                        } else {
                                while ( n != m ) {
                                        if ( m > n ) {
                                                m = m - n;
                                        } else {
                                                n = n - m;
                                        }
                                }
                                System.out.println(m);
                        }
                } else {
                        System.out.println("FALSCHE EINGABE");
                }
        }
}
```

Berücksichtigung von falschen Eingaben

```
import java.util.Scanner;

/*
 * Liest beliebig viele Ganzzahlen ein und gibt nach Ende
 * der Eingabe deren Summe aus. Falls eine nicht als
 * Ganzzahlen interpretierbare Zeichenkette eingegeben
 * werden wird, wird mit einer Fehlermeldung abgebrochen.
 */
public class SumN {
    public static void main (String[] args) {
        Scanner reader = new Scanner(System.in);
        boolean ok = true;
        int sum = 0;

        while (reader.hasNext() && ok) {
            if (reader.hasNextInt()) {
                sum += reader.nextInt();
            } else ok = false;
        }

        if (ok) {
            System.out.println(sum);
        } else {
            System.out.println("FALSCHE EINGABE");
        }
    }
}
```

Berücksichtigung von falschen Eingaben

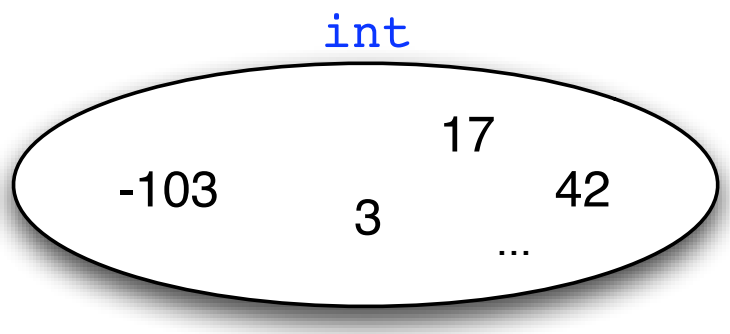
```
import java.util.Scanner;

/*
 * Liest beliebig viele Ganzzahlen ein und gibt nach Ende
 * der Eingabe deren Summe aus. Falls eine nicht als
 * Ganzzahlen interpretierbare Zeichenkette eingegeben
 * werden wird, wird mit einer Fehlermeldung abgebrochen.
 */
public class SumN {
    public static void main (String[] args) {
        Scanner reader = new Scanner(System.in);
        boolean ok = true;
        int sum = 0;

        while (reader.hasNext()) {
            if (reader.hasNextInt()) {
                sum += reader.nextInt();
            } else {
                ok = false;
                break; //Schleife wird abgebrochen
            }
        }

