

252-0027

Einführung in die Programmierung

2.0 Einfache Java Programme

Thomas R. Gross

**Department Informatik
ETH Zürich**

252-0027

Einführung in die Programmierung

2.5 Schleifen («Loops»)

Thomas R. Gross

**Department Informatik
ETH Zürich**

Übersicht

- **2.5 Schleifen (Loops)**
 - 2.5.1 «for» Loops
 - 2.5.2 Verschachtelte Schleifen
 - 2.5.3 «while» Loops

252-0027

Einführung in die Programmierung

2. Z Zufallszahlen

Thomas R. Gross

**Department Informatik
ETH Zürich**

Zufallszahlen aus der Bibliothek

- **Statt Zahlen einlesen ... mit «zufälligen» Werten arbeiten**
 - Auch dafür hat Java einen Service
- **Random ist in der Bibliothek `java.util` definiert**
 - Muss erst bekannt gegeben werden
`import java.util.Random; // so dass wir Random benutzen können`
- **Programm braucht ein Random Objekt :**
 - Dieses muss konstruiert werden
`Random zufall = new Random();`

Random Methoden («Services»)

- Random liefert einen Zufallszahlengenerator
 - Pseudozufallszahlen (z.B. zwischen 0 ... 9, einschliesslich)

Method name	Description
nextInt()	returns a random integer
nextInt(max)	returns a random integer in the range $[0, \text{max})$ in other words, 0 to $\text{max} - 1$ inclusive
nextDouble()	returns a random real number in the range $[0.0, 1.0)$

- **Beispiel:**

```
import java.util.Random;

Random rand = new Random();
int randomNumber = rand.nextInt(10); // 0-9
```

Erzeugen von Zufallszahlen

- Häufig brauchen wir (ganze) Zufallszahlen zwischen 1 und N

```
int n = rand.nextInt(20) + 1; // 1-20 inclusive
```

- Um eine ganze Zahl in irgendeinem Interval [*min*, *max*] zu bekommen (inklusive Grenzen):

```
name.nextInt(size of range) + min; // name Zufallszahlengenerator  
mit (size of range) == (max - min + 1)
```

- Beispiel: Eine zufällige ganze Zahl zwischen 4 und 10 einschliesslich:

```
int n = rand.nextInt(7) + 4;
```

Fragen zu Random

Mit dieser Deklaration

```
Random rand = new Random();
```

wie würden Sie erhalten?

1. Eine zufällige ganze Zahl zwischen 1 und 47 einschliesslich?

```
int random1 = rand.nextInt(47) + 1;
```
2. Eine zufällige ganze Zahl zwischen 23 und 30 einschliesslich?

```
int random2 = rand.nextInt(8) + 23;
```
3. Eine zufällige ganze *gerade* Zahl zwischen 4 and 12 einschliesslich?

```
int random3 = rand.nextInt(5) * 2 + 4;
```


Random und andere Typen

- Jede Menge von Werten der Basistypen kann auf die ganzen Zahlen abgebildet werden – hilft auch bei anderen Typen
 - Code um zufällig Schere-Stein-Papier zu spielen:

```
int r = rand.nextInt(3);
if (r == 0) {
    System.out.println("Schere");
} else if (r == 1) {
    System.out.println("Stein");
} else { // r == 2
    System.out.println("Papier");
}
```

2.5.1 Einfache Schleifen: «for»-loop

- Schleifen erlauben wiederholte Ausführung einer (oder mehrerer) Anweisung(en).
- Schleifen («loops») kommen in verschiedenen Varianten
- Zuerst: «for»-loop
 - Fixe Anzahl an Wiederholungen (wenn richtig eingesetzt ...)

Eine einfache Additionsaufgabe

■ Man nehme:

- Zwei ganze Zahlen Z_1 und Z_2 zwischen 1 .. 10 (einschliesslich)
- **Präsentiere** die Aufgabe $Z_1 + Z_2$ und **lese die Eingabe**
- Vergleiche Eingabe mit Summe, zähle Anzahl Fehler

■ ... besser: drei Aufgaben

■ Programm(segment)

```
Scanner console = new Scanner(System.in);
Random rand = new Random() // play games
int wrong = 0;
int operand1;
int operand2;

operand1 = rand.nextInt(10) + 1;
operand2 = rand.nextInt(10) + 1;
System.out.print(operand1 + " + " +
+ operand2 + " = ");
if (console.nextInt() !=
(operand1+operand2)) {
wrong = wrong + 1;
}
```

Statt Wiederholungen der Anweisungen ...

```
Scanner console = new Scanner(System.in);
Random rand = new Random() // play games
int wrong = 0;
int operand1;
int operand2;
```

**Drei einfache Additionsaufgaben
mit zufälligen Zahlen 1 ... 10**

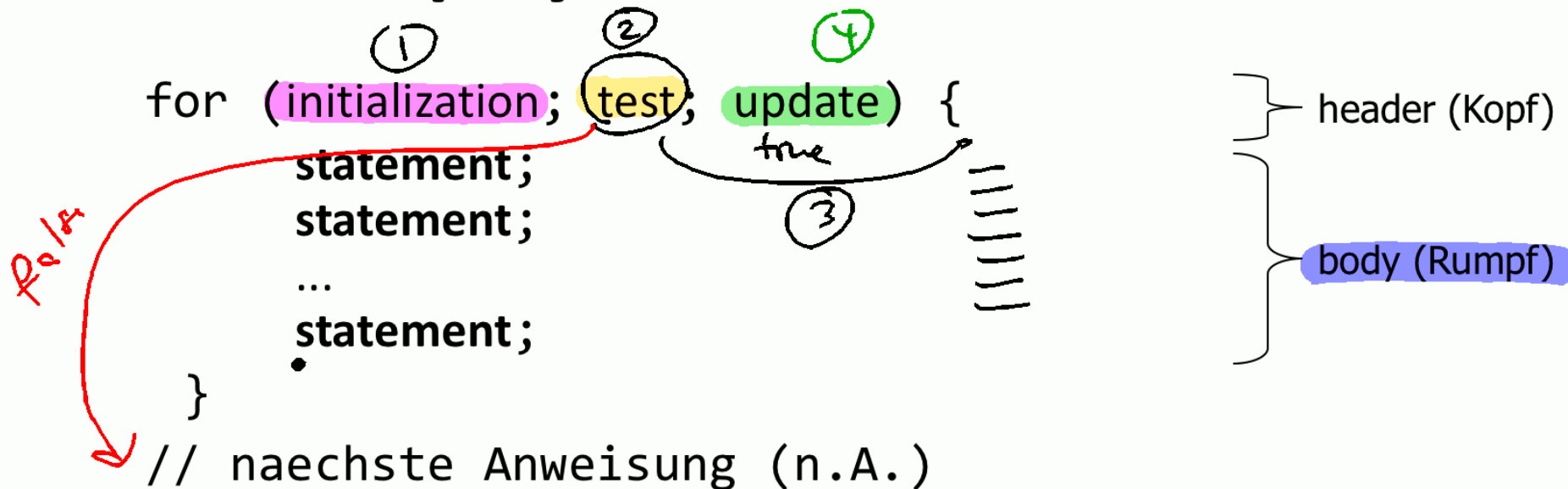
```
operand1 = rand.nextInt(10) + 1;
operand2 = rand.nextInt(10) + 1;
System.out.print(operand1 + " + " + operand2 + " = ");
if (console.nextInt() != (operand1+operand2)) { wrong = wrong + 1;}
operand1 = rand.nextInt(10) + 1;
operand2 = rand.nextInt(10) + 1;
System.out.print(operand1 + " + " + operand2 + " = ");
if (console.nextInt() != (operand1+operand2)) { wrong = wrong + 1;}
operand1 = rand.nextInt(10) + 1;
operand2 = rand.nextInt(10) + 1;
System.out.print(operand1 + " + " + operand2 + " = ");
System.out.println("You made " + wrong + " mistake(s).");
```

«for»-loops erlauben Wiederholungen

- Javas «for»-loop Anweisung wiederholt Anweisungen

```
for (int i = 1; i <= 3; i = i + 1) {  
  
    // repeat 3 times  
    operand1 = rand.nextInt(10) + 1;  
    operand2 = rand.nextInt(10) + 1;  
    System.out.print(operand1 + " + " + operand2 + " = ");  
    if (console.nextInt() != (operand1+operand2)) {  
        wrong = wrong + 1;  
    }  
  
}
```

«for»-loop Syntax



- ①
 - **Initialisierung («Initialization»)** wird einmal ausgeführt.
 - Wiederhole diese Schritte:
 - Prüfe ob test wahr (true) ergibt. Wenn nicht, stop und weiter mit n.A.
 - Führe die Anweisung(en) (**Statement(s)**) aus.
 - Führe die Aktualisierung (**Update**) aus.

