

252-0027

Einführung in die Programmierung

2.0 Einfache Java Programme

Thomas R. Gross

**Department Informatik
ETH Zürich**

252-0027

Einführung in die Programmierung

2.6 Methoden, Teil 2

Thomas R. Gross

**Department Informatik
ETH Zürich**

Übersicht

- **2.6 Methoden, Teil 2**
 - 2.6.1 Methoden mit Parametern
 - 2.6.2 Rückgabewerte
 - 2.6.3 Namensräume
- **2.7 Strings**
- **2.8 Nochmals Schleifen**

2.6.3 Sichtbarkeit von Variablennamen

- **Namesräume («Scope»): Bereich in dem ein Name sichtbar ist**
 - Dann kann die Variable gelesen/modifiziert werden
 - Dann kann eine Methode aufgerufen werden (später)
- **1. Approximation für das Innenleben von Methoden**
 - Weitere Aspekte in späteren Vorlesungen

Scope (Sichtbarkeitsbereich)

scope: Der Teil eines Programm in dem eine Variable sichtbar ist.

- Variable müssen deklariert sein *bevor* sie sichtbar sind
 - Deklarationen müssen eindeutig sein
- Sichtbar von Deklaration bis zum Ende des Blocks für den die Variable deklariert ist

Block: durch { und } begrenzt

{ und } strukturieren ein Programm

```
public static void fct(int j) {
```

```
    int i;
```

i sichtbar

```
    int k;
```

k sichtbar

j sichtbar

```
}
```

{ und } strukturieren ein Programm

```
if (...) {
```

```
    int i;
```

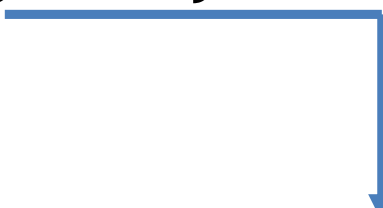
↓ i sichtbar


```
} else {
```

```
}
```

```
...
```

{ und } strukturieren ein Programm

```
for (int i = 0; ...; ...) {  
     i sichtbar  
}
```

```
...  
for (int i = 1; ...; ...) {  
     (anderes) i sichtbar  
}
```

```
...
```


Scope (Sichtbarkeitsbereich)

scope: Der Teil eines Programm in dem eine Variable sichtbar ist

- Variable müssen deklariert sein *bevor* sie sichtbar sind
 - Deklarationen müssen eindeutig sein
- Sichtbar von Deklaration bis zum Ende des Blocks (der durch { und } angegeben wird)
 - Eine Variable die in einer «for»-Schleife deklariert wurde kann nur im Rumpf der Schleife verwendet werden.
 - Eine Variable die in einer Methode deklariert wurde existiert nur in der Methode.

Blöcke können geschachtelt sein

- Loops in Methoden
- Loops in Loops -- geschachtelte Schleifen («nested loops»)
- (Java: Methoden können *nicht* in anderen Methoden geschachtelt sein.)

Scope (Sichtbarkeitsbereich)

```
public static void example() {  
    int x = 3;  
    x = x * x;  
    for (int i = 1; i <= 10; i = i+1) {  
        System.out.println(x+i);  
    } // i no longer exists here  
    System.out.println(x);  
} // x ceases to exist here
```

i's scope

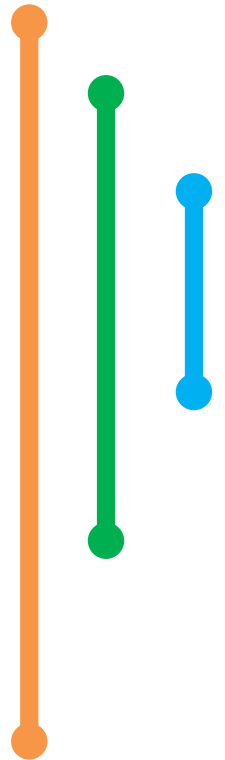
x's scope

// i no longer exists here

// x ceases to exist here

Scope (Sichtbarkeitsbereich)

```
public static void example(int x) {  
    for (int i = 1; i <= 10; i=i+1) {  
        for (int j = i; j<=10; j=j+1) {  
            System.out.print(x + i + j + " ");  
        } // j no longer exists here  
        System.out.println(i);  
    } // i no longer exists here  
    System.out.println(x);  
} // x no longer exists here
```