        if (ok) {
            System.out.println(sum);
        } else {
            System.out.println("FALSCHE EINGABE");
        }
    }
}
```

Lektion 2, Teil 3: Klassen (1)

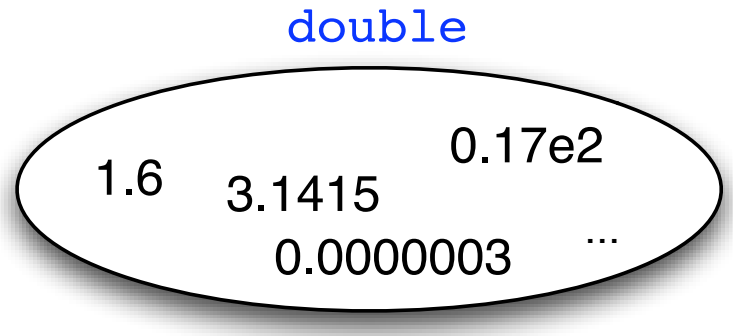
Klassen - eigene Datentypen definieren

Wiederholung: Ein Datentyp bestimmt

- eine Menge von **Werten** und
- darauf anwendbare **Operationen**



`+ - * / %` ...



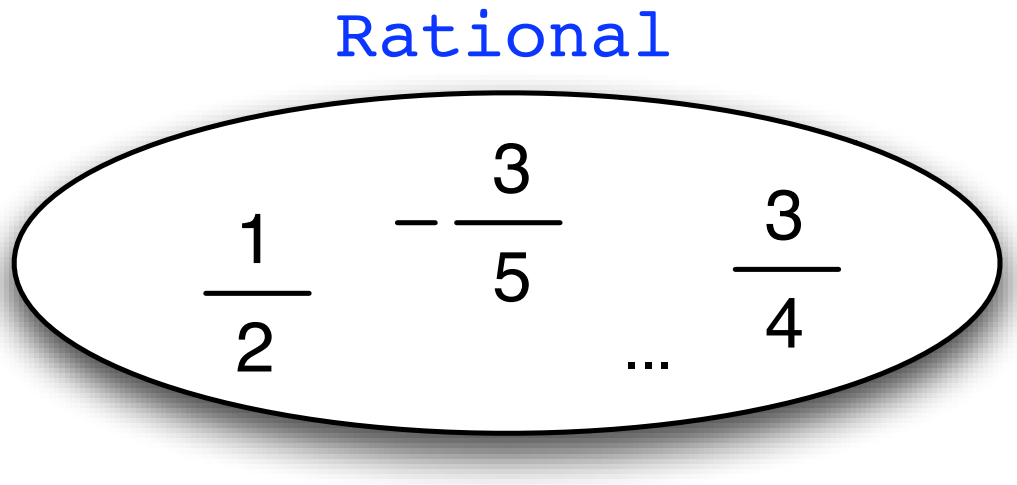
`+ - * / %` ...

Klassen - eigene Datentypen definieren

- Wir haben bisher folgende Datentypen verwendet:
  - `int`
  - `double`
  - `char`
  - `String`

Klassen - eigene Datentypen definieren

- Wir wollen lernen, eigene komplexere "Datentypen" selbst zu erstellen



Klassen - Beispiel `Rational`

Datentyp für Brüche (Bezeichnung `Rational` ist selbst gewählt) ist zusammengesetzt aus 2 Zahlenwerten:

- ganzzahliger Zähler (*Numerator*)
- ganzzahliger Nenner (*Denominator*)

Beispiele:

Mögliche "Werte" für `Rational`:

$$\frac{3}{4} \qquad \frac{1}{9}$$

$\leftarrow$ *Numerator*

$\leftarrow$ *Denominator*

Klassen - Beispiel `Rational`

Die (minimale) Definition unseres neuen Datentyps erfolgt zunächst durch

```
class Rational {
    int num;        //Zaehler
    int denom;      //Nenner
}
```

Das bedeutet:

- Der neue Typ `Rational` besteht aus 2 ganzzahligen Komponenten: `num` und `denom`
- `num` und `denom` sind Namen von **Datenelementen** (manchmal auch **Instanzvariablen** oder **Felder** genannt)

Klassen - Beispiel `Rational`

Klassendefinition