Initialisierung

```
for (int i = 1; i <= 5; i = i + 1) {  
    System.out.println("Ich werde die Uebungsaufgaben machen");  
}
```

- **Legt die Variable fest, die im Loop verwendet wird**
- **Wird *einmal* am Anfang der Schleife ausgeführt**
 - Diese Variable heisst Schleifenzähler («loop counter»)
 - Kann jeden Namen haben, nicht nur `i`
 - Kann mit jedem Wert anfangen, nicht nur 1

Test

```
for (int i = 1; i <= 5; i = i + 1) {  
    System.out.println("Ich werde die Uebungsaufgaben machen");  
}
```

- **Vergleicht die Zählervariable mit einem Grenzwert**
- **Verwendet Vergleichsoperatoren («comparison operators»)**
 - Die selben wie für if-Statements
 - < weniger als («less than»)
 - <= weniger als oder gleich («less than or equal to»)
 - > grösser als («greater than»)
 - >= grösser als oder gleich («greater than or equal to»)

Aktualisierung

```
for (int i = 1; i <= 5; i = i + 1) {  
    System.out.println("Ich werde die Uebungsaufgaben machen");  
}
```

- **Die Zählervariable muss sich ändern**
 - Sonst findet die Schleife kein Ende
 - test ergibt immer true
- **Ein beliebiger Ausdruck zulässig**
 - Compiler prüft nicht dass die Zählervariable verwendet wird

Wiederholungen für ein Interval [1..6]

```
System.out.println("1 hoch 2 = " + 1 * 1);  
System.out.println("2 hoch 2 = " + 2 * 2);  
System.out.println("3 hoch 2 = " + 3 * 3);  
System.out.println("4 hoch 2 = " + 4 * 4);  
System.out.println("5 hoch 2 = " + 5 * 5);  
System.out.println("6 hoch 2 = " + 6 * 6);
```

- Intuition: «Ich will eine Zeile für jede Zahl von 1 bis 6 ausgeben»
- **Ein «for»-Loop erledigt genau diesen Job!**

```
for (int i = 1; i <= 6; i = i+1) {  
    System.out.println(i + " hoch 2 = " + (i * i));  
}
```

- "Für jede ganze Zahl *i* von 1 bis 6, drucke..."

Mehrere Anweisungen im Rumpf

```
System.out.println("+-----+");
for (int i = 1; i <= 3; i = i+1) {
    System.out.println("\    /");
    System.out.println("/    \");
}
System.out.println("+-----+");
```

■ Output:

```
+-----+
\      /
/      \
\      /
/      \
\      /
/      \
+-----+
```

Schleifenkontrolle

```
int highTemp = 5;  
for (int i = -3; i <= highTemp / 2; i = i + 1) {  
    System.out.println(i + " C = " + (i * 1.8 + 32) + " F");  
}
```

Output:

```
-3 C = 26.6 F  
-2 C = 28.4 F  
-1 C = 30.2 F  
0 C = 32.0 F  
1 C = 33.8 F  
2 C = 35.6 F
```

Hochzählen, herunterzählen

- Die Aktualisierung («update») kann auch den Schleifenzähler herunterzählen

Aber der Vergleich in *test* muss dann > anstatt von < verwenden

```
System.out.print("T-minus ");
for (int i = 10; i >= 1; i = i-1) {
    System.out.print(i + " ");
}
System.out.println("blastoff!");
System.out.println("The end.");
```

Output: T-minus 10 9 8 7 6 5 4 3 2 1 blastoff!
The end.

2.5.2 Verschachtelte for-Schleifen

«for»-Schleife

```
for (initialization; test; update) {  
    statement;  
    statement;  
    ...  
    statement;  
}
```

} header (Kopf)

} body (Rumpf)

statement (Anweisung im Rumpf der Schleife) kann beliebige Java Anweisung sein

Auch wieder eine Schleife

Verschachtelte Schleifen

- **Verschachtelte Schleifen («nested loop»): Schleife in einer Schleife**

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; j <= 10; j = j+1) {  
        System.out.print("*");  
    }  
    System.out.println();    // to end the line  
}
```


Verschachtelte Schleifen

- **Verschachtelte Schleifen («nested loop»): Schleife in einer Schleife**

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; j <= 10; j = j+1) {  
        System.out.print("*");  
    }  
    System.out.println();    // to end the line  
}
```

Output:

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

- **Der Rumpf der *äusseren Schleife* wird 5-mal ausgeführt, der Rumpf der *inneren (Schleife)* 10-mal (jedesmal)**

Verschachtelte Schleifen

Was gibt dieses Programmsegment aus?

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; j <= i; j = j+1) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

■ Output:

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

Verschachtelte Schleifen

Was gibt dieses Programmsegment aus?

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; j <= i; j = j+1) {  
        System.out.print(i);  
    }  
    System.out.println();  
}
```

■ Output:

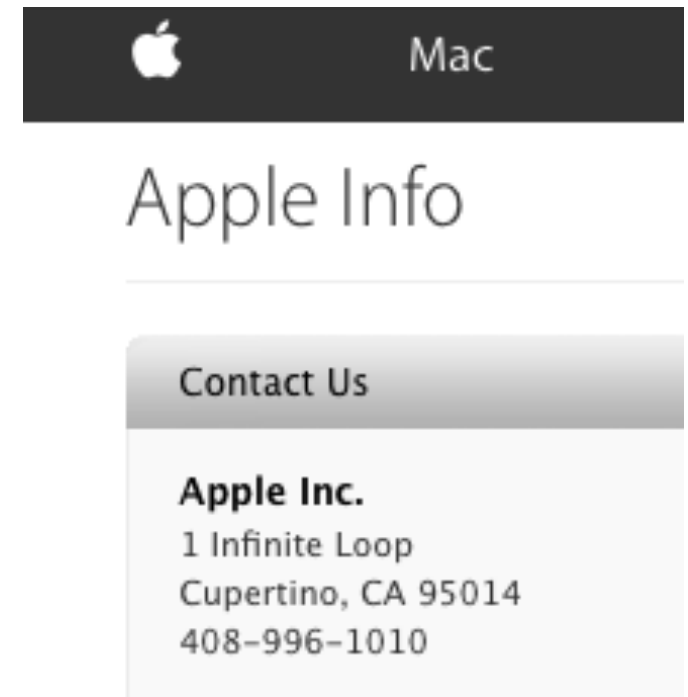
```
1  
22  
333  
4444  
55555
```

Mögliche Fehler

- Die Schleife *terminiert* nicht
 - Läuft und läuft und läuft
 - Endlosschleife («infinite loop»)

Mögliche Fehler

- Die Schleife *terminiert* nicht
 - Läuft und läuft und läuft
 - Endlosschleife («infinite loop»)



Mögliche Fehler

- Die Schleife *terminiert* nicht

- Läuft und läuft und läuft
- Endlosschleife («infinite loop»)

- **Beispiele:**

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; i <= 10; j = j+1) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

Mögliche Fehler

- Die Schleife *terminiert* nicht

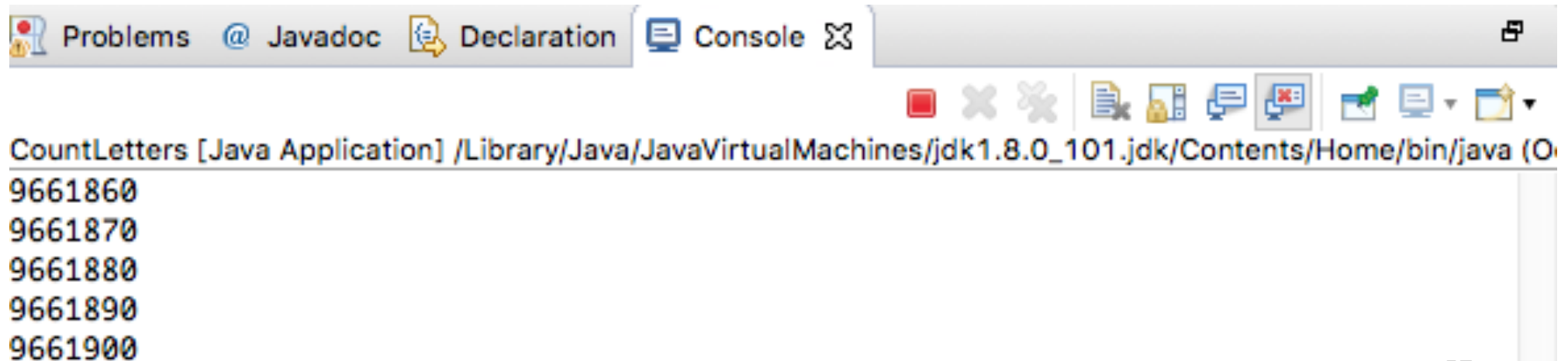
- Läuft und läuft und läuft
- Endlosschleife («infinite loop»)

- **Beispiele:**