Folgen der Sichtbarkeitsregeln

- Variable ohne überlappenden Sichtbarkeitsbereich können den selben Namen haben.

```
for (int i = 1; i <= 100; i=i+1) {  
    System.out.print("/");  
}  
for (int i = 1; i <= 100; i=i+1) {    // OK  
    System.out.print("\\");  
}  
int i = 5;                          // OK: outside of loop's scope
```

Folgen der Sichtbarkeitsregeln

- Eine Variable kann in einem Sichtbarkeitsbereich nur *einmal* deklariert werden.

```
for (int i = 1; i <= 100 * line; i=i+1) {  
    int i = 2;                // ERROR: overlapping scope  
    System.out.print("/");  
}  
i = 4;                        // ERROR: outside scope
```

- Eine Variable kann nicht ausserhalb ihres Sichtbarkeitsbereiches verwendet werden

Folgen der Sichtbarkeitsregeln

- Eine Variable kann in einem Sichtbarkeitsbereich nicht mehrmals deklariert werden.

```
for (int i = 1; i <= 100 * line; i=i+1) {  
    for (int i = 2; i < line; i=i+1) {  
        // ERROR: overlapping scope  
        // variable i is already defined in method ...  
        System.out.print("/");  
    }  
}
```

Sichtbarkeitsregeln für Parameter Variable

- **Die selben Regeln gelten auch für Parameter Variable**

Scope (Sichtbarkeitsbereich)

```
public static void function(int k) {  
    int x = 3;  
    int y = k+x;  
    System.out.println(y);  
} // k ceases to exist here
```

y's scope

k's scope

Scope (Sichtbarkeitsbereich)

```
public static void function(int k) {  
    int x = 3;  
    int y = anotherFct(k+x);  
    System.out.println(y);  
} // k ceases to exist here
```

y's scope {

x's scope {


```
public static void otherFct(int x) {  
    int y = 5;  
    System.out.println(x+y);  
}
```

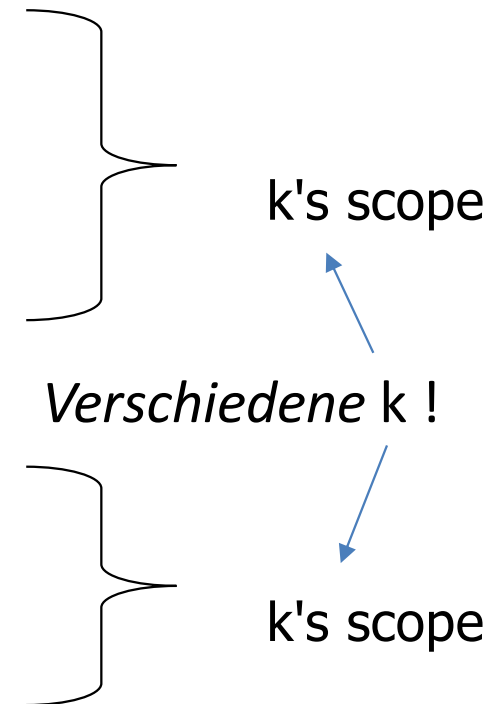
y's scope {

x's scope {

Scope (Sichtbarkeitsbereich)

```
public static void function(int k) {  
    int x = 3;  
    int y = anotherFct(k+x);  
    System.out.println(y);  
} // k ceases to exist here
```

```
public static void otherFct(int k) {  
    int y = 5;  
    System.out.println(k+y);  
} // k ceases to exist here
```



```
public static void f(int x) {  
    int i = 3;  
    // A  
    for (int j = 0; j < x; j=j+1) {  
        // B  
        if (j == x-1) {  
            int k = 0;  
            // C  
            k = i;  
        } else {  
            int m = 1;  
            // D  
        }  
        // E  
    }  
    //F  
}  
// G
```

**Wo sind i, j, k, m, x
sichtbar?**

Ist am Punkt sichtbar?

	A	B	C	D	E	F	G
i							
j							
k							
m							
x							

Warum diese Regeln

- **Lesbarkeit der Programme**
- **Vereinfachung der Verwaltung des Speichers**
 - Platz für eine Variable eines Basistypes muss nur in dem Block organisiert werden, in dem die Variable deklariert ist
 - Werte (die in einer Variable eines Basistypes) gespeichert werden verschwinden am Ende des Blockes

Übersicht

- **2.7 Strings**
 - Nur das wichtigste ...