```
class Rational {
    int num;        ⇐ Variablendefinition
    int denom;      ⇐ Variablendefinition
}
```

- Eine Klassendefinition ist wie ein Bauplan
- Die Datenelemente werden mit Variablendefinitionen festgelegt

Klassendefinition allgemein

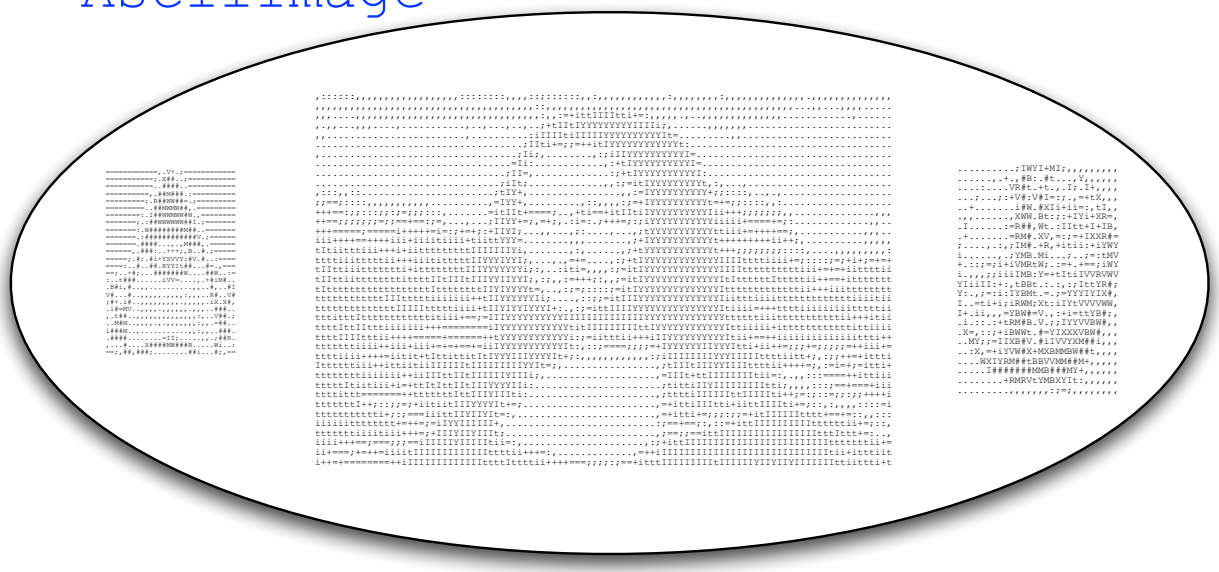
Klassendefinition
```
class Identifier {
    Variabledefinitions(Datenelemente)
    ...
}
```

- Eine Klassendefinition ist wie ein Bauplan
- Die Datenelemente werden mit Variablendefinitionen festgelegt

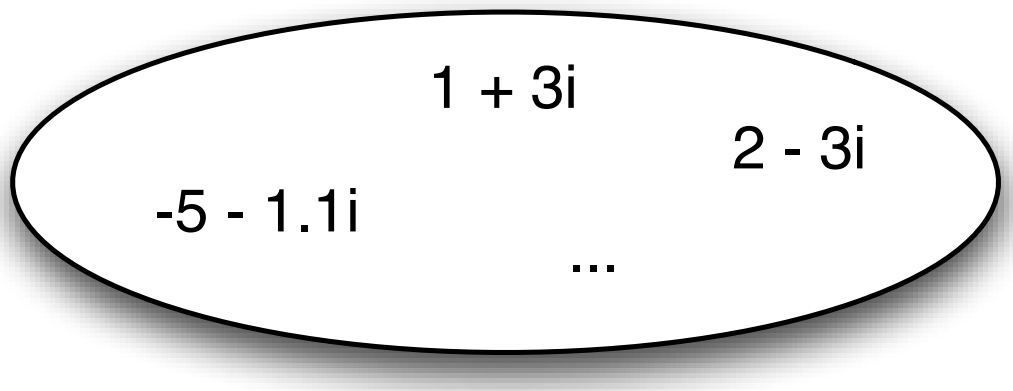
Klassen - eigene Datentypen definieren

- Weitere Beispiele von Typen mit 2 oder mehreren Komponenten (aus den Übungsrunden):

`AsciiImage`



`Complex`



Klassen - Beispiel `Rational`

- Wie können Klassen benutzt werden?
- Wie spezielle, selbst definierte Typen...
- Aber wie wird ein neuer "Wert" erzeugt und zugewiesen?

Beispiel (Verwendung von Klassen)

primitiver Typ