```
for (int i = 1; i <= 5; i = i+1) {  
    for (int j = 1; j <= 10; i = i+1) {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

Was tun?

- Eclipse kann solche Programme stoppen.
- Klicken auf «Terminate» 



Mehr Beispiele

- Welche verschachtelten for-Schleifen erzeugen diesen Output?

....1

...2

..3

.4

5

Mehr Beispiele

- **Welche verschachtelten for-Schleifen erzeugen diesen Output?**

innere Schleife (für jede Zeile)

.....1

...2

..3

.4

5

äussere Schleife (5-mal da es 5 Zeilen gibt)

- **Wir müssen eine Ausgabezeile konstruieren:**

- mit einer äusseren Schleife für jede Zeile
- mit innerer(n) Schleife(n) für das Muster jeder Zeile

Äussere und innere Schleife

- Legen Sie erst die äussere Schleife fest, zählt von 1 bis zur Anzahl der Zeilen

```
for (int line = 1; line <= 5; line = line+1) {  
    ...  
}
```

- Analysieren Sie jede Zeile. Entdecken Sie das Muster:

ein paar Punkte (0 Punkte in der letzten Zeile) gefolgt von Zahl

....1

...2

..3

.4

5

Äussere und innere Schleife

- Legen Sie erst die äussere Schleife fest, zählt von 1 bis zur Anzahl der Zeilen

```
for (int line = 1; line <= 5; line = line+1) {  
    ...  
}
```

- Analysieren Sie jede Zeile. Entdecken Sie das Muster:

ein paar Punkte (0 Punkte in der letzten Zeile) gefolgt von Zahl

....1

...2

..3

.4

5

Beobachtung: Die Anzahl der Punkte hängt von der Zeilennummer ab.

Zahlenfolgen → Schleifen

```
for (int count = 1; count <= 5; count = count+1) {  
    System.out.print( ... );  
}
```

Welche Anweisung im Rumpf würde diesen Output ergeben:

4 7 10 13 16

```
for (int count = 1; count <= 5; count = count+1) {  
    System.out.print(3 * count + 1 + " ");  
}
```

Zahlenfolgen → Schleifen mit Tabellen

- Welche Anweisung im Rumpf würde diesen Output ergeben:
2 7 12 17 22
- **Zum Finden des Musters erstellen Sie eine Tabelle mit count und den Zahlen.**
 - Wenn sich count um 1 erhöht, sollte die Zahl um 5 heraufgehen.
 - Aber $\text{count} * 5$ ist zu gross (um 3), also subtrahieren wir 3.

count	Zahl in Folge	$5 * \text{count}$	$5 * \text{count} - 3$
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Weiteres Tabellen Beispiel

- Welche Anweisung im Rumpf würde diesen Output ergeben:

17 13 9 5 1

- **Konstruieren wir die Tabelle.**

- Wenn sich count um 1 erhöht, sollte die Zahl ...
- Aber dieses Produkt ist zu ...

count	Zahl in Folge	$-4 * \text{count}$	$-4 * \text{count} + 21$
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

Zurück zum Beispiel mit «for»-Schleife

- Konstruieren wir eine Tabelle

.....1
...2
..3
.4
5

line	# Punkte	-1 * line	-1 * line + 5
1	4	-1	4
2	3	-2	3
3	2	-3	2
4	1	-4	1
5	0	-5	0

- Um einen Buchstaben mehrfach zu drucken verwenden wir eine for-Schleife.

```
for (int j = 1; j <= 4; j = j+1) {  
    System.out.print(".");           // 4 Punkte  
}
```


Lösung mit «for»-Schleife

■ Antwort:

```
for (int line = 1; line <= 5; line = line+1) {  
    for (int j = 1; j <= (-1 * line + 5); j = j+1) {  
        System.out.print(".");  
    }  
    System.out.println(line);  
}
```

■ Output:

```
.....1  
....2  
...3  
..4  
.5  
5
```

Verschachtelte «for»-Schleifen

- Was ist der Output dieser verschachtelten Schleifen?

```
for (int line = 1; line <= 5; line = line+1) {  
    for (int j = 1; j <= (-1 * line + 5); j = j+1) {  
        System.out.print(".");  
    }  
    for (int k = 1; k <= line; k = k+1) {  
        System.out.print(line);  
    }  
    System.out.println();  
}
```

- Answer:

```
.....1  
...22  
..333  
.4444  
55555
```

Verschachtelte «for»-Schleifen Übung

- Verändern Sie das letzte Programm so dass dieser Output erzeugt wird:

....1

...2.

..3..

.4...

5....

Verschachtelte «for»-Schleifen Übung

Verändern Sie das letzte Programm so dass dieser Output erzeugt wird:

....1

...2.

..3..

.4...

5....

(Eine) Antwort:

```
for (int line = 1; line <= 5; line=line+1) {  
    for (int j = 1; j <= (-1 * line + 5); j=j+1) {  
        System.out.print(".");  
    }  
    System.out.print(line);  
    for (int j = 1; j <= (line - 1); j=j+1) {  
        System.out.print(".");  
    }  
    System.out.println();  
}
```

Übersicht

- **2.5 Schleifen (Loops)**
 - 2.5.1 «for»-Loops
 - 2.5.2 Verschachtelte Schleifen
 - 2.5.3 «while»-Loops

2.5.3 «while»-Schleifen

Klassifizierung von Schleifen

- **Bestimmte Schleife («*definite loop*»):** Anzahl der Ausführungen des Rumpfes («Iterationen») ist vor Beginn der Ausführung der Schleife bekannt.
 - Die «for»-Schleifen waren bisher immer bestimmte Schleifen.
 - Drucke "hello" 10-mal.
 - Finden Sie alle Primzahlen $<$ einer ganzen Zahl n .
 - Drucken Sie jede ungerade Zahl zwischen 7 und 91.
- **Unbestimmte Schleife («*indefinite loop*»):** Anzahl der Iterationen ist nicht vorher bekannt.