2.7 Strings

Strings

- **String: Eine Folge von Buchstaben/Zeichen**

- Java Typ String definiert in Standard Bibliothek
- String Variable definiert wie alle anderen Variablen

`String name ;`

- Initialisierung durch String Literal

- Folge zwischen " und " ohne Zeilenende, ggf. mit Ersatzdarstellungen

`String name = "Here ";`

oder Textblock

`String name = """`

`Here`

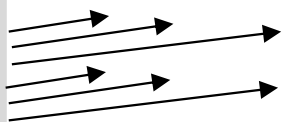
`we go""";`

3 x "

Ende der Zeile

3 x "

Leerzeichen
ignoriert!



Strings

- **String: Eine Folge von Buchstaben/Zeichen**

- Java Typ String definiert in Standard Bibliothek
- String Variable definiert wie alle anderen Variablen

```
String name ;
```

- Initialisierung durch String Literal

- Folge zwischen " und " ohne Zeilenende, ggf. mit Ersatzdarstellungen

```
String name = "Here ";
```

oder Textblock

```
String name = ""  
             Here  
             we go"";
```

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

```
Here  
we go
```

Strings

- **+ erzwingt Konversion von anderen Typen (zu String)**

```
int x = 3;
```

```
String point = "(" + x + ", " + 5 + ");"
```

- **Konversion von anderen Typen (z.B. int) zu String**

```
String s = "" + x;
```

- `""` ist ϵ , der leere String

Strings

- **Weil Strings wichtig sind werden sie vom Compile/Laufzeit-system besonders behandelt**
 - Wie bei `int` und `double` zwingen praktische Überlegungen die Programmiersprache dazu, Details in der Programmierung zu erwarten
- **Standard Bibliothek enthält viele Methoden um Strings zu bearbeiten**
 - Immer vorhanden, ohne `import java.util.*;`
 - Alle Methoden lassen Strings unverändert
 - Strings sind unveränderbar («*immutable*»)

Strings

- **Strings sind Objekte – Methoden mit «dot» Notation**
 - Beispiele: `toUpperCase()`, `toLowerCase()`, ...
- **Können für jeden String seine Länge (Anzahl Zeichen) herausfinden**

```
String str = "Hello";
```

```
System.out.println("Laenge: " + str.length() );
```

Output:

Laenge: 5

- **Strings erlauben Zugriff auf die Buchstaben die den Text ausmachen.**

Teile eines Strings

- Auf Teile eines Strings wird mit einem *Index* zugegriffen

- Basis 0

```
String name = "B. Dylan";
```

index	0	1	2	3	4	5	6	7
Zeichen	B	.		D	y	l	a	n

- Index des ersten Buchstabens: 0
 - Index des letzten Buchstabens: 1 weniger als die Länge des Strings

```
name.length() == 8
```

- **Strings sind keine Arrays!**

- Arrays werden in Teil 3 behandelt (Fragen bitte zurückhalten)

Strings

- **Zugriff auf Elemente eines Strings erfolgt mit (vordefinierten) Methoden**
 - Aufruf dieser Methoden in Punktnotation («dot notation»)
`String s = "hello";`
`s.method(parameterValues);`
 - Führe Methode *method* für *s* aus, «wende *method* auf *s* an», «rufe *method* für *s* auf»
 - Keine Änderung von *s* !
- **Ergebnis kann sein String, int, boolean oder ein Zeichen (Buchstabe)**

String Methoden die String liefern

Method name	Description
substring(index1 , index2) or substring(index1)	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (<u>exclusive</u>); if <i>index2</i> is omitted, grabs till end of string
toLowerCase()	a new string with all lowercase letters
toUpperCase()	a new string with all uppercase letters
stripLeading()	a new string whose value is this string, with all leading white space removed.
stripTrailing()	a new string whose value is this string, with all trailing white space removed.

- «white space» -- Leerzeichen (blank, space), Tabulatorzeichen, LineFeed/CarriageReturn/Return/Enter/Zeilenumbruch ...