```
double temperature;
temperature = 12.5;
```

neuer Typ (Klasse)

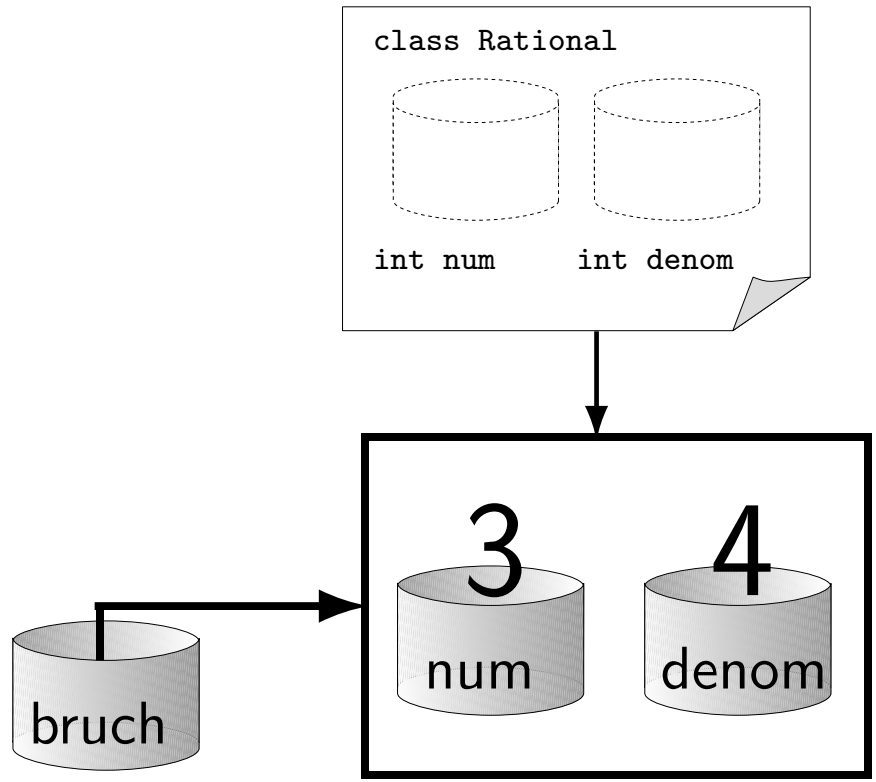
```
Rational r;
r = ...?
```

Klassen - Beispiel `Rational`

- Klassen sind wie Baupläne
- Nach diesem Bauplan können zur Ausführungszeit des Programms **Objekte** dieser Klasse erzeugt werden
- Objekte heißen manchmal auch **Exemplare** oder **Instanzen** der Klasse
- Objekte belegen Speicherplatz zur Laufzeit des Programms

Klassen - Beispiel `Rational`

- Klassen definieren **Referenztypen**
- Variablen eines solchen Typs und der Speicherplatz des referenzierten Objekts existieren getrennt voneinander




```
Rational bruch;
bruch = new Rational();
bruch.num = 3;
bruch.denom = 4;
```

Unterschied "primitive" Variablen - Referenzvariablen

Variablen eines primitiven Typs:

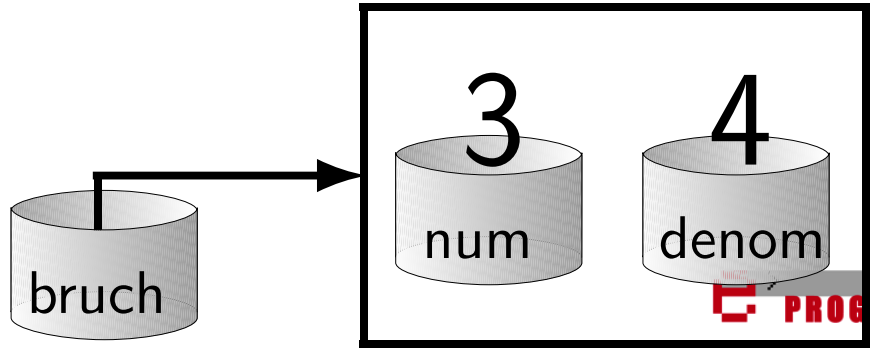
- `int`
- `double`
- `char`

```
int answer = 42;
```

Referenzvariablen sind Variablen der "Objekt-Typen" (Klassen):

- `String`
- `Rational` (siehe oben)
- ...