Beispiele von unbestimmten Schleifen

- **Unbestimmte Schleife («*indefinite loop*»):** Anzahl der Iterationen ist nicht vorher bekannt.
- **Beispiele:**
 - Lesen Sie den Input von der Konsole bis der Benutzer eine nicht-negative ganze Zahl eingeben hat.
 - Wiederholen Sie bis der Benutzer ein «q» eingegeben hat.
 - Lesen Sie eine Datei bis drei aufeinanderfolgende Sätze mit einem «!» enden.
 - Nehmen Sie Beiträge (via crowdfunding) entgegen bis das Ziel erreicht ist.

Die «while»-Schleife

- Eine «while»-Schleife führt Schleifenrumpf so lange aus wie der boolesche Ausdruck test den Wert true ergibt

```
while (test) {  
    statement(s);  
}
```

- **Beispiel:**

```
int num = 1;                                // initialization  
while (num*num <= 2000) {                    // test  
    System.out.print(num + " ");  
    num = num * 2;                           // update  
}  
// output:  1 2 4 8 16 32
```

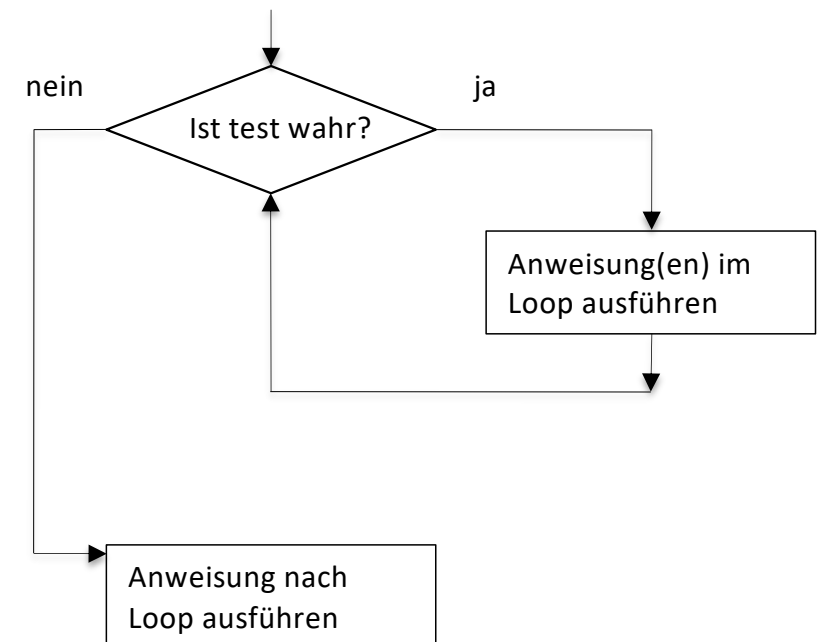
Die «while»-Schleife

- Eine «while»-Schleife führt Schleifenrumpf so lange aus wie der boolesche Ausdruck test den Wert true ergibt

```
while (test) {  
    statement(s);  
}
```

- **Beispiel:**

```
int num = 1;  
while (num*num <= 2000) {  
    System.out.print(num + " ");  
    num = num * 2;  
}  
// output: 1 2 4 8 16 32
```



Beispiel «while»-Schleife

```
// finds the first factor of 91, other than 1
int n = 91;
int factor = 2;
while (n % factor != 0) {
    factor = factor + 1;
}
System.out.println("First factor is " + factor);

// output: First factor is 7
```

- while ist hier besser als for weil wir nicht wissen wie oft wir den Zähler erhöhen müssen um den 1. Faktor zu finden

Übersicht

- **2.6 Methoden, Teil 2**
 - 2.6.1 Methoden mit Parametern
 - 2.6.2 Rückgabewerte
 - 2.6.3 Namensräume

2.6.1 Methoden mit Parametern

«for»-loops erlauben Wiederholungen

- Javas «for»-loop Anweisung wiederholt Anweisungen

```
for (int i = 1; i <= 3; i = i + 1) {  
  
    // repeat 3 times  
    operand1 = rand.nextInt(10) + 1;  
    operand2 = rand.nextInt(10) + 1;  
    System.out.print(operand1 + " + " + operand2 + " = ");  
    if (console.nextInt() != (operand1+operand2)) {  
        wrong = wrong + 1;  
    }  
  
}
```

Wiederverwendung durch Methoden

- Wenn wir dieses Programm wiederverwenden wollen dann definieren wir eine Methode
 - Erspart vielfaches Schreiben der Anweisungen
 - Änderungen nur an einer Stelle
- **Beispiel:**

```
public static void addTest() {    // NICHT vollstaendig!
    for (int i = 1; i <= 3; i = i + 1) { // repeat 3 times
        operand1 = rand.nextInt(10) + 1;
        operand2 = rand.nextInt(10) + 1;
        System.out.print(operand1 + " + " + operand2 + " = ");
        if (console.nextInt() != (operand1+operand2)) {
            wrong = wrong + 1;
        }
    }
}
```

***Wiederverwendung* erfordert Flexibilität**

- Diese «Lösung» liefert immer 3 Aufgaben
- Was wenn wir auch andere Kombinationen wollen?
 - Könnten addTest1, addTest2, addTest3, ... definieren
- Wir brauchen einen Weg, die Anzahl Wiederholungen der Situation anzupassen
 - *Parametrisierung*: mit (veränderbaren) Parametern versehen

Parametrisierung

- **Parameter: Ein Wert den eine aufgerufene Methode von der aufrufenden Methode erhält.**
 - Wenn wir eine Methode deklarieren dann geben wir an dass diese Methode einen Parameter braucht
`addTest(int anzahl)`
 - Wenn wir die Methode aufrufen geben wir Wert für den Parameter an
`addTest(3)` oder `addTest(5)`
- **Methode vielseitiger einsetzbar aber Entwicklung anspruchsvoller**

Parameterdeklarationen

Gibt an dass eine Methode einen Parameter braucht um ausgeführt werden zu können

```
public static void methodName ( type name ) {  
    statement;  
}
```

name: «Parameter Variable»

type: Typ der «Parameter Variable»

- Basistyp (int, double,... z.Zt.)

Parameterdeklarationen

Gibt an dass eine Methode einen Parameter braucht um ausgeführt werden zu können

```
public static void methodName ( type name ) {  
    statement;  
}
```

Wenn echoPin aufgerufen wird dann muss der Aufrufer einen int Wert angeben.



■ Beispiel:

```
public static void echoPin(int code) {  
    System.out.println("Die Geheimnummer ist: " + code);  
}
```

Wert(e) für Parameter

Beim Aufruf der Methode muss ein Wert für den Parameter angegeben werden.

name (**expression**);

- **Beispiel:**

```
public static void main(String[] args) {  
    echoPin(42);  
    echoPin(12345);  
}
```

- **Output**

Die Geheimnummer ist: 42

Die Geheimnummer ist: 12345

Wie werden Parameter übergeben?

- **Übergeben: vom Aufrufer zur aufgerufenen Methode**
- **Wenn eine Methode aufgerufen wird dann:**
 - Der Wert wird in der Parameter Variable gespeichert
 - Die Anweisungen der Methode werden ausgeführt (mit diesem Wert für die Parameter Variable).