```
Rational bruch;
bruch = new Rational();
bruch.num = 3;
bruch.denom = 4;
```



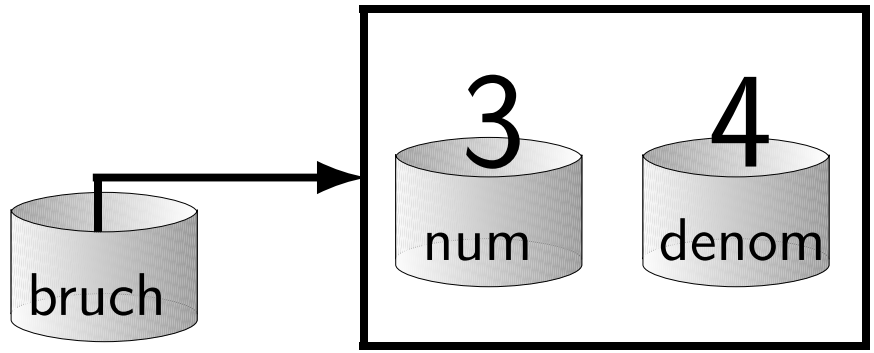
Unterschied "primitive" Variablen - Referenzvariablen

- Variablen eines primitiven Typs speichern Werte

```
int answer;
```

- Referenzvariablen verweisen auf Objekte

```
Rational bruch;
```



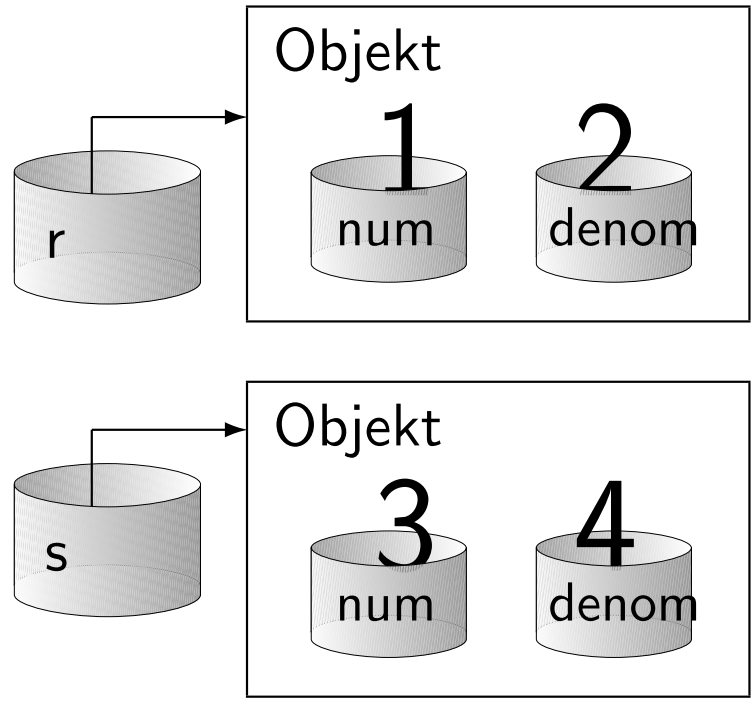
Datenelemente - Zugriff

- Die Datenelemente eines Objektes können einzeln angesprochen werden
- Syntax: *ObjectvariableIdentifier*.*ElementIdentifier*
- Werden genauso gehandhabt wie lokale Variablen des gleichen Typs

```
Rational r = new Rational();
r.num = 1;
r.denom = 3;

r.num++;
r.denom *= 2;
double rationalValue = (double) r.num / r.denom;

System.out.println("Der Wert des Bruches "
                   + r.num + "/" + r.denom
                   + " ist " + rationalValue);
```

Referenztypen

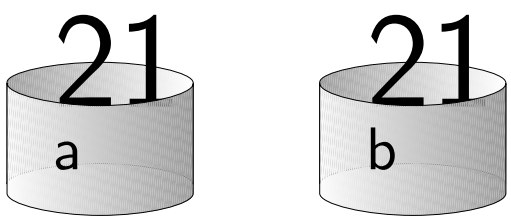
```
Rational r = new Rational();
r.num = 1;
r.denom = 2;