Zurück zum Beispiel:

Methode mit Parameter

```
import java.util.Random;  
import java.util.Scanner;
```

```
public class Beispiel {  
    public static void main (String[] args) {  
        addTest(3);  
    } // end main  
  
    public static void addTest(int anzahl) {  
        Random rand = new Random();  
        Scanner console = new Scanner(System.in);  
        int operand1; int operand2; int wrong = 0;  
        for (int i = 1; i <= anzahl; i = i + 1) { // repeat anzahl times  
            operand1 = rand.nextInt(10) + 1;  
            operand2 = rand.nextInt(10) + 1;  
            System.out.print(operand1 + " + " + operand2 + " = ");  
            if (console.nextInt() != (operand1+operand2)) {  
                wrong = wrong + 1;  
            }  
        }  
    } // end addTest  
}
```

Parameter

- Ein Parameter in der Deklaration einer Methode heisst *formaler Parameter* («formal parameter»).
- Formal Parameter: deklariert Variable für Methode
- Beim Aufruf der Methode muss ein Wert für den Parameter angegeben werden: `name (expression)` ;
 - Übergebener Wert wird in Parameter Variable gespeichert (die so initialisiert/definiert wird)
 - Übergebener Wert heisst *tatsächlicher Argument Wert* («actual argument value») oder *tatsächlicher Argument Ausdruck* («actual argument expression»)
 - Aktuell: augenblicklich, derzeitig [Duden]
- Oder «Argument» wenn kein Missverständnis

Parameter Variable

Die Parameter Variable kann in der Methode wie jede Variable verwendet werden (z.B. Anzahl der Iterationen einer Schleife kontrollieren)

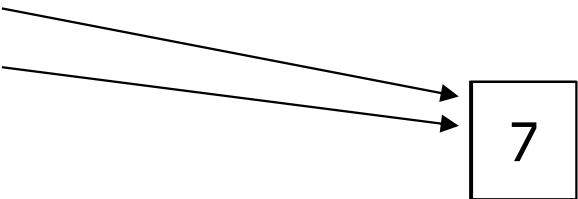
```
public static void main(String[] args) {  
    printPunkt(3);  
}  
  
public static void printPunkt(int times) {  
    for (int i = 1; i <= times; i = i+1) {  
        System.out.print(".");  
    }  
    System.out.println();  
}
```

Output:

...

Wie werden Parameter übergeben?

```
public static void main(String[] args) {  
    printPunkt(3);  
    printPunkt(7);  
}
```



```
public static void printPunkt(int times) {  
    for (int i = 1; i <= times; i = i+1) {  
        System.out.print(".");  
    }  
    System.out.println();  
}
```

Output:

```
...  
.....
```

Mögliche Fehler

- Wenn eine Methode ein Argument erwartet dann muss dieses auch übergeben werden.

```
printPunkt();    // ERROR: parameter value required
```

- Der Wert (bzw. Ausdruck) muss den richtigen Typ haben

```
printPunkt(3.7); // ERROR: must be of type int
```

- Die Regeln für Umwandlungen gelten auch hier

Mehrere Parameter

- Eine Methode kann mehrere Parameter definieren (getrennt durch Komma («,»)) in Deklaration und im Aufruf)
 - Wenn die Methode aufgerufen wird muss ein Wert für jeden Parameter angegeben werden
- **Deklaration:**

```
public static void method(type name, ..., type name) {  
    statement;  
}
```
- **Aufruf:** `method (value, value, ..., value);`

Mehrere Parameter

- Eine Methode kann mehrere Parameter definieren (getrennt durch Komma («,»)) in Deklaration und im Aufruf)
 - Wenn die Methode aufgerufen wird muss ein Wert für jeden Parameter angegeben werden
- **Deklaration:**

```
public static void method(type1 name1, ..., typeN nameN) {  
    statement;  
}
```
- **Aufruf:** `method (value1, value2, ..., valueN);`

Beispiel mit mehreren Parametern

```
public static void main (String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("Input lower bound: ");
    int low = console.nextInt();
    System.out.print("Input upper bound: ");
    int up = console.nextInt();
    printOdd(low, up);
    printOdd(-up, -low);
}

public static void printOdd(int from, int to) {
    for (int i=from; i<=to; i = i+1) {
        if (i%2==1) {
            System.out.println(i);
        }
    }
}
```

Wie werden Parameter übergeben?

- **Wenn eine Methode aufgerufen wird:**
 - Wert für Parameter wird von Aufrufer berechnet und übergeben
 - Wert wird von aufgerufener Methode in der Parameter Variable gespeichert
 - Die Anweisungen der aufgerufenen Methode werden ausgeführt (anfangs mit diesem Wert für die Parameter Variable).
- **Der Wert, den der Aufrufer übergibt, kann durch einen Ausdruck (Expression) gegeben sein**
 - Der Wert des Expressions wird berechnet und übergeben
 - Die aufgerufene Methode erhält den Wert und hat keine Kenntnis davon wie der Wert berechnet wurde

Wie werden Parameter übergeben?

- **Wenn eine Methode aufgerufen wird:**
 - Wert für Parameter wird von Aufrufer berechnet und übergeben
 - Wert wird von aufgerufener Methode in der Parameter Variable gespeichert
 - Die Anweisungen der aufgerufenen Methode werden ausgeführt (anfangs mit diesem Wert für die Parameter Variable).
- **Der Wert, den der Aufrufer übergibt, kann durch eine Variable gegeben sein**
 - Der Wert der Variable wird übergeben
 - Die aufgerufene Methode erhält den Wert und hat keine Kenntnis davon wo der Wert herkam

Übergabe von Werten («Value semantics»)

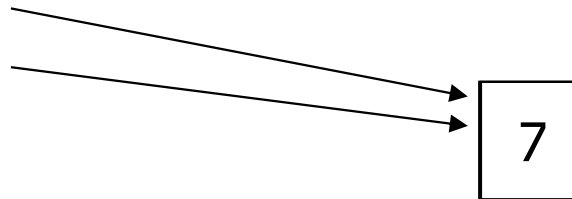
- **Wenn ein (aktuelles) Argument eines Basistyps (z.B. `int`, `double`, `boolean`) übergeben wird, dann wird der Wert vom Aufrufer kopiert («value semantics»)**

Wenn der aktuelle Parameter durch einen Ausdruck bestimmt wird, dann wird der Ausdrucks evaluiert und das Ergebnis kopiert.

- **Der kopierte Wert initialisiert die Parameter Variable in der aufgerufenen Methode**

Übergabe von Werten («Value semantics»)

```
public static void main(String[] args) {  
    int k = 3;  
    printPunkt(k);  
    printPunkt(k+4);  
}
```



```
public static void printPunkt(int times) {  
    for (int i = 1; i <= times; i = i+1) {  
        System.out.print(".");  
    }  
    System.out.println();  
}
```

Output:

...

.....

Übergabe von Werten («Value semantics»)

- Wenn ein (aktueller) Parameter eines Basistyps (z.B. `int`, `double`, `boolean`) übergeben wird, dann wird der Wert vom Aufrufer kopiert («value semantics»)
- **Veränderungen der Parameter Variable (des formalen Parameters) in der *aufgerufenen* Methode haben keine Auswirkung auf die *aufrufende* Methode**

Übergabe von Werten («Value semantics»)

```
public static void strange(int x) {  
    x = x + 1;  
    System.out.println("1. x = " + x);  
}
```

```
public static void main(String[] args) {  
    int x = 23;  
    strange(x);  
    System.out.println("2. x = " + x);  
    ...  
}
```

Output:

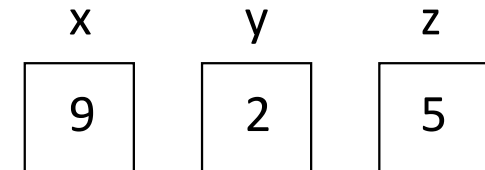
```
1. x = 24  
2. x = 23
```

Übergabe von Werten («Value semantics»)

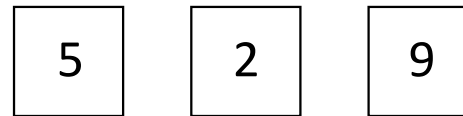
- Wenn ein (aktuelles) Argument durch eine Variable *V* eines Basistyps (`int`, `double`, `boolean`) bestimmt wird dann wird der Wert dieser Variable kopiert («value semantics»):
- Name der Variable *V* (oder des formalen Parameters) ist unwichtig.

"Parameter Übergabe" Problem

```
public class ParameterMystery {  
    public static void main(String[] args) {  
        int x = 9;  
        int y = 2;  
        int z = 5;
```

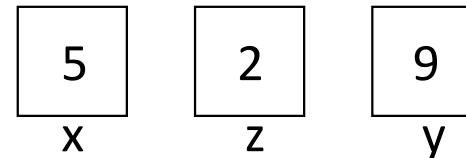


```
        mystery(z, y, x);
```



```
        mystery(y, x, z);
```

```
    }
```



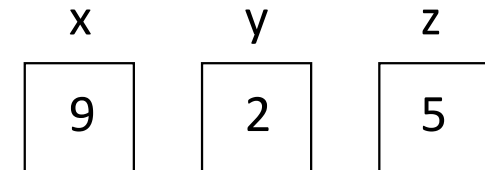
```
    public static void mystery(int x, int z, int y) {  
        System.out.println(z + " and " + (y - x));  
    }
```

```
}
```

Output: 2 and 4

"Parameter Übergabe" Problem

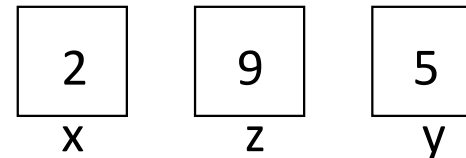
```
public class ParameterMystery {  
    public static void main(String[] args) {  
        int x = 9;  
        int y = 2;  
        int z = 5;
```



```
        mystery(z, y, x);
```

```
        mystery(y, x, z);
```

```
    }
```



```
    public static void mystery(int x, int z, int y) {  
        System.out.println(z + " and " + (y - x));  
    }
```

```
}
```

Output: 9 and 3

2.6.2 Ergebnis Rückgabe für Methoden

```

import java.util.*;
class PrintPrimes1 {

public static void main (String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("Input max: ");
    int max = console.nextInt();

    if (max >= 2) {
        printPrimes(max);
    }
}

public static void printPrimes(int limit)
    // Prints all prime numbers up to limit, limit >= 2

    System.out.print("2");
    for (int i = 3; i <= limit; i = i + 1) {
        if ( /* isPrime(i) */ ) {
            System.out.print(", " + i);
        }
    }
    System.out.println(); // to end output
}
}

```


Ergebnis Rückgabe

- **Parameter erlauben Kommunikation vom Aufrufer zur aufgerufenen Methode**
 - Bisher waren die Methoden sehr einfach
 - Methode als «Ersatz» für Anweisungen in `main` (oder anderer Methode)
 - Methoden können aber mehr ...
- **Ein Rückgabewert («return value») erlaubt der aufgerufenen Methode dem Aufrufer einen Wert zu übermitteln**
 - Damit eröffnen sich neue Möglichkeiten der Komposition

```
{  
...  
name ( 7+i, true);  
...  
}
```

public static

name(int k,
boolean b) {

int result;
while (k < 10) { k=k+1; ... }

if (b) { ... }

// result
}

```
{  
...  
name ( 7+i, true);  
...  
}
```

```
public static      name(int k,  
                        boolean b)  {  
    int result;  
    while (k < 10) { k=k+1; ... }  
  
    if (b) { ... }  
  
    // result  
}
```

Rückgabe eines Wertes

- Ein Rückgabewert muss deklariert werden

```
public static type name(parameters) {  
    statements;  
    ...  
    return expression;  
}
```

- Es gelten die selben Regeln für *type* wie bei der Deklaration von Variablen und Parametern
- Keyword `void` bedeutet: kein Rückgabewert

Rückgabeeanweisung

- Das «return»-Statement («Rückgabe Anweisung») wertet einen Ausdruck aus
 - Der Wert wird dann an den Aufrufer «zurückgegeben»
 - Der Ausdruck muss einen Wert des Typs *type* (der Methoden Deklaration) ergeben.
- Die Ausführung der «return»-Anweisung beendet die aufgerufene Methode.

Rückgabeeanweisung («return»-Statement)

- **return: Liefere einen Wert ab als das Ergebnis dieser Methode**
 - «sende» das Ergebnis zum Aufrufer
 - Das Gegenstück zu Parametern:
 - Parameters schicken Werte *in* die aufgerufene Methode, vom Aufrufer
 - Rückgabewerte schicken Werte *aus* der Methode zum Aufrufer
 - Ein Methodenaufruf kann Teil eines Ausdrucks sein.
 - Aufrufer muss den Wert «annehmen»

```
{  
...  
int size = name ( 7+i, true);  
...  
}
```

```
public static int name(int k,  
                        boolean b) {  
    int result;  
    while (k < 10) { k=k+1; ... }  
  
    if (b) { ... }  
  
    return result;  
}
```

Beispiellösung mit Rückgabe eines Wertes

```
public static boolean isPrime (int arg){  
    // Determine how many factors the given number has.  
    boolean found = false;  
    int step = 2;  
  
    while (!found) {  
        if (arg % step == 0) {  
            found = true; // factor found  
        }  
        else {  
            step = step + 1; // keep on searching  
        }  
    }  
    // factor == arg: prime found  
    return (step == arg);  
}
```


Rückgabe eines Wertes

Beispiel:

```
// Returns the slope of the line between the given points.  
public static double slope(int x1, int y1, int x2, int y2) {  
    double dy = y2 - y1;  
    double dx = x2 - x1;  
    return dy / dx;  
}
```

`slope(1, 3, 5, 11)` liefert 2.0

return ohne einen Wert

- Wenn eine Methode keinen Wert zurück liefert dann braucht ein «return»-Statement keinen Wert zu schicken.

```
public static void printPoint(int x, int y) {  
    System.out.println("x = " + x + " y = " + y) ;  
    return;  
}
```

- In dem Fall kann man das «return»-Statement auch weglassen (meine Empfehlung)

Weitere Beispiele

```
// Converts degrees Fahrenheit to Celsius.
```

```
public static double fToC(double degreesF) {  
    double degreesC = 5.0 / 9.0 * (degreesF - 32);  
    return degreesC;  
}
```

```
// Computes triangle hypotenuse length given its side lengths.
```

```
public static double hypotenuse(int a, int b) {  
    double c = squareRoot(a * a + b * b);  
    return c;  
}
```

Weitere Beispiele

Ein «return»-Statement kann auch einen (arithmetischen oder booleschen) Ausdruck verwenden

```
public static double fToC(double degreesF) {  
    return 5.0 / 9.0 * (degreesF - 32);  
}
```

Mögliche Fehler: Resultat nicht gespeichert

- Ein «return»-Statement schickt einen Wert an den Aufrufer
- Namen, die in der aufgerufenen Methode verwendet werden, sind belanglos (für den Aufrufer)

Was ist hier nicht richtig?

```
public static void main(String[] args) {  
    slope(0, 0, 6, 3);  
    // Problem: return value not used/stored  
}
```

```
public static double slope(int x1, int x2, int y1, int y2) {  
    double dy = y2 - y1;  
    double dx = x2 - x1;  
    double result = dy / dx;  
    return result;  
}
```

Was ist hier nicht richtig?

```
public static void main(String[] args) {  
    slope(0, 0, 6, 3);  
    System.out.println("The slope is " + result); // ERROR:  
}                                                    // result not defined
```

```
public static double slope(int x1, int x2, int y1, int y2) {  
    double dy = y2 - y1;  
    double dx = x2 - x1;  
    double result = dy / dx;  
    return result;  
}
```

Den Fehler vermeiden

- **return** schickt den *Wert* der Variable zurück zum Aufrufer.
 - Der zurückgegebene Wert muss gespeichert werden – oder in einem Ausdruck verwendet werden.
- **Der Compiler generiert keine Warnung oder Fehlermeldung wenn dies vergessen wird.**

Den Fehler vermeiden

```
public static void main(String[] args) {  
    double s = slope(0, 0, 6, 3);  
    System.out.println("The slope is " + s);  
}
```

```
public static double slope(int x1, int x2, int y1, int y2) {  
    double dy = y2 - y1;  
    double dx = x2 - x1;  
    double result = dy / dx;  
    return result;  
}
```