Rational s = new Rational();
s.num = 3;
s.denom = 4;
```



Unterschied "primitive" Variablen - Referenzvariablen

primitive Variable: Hier wird der Wert von `a` in `b` gespeichert (kopiert)...

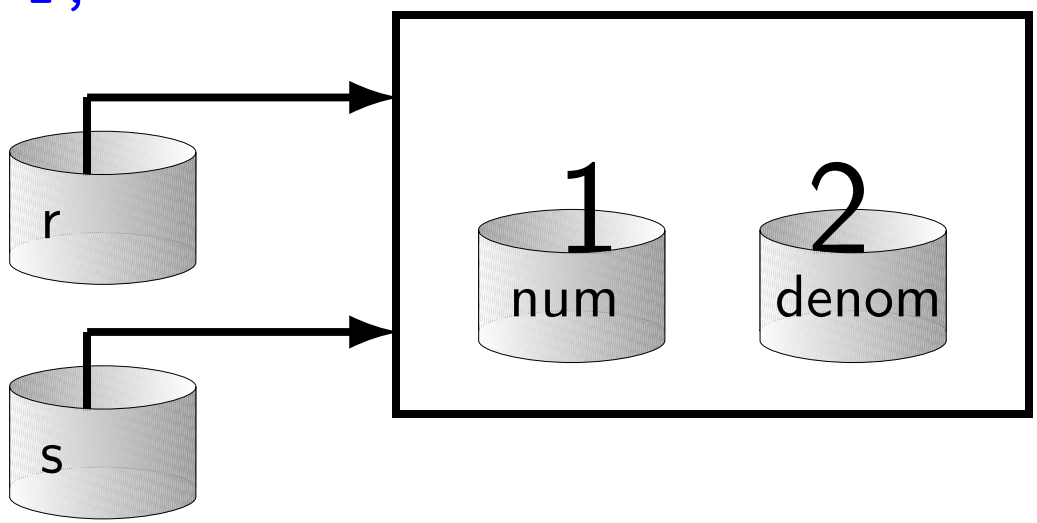
```
int a = 21;
int b = a;
```



Unterschied "primitive" Variablen - Referenzvariablen

Referenzvariable: Beide Variablen verweisen hier auf *dasselbe* Objekt... (es wird hier keine Kopie erzeugt)

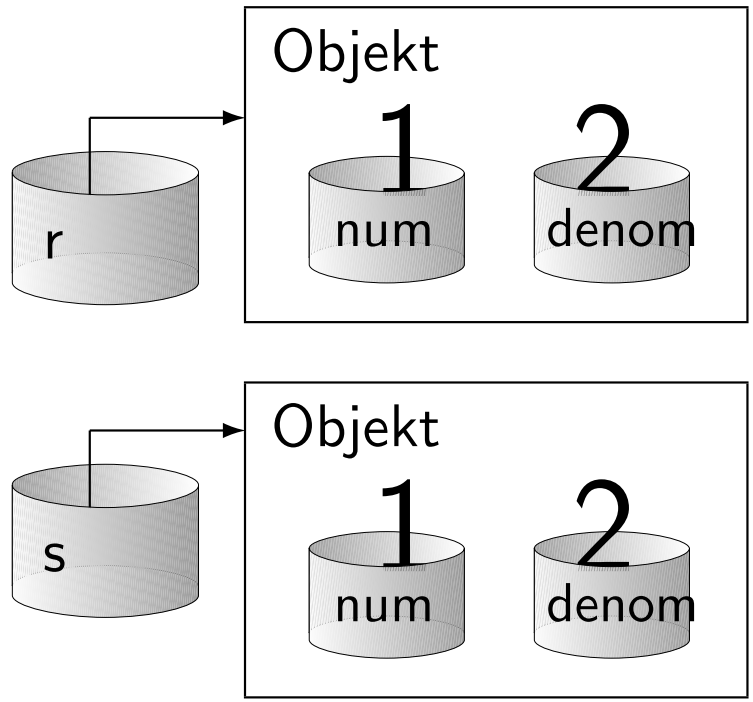
```
Rational r = new Rational();
r.num = 1;
r.denom = 2;

Rational s = r;
```



Referenztypen

Hier wird eine echte Kopie eines Objektes erzeugt. Das Resultat sind Objekte, die gleich sind aber nicht identisch...

```
Rational r = new Rational();
r.num = 1;
r.denom = 2;

Rational s = new Rational();
s.num = r.num;
s.denom = r.denom;
```



Objekt-Identitätsprinzip

- Jedes Objekt ist per Definition unabhängig von seinen konkreten Attributwerten (=Werte der Datenelemente) von allen anderen Objekten eindeutig zu unterscheiden

Unterschied "primitive" Variablen - Referenzvariablen

```
int a = 1;
int b = a;

b = 2;
System.out.println("Wert von a: " + a);
```

Ausgabe: `Wert von a:  1`

Unterschied "primitive" Variablen - Referenzvariablen

```
Rational r = new Rational();
r.num = 1;
r.denom = 3;

Rational s = r;

s.num = 2;
System.out.println("Bruch r: " + r.num + "/" + r.denom);
```

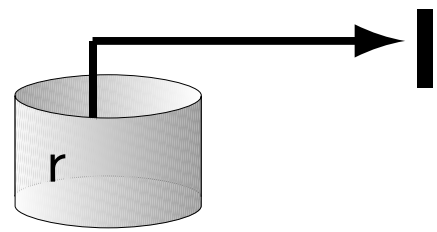
Welche Ausgabe ist hier zu erwarten?

1. `Bruch r: 1/3`
2. `Bruch r: 2/3` ✓ **"Aliasing"**

null-Referenz

- Wenn eine Referenzvariable kein Objekt referenzieren soll kann man ihr `null` zuweisen (**null**-Literal)
- `null` ist nicht zu verwechseln mit einer fehlenden Initialisierung!

```
Rational r;

//Fehler: Zugriff ohne Initialisierung
if (r == null) { ... }

r = null;

if (r == null)
    System.out.println("kein Objekt");
```



Datenelemente: Gültigkeitsbereich und Lebensdauer

- Ein Datenelement ist innerhalb der Klasse gültig, in der es definiert wurde
- Die Reihenfolge der Definition in der Klasse spielt keine Rolle
- Die Lebensdauer entspricht der Lebensdauer des Objektes
  - Beginn: Erzeugung mit `new`
  - Ende: Wenn keine Referenzvariable mehr auf das Objekt verweist (z.B. wenn man im obigen Beispiel `r` und `s` ein neues Objekt oder `null` zuweist)

Objekte

- In der realen Welt vorkommenden Gegenstände können durch Objekte modelliert werden
- Beispiele für Objekte: Lampe, Telefon, Fahrrad, Mensch, Versicherungspolizze
- Objekte können charakterisiert werden durch:
  - Zustand
  - Verhalten
  - Identität
- Kapsel mit eigenem Zustand+Verhalten+Identität

Klassen

Eine Klasse

- beschreibt die Struktur und das Verhalten einer Menge gleichartiger Objekte
- fasst Attribute (Datenelemente) und Operationen (Methoden) zu einer Einheit zusammen.
- Attribute (Datenelemente): beschreiben Zustand
- Operationen (Methoden): beschreiben Verhalten

Objekt-Klassen-Prinzip

- Ein Objekt ist eine zur Ausführungszeit vorhandene Instanz (ein Exemplar), die
  - für ihre Datenelemente Speicherplatz allokiert
  - sich entsprechend dem Protokoll ihrer Klasse verhält

Klassen

```
class Bicycle {
        // Datenelemente
        int speed = 0;
        int gear = 1;