«return»-Anweisungen

- Eine Methode kann mehrere «return»-Anweisungen enthalten.
 - Sinnvoll für Fallunterscheidungen
- Eine Methode die einen Rückgabewert deklariert *muss* eine (oder mehrere) «return»-Anweisung(en) enthalten

if/else mit return

```
// Returns the larger of the two given integers.  
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

- **Methoden können ein «return»-Statement in durch if/else kontrollierten Blöcken enthalten**
 - Das «return»-Statement am Ende eines Pfades liefert den Rückgabewert für diese Methode.

if/else mit return

- **Die Ausführung eines «return»-Statements beendet die aufgerufene Methode.**
 - Einem return sollten keine weiteren Anweisungen folgen
- **Alle Pfade durch eine Methode müssen ein «return»-Statement enthalten**
 - Wenn die Methode einen Rückgabewert deklariert hat

Alle Pfade ...

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    // Error: not all paths return a value  
}
```

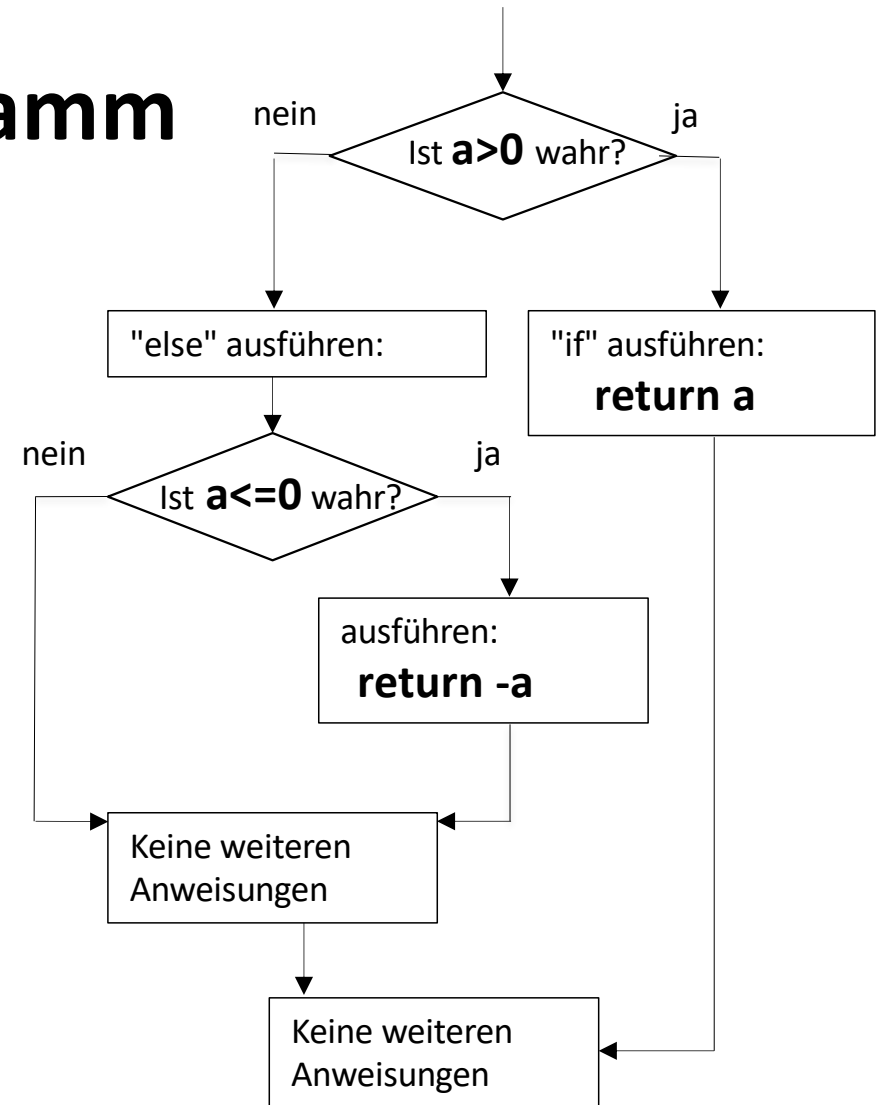
■ Der Compiler ist manchmal naiv:

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else if (b >= a) {  
        return b;  
    }  
}
```

Der Compiler meint dass es einen Pfad ohne return gibt.

So versteht Java das Programm

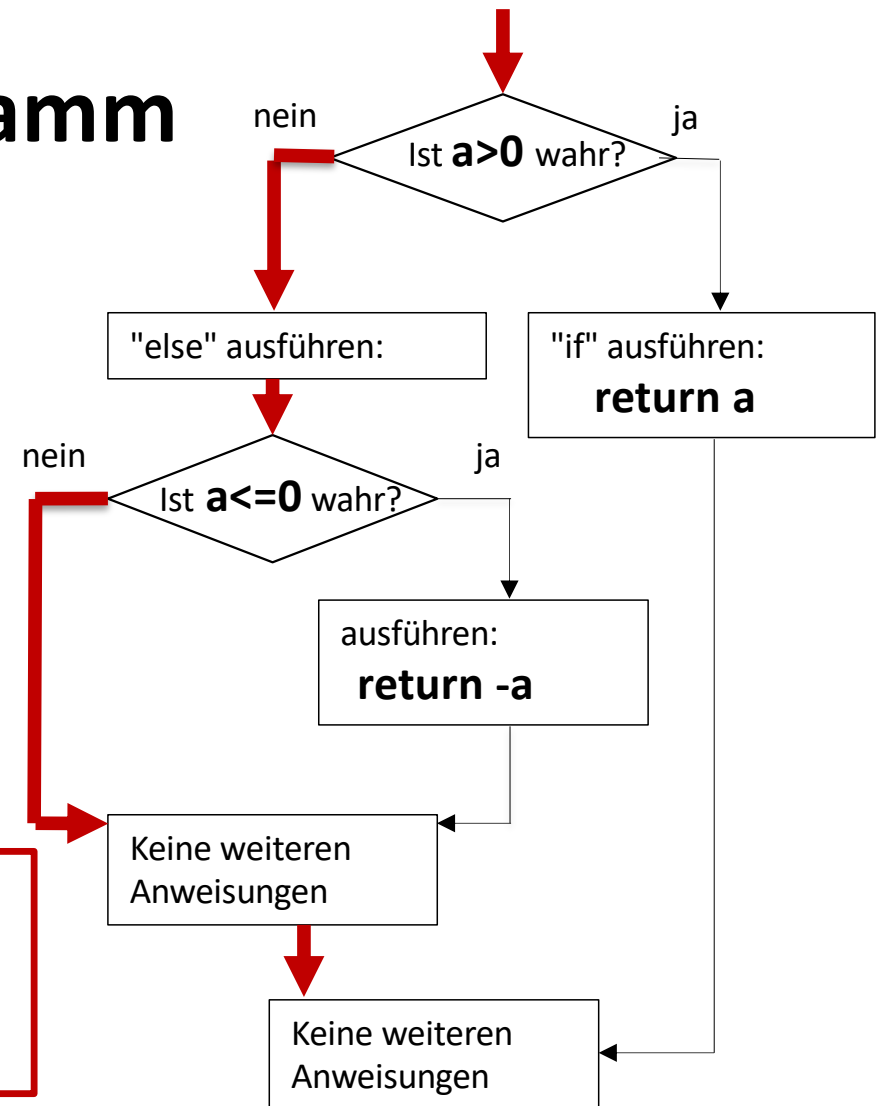
```
int foo(int a) {  
    if (a > 0) {  
        return a;  
    } else {  
        if (a <= 0) {  
            return -a;  
        }  
    }  
}
```



So versteht Java das Programm

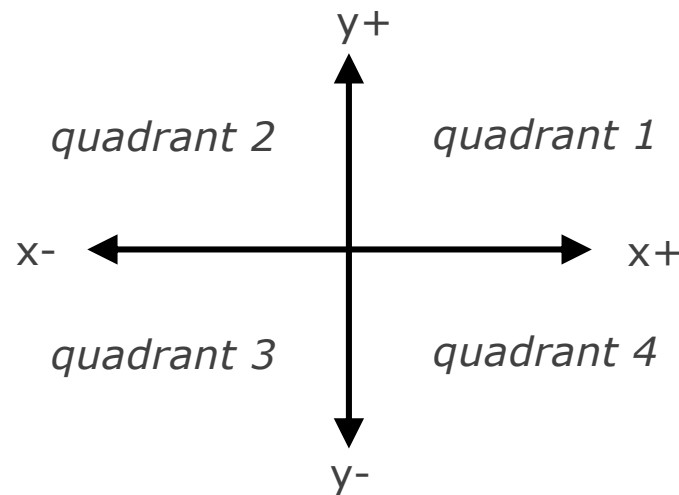
```
int foo(int a) {  
    if (a > 0) {  
        return a;  
    } else {  
        if (a <= 0) {  
            return -a;  
        }  
    }  
}
```

**Jeder Pfad ist
möglich: Daher
Fehlermeldung**



if/else, return **Beispiel**

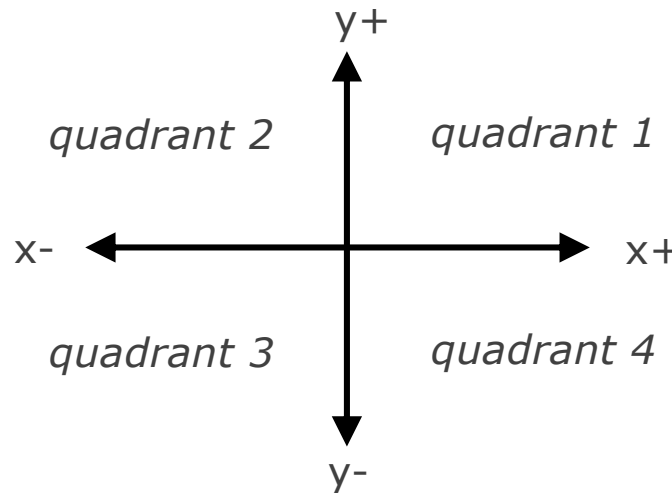
- Schreiben Sie eine Methode `quadrant` die für ein Paar von reellen Zahlen den Quadranten liefert in dem dieser Punkt liegt.



- Beispiel: `quadrant(-4.2, 17.3)` liefert 2
- Fällt der Punkt auf eine der Achsen des Koordinatensystems liefere 0.¹⁶⁷

if/else, return **Beispiel**

- Schreiben Sie eine Methode `quadrant` die für ein Paar von reellen Zahlen den Quadranten liefert in dem dieser Punkt liegt.



Schreiben
Sie Ihre
Lösung als
Clicker
Antwort!

- Beispiel: `quadrant(-4.2, 17.3)` liefert 2
- Fällt der Punkt auf eine der Achsen des Koordinatensystems liefere 0.

if/else, return Beispiel