        // Methoden
        void speedUp() {
             speed = speed + 5;
        }

        void applyBrakes() {
             speed = speed - 5;
        }

        void changeGear(int newValue) {
             gear = newValue;
        }

        void printStates() {
             System.out.println("Geschwindigkeit:" + speed + " Gang:" + gear);
        }
}
```

Hauptprogramm (benutzt Klasse `Bicycle`)

```
class BicycleApplication {
     public static void main(String[] args) {

          // Erzeuge zwei Bicycle Objekte
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Methoden der Objekte werden aufgerufen
          bike1.speedUp();
          bike1.speedUp();
          bike1.changeGear(2);
          bike1.printStates();

          bike2.speedUp();
          bike2.changeGear(2);
          bike2.speedUp();
          bike2.changeGear(3);
          bike2.printStates();
     }
}
```

Methoden

Eine Klassendefinition besteht aus der Definition von

- Datenelementen: Welche Daten sollen Objekte beschreiben - speichern **Zustand der Objekte**
- Methoden: Welche Aktionen können Objekte auf den Daten durchführen - steuern **Verhalten der Objekte**

```
class Rational {

    //Datenelemente
    int num;
    int denom;

    //Methode: Ausgabe am Bildschirm
    void output() {
        System.out.println(num + "/" + denom);
    }
}
```

Aufruf einer Methode

- Der Aufruf einer Methode ähnelt dem Zugriff auf die Datenelemente
- Syntax: *ObjectvariableIdentifier*`.`*MethodIdentifier*`()`

```
Rational r = new Rational();
r.num = 4;
r.denom = 6;
r.output();
```

erzeugt am Bildschirm

```
4/6
```

Methoden

Hauptprogramm

```
class Main {
    public static void main (String[] args) {
        //Zutaten
        Rational r = new Rational();

        //Zubereitung
        r.num = 4;
        r.denom = 6;
        r.output();

        r.num = 3;
        r.output();
    }
 }
```

Klassendefinition

```
class Rational {

    //Datenelemente
    int num;
    int denom;

    //Methode: Ausgabe am Bildschirm
    void output() {
        System.out.println(num + "/" + denom);
    }
}
```

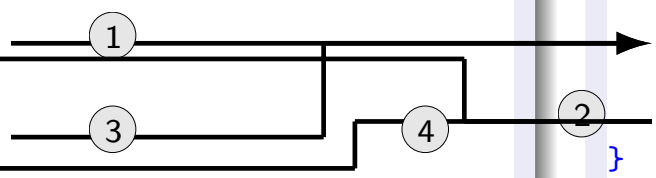
Methoden: Call Sequence

1. Das aufrufende Programm `Main.java` wird unterbrochen
2. Der Methodenrumpf wird durchlaufen
3. Das aufrufende Programm wird fortgesetzt

Hauptprogramm

`Main.java`

```
class Main {
    public static void main (String[] args) {
        //Zutaten
        Rational r = new Rational();

        //Zubereitung
        r.num = 4;
        r.denom = 6;
        r.output();

        r.num = 3;
        r.output();
    }
}
```



Klassendefinition

`Rational.java`

```
class Rational {

    //Datenelemente
    int num;
    int denom;

    //Methode: Ausgabe am Bildschirm
    void output() {
        System.out.println(num + "/" + denom);
    }
}
```

Aufbau einer Methode

- Der Aufbau eines Methodenrumpfs ist so, wie wir es von `main` gewohnt sind und enthält
  - Definitionen von **lokalen** Variablen
  - Zuweisungen
  - Kontrollanweisungen
- Die lokalen Variablen werden bei jedem Aufruf der Methode neu erzeugt und beim Verlassen wieder verworfen

Aufbau einer Methode

- Jedes Java-Programm ist eine **Klassendefinition**
- Jede Methode ist **Komponente einer Klasse**
- in den bisher verwendeten "Programmrahmen" haben wir immer eine Methode `main` benutzt
- Diese war Komponente einer **ausführbaren Klasse** ("entry-point")
- Klassen ohne Methode `main` können nicht direkt ausgeführt werden, sondern definieren Referenztypen die von anderen Klassen benutzt werden können (mit `new`)