```
public static      quadrant(          ) {  
  
  
  
  
  
  
  
  
  
}
```

if/else, return Beispiellösung

```
public static int quadrant(double x, double y) {  
    if (x > 0 && y > 0) {  
        return 1;  
    } else if (x < 0 && y > 0) {  
        return 2;  
    } else if (x < 0 && y < 0) {  
        return 3;  
    } else if (x > 0 && y < 0) {  
        return 4;  
    } else {          // at least one coordinate equals 0  
        return 0;  
    }  
}
```

if/else, return Beispiellösung

```
public static int quadrant(double x, double y) {  
    if (x > 0.0 && y > 0.0) {  
        return 1;  
    } else if (x < 0.0 && y > 0.0) {  
        return 2;  
    } else if (x < 0.0 && y < 0.0) {  
        return 3;  
    } else if (x > 0.0 && y < 0.0) {  
        return 4;  
    } else {          // at least one coordinate equals 0  
        return 0;  
    }  
}
```

if/else, return weitere Beispiele

- **Schreiben Sie eine Methode `countFactors` die die Anzahl der Faktoren (Teiler) einer Zahl liefert.**
 - `countFactors(24)` liefert 8 da 1, 2, 3, 4, 6, 8, 12, und 24 alle Teiler von 24 sind.

if/else, return weitere Beispiele

■ Lösung:

```
// Returns how many factors the given number has.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i = i + 1) {
        if (number % i == 0) {
            count = count + 1; // i is a factor of number
        }
    }
    return count;
}
```

wp(S,Q): Was ist die weakest precondition?

Alles int Variable, kein Over-/Underflow

1. `x = x + 1;` // S ist ein Statement
`{ x > 0 }` // Q

2. `k = j * 2;` // S ist eine Folge von Statements
`m = k + 1;`
`{ k > 0 && m <= 3 }` // Q

3. `if (y > x) {` // S ist ein if-Statement
`max = y;`
`}` `else {`
`max = x;`
`}`
`{ max >= x && max >= y }` // Q

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { $x > -1$ }

$x = x + 1;$

{ $x > 0$ }

2. { $j == 1$ }

$k = j * 2;$

$m = k + 1;$

{ $k > 0 \ \&\& \ m \leq 3$ }

3. { $(b \ \&\& \ wp(S1,Q)) \ || \ (!b \ \&\& \ wp(S2,Q))$ }

if ($y > x$) {

$max = y;$

} else {

$max = x;$

}

{ $max \geq x \ \&\& \ max \geq y$ }

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { $x > -1$ }

$x = x + 1;$

{ $x > 0$ }

2. { $j == 1$ }

$k = j * 2;$

$m = k + 1;$

{ $k > 0 \ \&\& \ m \leq 3$ }

3. { $(b \ \&\& \ wp(S1,Q)) \ || \ \dots$ }

if ($y > x$) {

$\max = y;$

} else {

$\max = x;$

}

{ $\max \geq x \ \&\& \ \max \geq y$ }

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { $x > -1$ }

$x = x + 1;$

{ $x > 0$ }

2. { $j == 1$ }

$k = j * 2;$

$m = k + 1;$

{ $k > 0 \ \&\& \ m \leq 3$ }

3. { $(y > x \ \&\& \text{wp}(\text{max} = y, \text{max} \geq x \ \&\& \text{max} \geq y)) \ || \dots$ }

if ($y > x$) {

$\text{max} = y;$

} else {

$\text{max} = x;$

}

{ $\text{max} \geq x \ \&\& \ \text{max} \geq y$ }

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { x > -1 }

x = x + 1;

{ x > 0 }

2. { j == 1 }

k = j * 2;

m = k + 1;

{ k > 0 && m <= 3 }

3. { (y > x && y >= x && y >= y) || ... }

if (y > x) {

max = y;

} else {

max = x;

}

{ max >= x && max >= y }

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { $x > -1$ }

$x = x + 1;$

{ $x > 0$ }

2. { $j == 1$ }

$k = j * 2;$

$m = k + 1;$

{ $k > 0 \ \&\& \ m \leq 3$ }

3. { $(y > x) \ || \ (y \leq x)$ }

if ($y > x$) {

$max = y;$

} else {

$max = x;$

}

{ $max \geq x \ \&\& \ max \geq y$ }

wp(S,Q): Was ist die weakest precondition?

Poll

Alles int Variable, kein Over-/Underflow

1. { $x > -1$ }

$x = x + 1;$

{ $x > 0$ }

2. { $j == 1$ }

$k = j * 2;$

$m = k + 1;$

{ $k > 0 \ \&\& \ m \leq 3$ }

3. { true }

if ($y > x$) {

$max = y;$

} else {

$max = x;$

}

{ $max \geq x \ \&\& \ max \geq y$ }