String Methoden die String liefern

Method name	Description
substring(index1 , index2) or substring(index1)	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive); if <i>index2</i> is omitted, grabs till end of string

■ Beispiel

index	0	1	2	3	4	5	6	7	8	9
Zeichen	S	.		B	e	c	k	e	t	t

```
String writer = "S. Beckett";           //Laenge: 10
System.out.println(writer.substring(8)); // tt
System.out.println(writer.substring(0,1)); // S
System.out.println(writer.substring(3,7)); // Beck
String w = writer.substring(1,2);        // wie w = "."
```

String Methoden die int liefern

Method name	Description
<code>indexOf(str)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>length()</code>	number of characters in this string
<code>indexOf(str, fromIndex)</code>	index within this string of the first occurrence of the specified substring, starting at the specified index (-1 if not found)

■ Beispiel

index	0	1	2	3	4	5	6	7	8	9
Zeichen	S	.		B	e	c	k	e	t	t

```
String writer = "S. Beckett";           //Laenge: 10
System.out.println(writer.indexOf("Beck")); // 3
System.out.println(writer.indexOf("e"));    // 4
System.out.println(writer.indexOf("e",5));  // 7
```

String weitere Beispiele

```
// index    0123456789012
String s1 = "Alice Munro";
String s2 = "Doris Lessing";

System.out.println(s1.length());           // 11
System.out.println(s1.indexOf("e"));       // 4
System.out.println(s2.substring(6, 9));    // Les

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());     // oris l
```

■ Mit diesem String

```
// index          1           2           3
                  012345678901234567890123456789012
String vorlesung = "Einfuehrung in die Programmierung";
```

Wie würden Sie das Wort "die" extrahieren ?

String weitere Beispiele

```
// index    0123456789012
String s1 = "Alice Munro";
String s2 = "Doris Lessing";

System.out.println(s1.length());           // 11
System.out.println(s1.indexOf("e"));       // 4
System.out.println(s2.substring(6, 9));    // Les

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());     // oris l
```

■ Mit diesem String

```
// index          1         2         3
                  012345678901234567890123456789012
String vorlesung = "Einfuehrung in die Programmierung";
```

Wie würden Sie das Wort "die" extrahieren ?

```
vorlesung.indexOf("die");           // 15
vorlesung.substring(15, 18);
```

String weitere Beispiele

```
// index    0123456789012
String s1 = "Alice Munro";
String s2 = "Doris Lessing";

System.out.println(s1.length());           // 11
System.out.println(s1.indexOf("e"));       // 4
System.out.println(s2.substring(6, 9));    // Les

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());     // oris l
```

■ Mit diesem String

```
// index          1         2         3
                  012345678901234567890123456789012
String vorlesung = "Einfuehrung in die Programmierung";
```

Wie würden Sie das Wort "die" extrahieren ?

```
int loc = vorlesung.indexOf("die");        // 15
vorlesung.substring(loc, loc+3);
```

String Vergleiche/Abfragen

Method	Description
<code>equals(str)</code>	ob 2 Strings die selben Buchstaben enthalten
<code>equalsIgnoreCase(str)</code>	ob 2 Strings die selben Buchstaben enthalten, ohne Berücksichtigung von Gross- und Kleinschreibung
<code>startsWith(str)</code>	ob der String mit den Buchstaben des anderen (str) anfängt
<code>endsWith(str)</code>	ob ... endet
<code>contains(str)</code>	ob der String str (irgendwo) auftritt

```
String s = "Hello";
String t = s.toUpperCase();
if (s.equals(t)) { System.out.println("Equal")); }
else {System.out.println("Not equal")); }    //Not equal
if ("Hello".equals(s)) { System.out.println("Equal")); }
else {System.out.println("Not equal")); }    //Equal
```

Elemente eines Strings

- Die einzelnen Buchstaben sind Werte des (Basistyps) char (später mehr)

```
String name = "B. Dylan";
```

```
name.charAt(0)    // B
```

index	0	1	2	3	4	5	6	7
Zeichen	B	.		D	y	l	a	n

```
if (name.charAt(1) == '.') { ... } // Paar single quote ' '
```

```
char c = name.charAt(7);
```

```
System.out.println(name.indexOf('.')); // 1
```

- Verwenden Sie == nur für Basistypen (z.B. int oder char), nicht für String Variable
 - Später sehen wir wann wir == verwenden können

Zur Erinnerung

- **Zuweisungen zu String Variablen ändern nicht den String**
 - Die Variable verweist auf einen anderen String

Animation

Zur Erinnerung

- **Zuweisungen zu String Variablen ändern nicht den String**

- Die Variable verweist auf einen anderen String

- **Methoden (z.B. substring) liefern neuen String**

- Keine Modifikation des String für den sie aufgerufen wurden

```
String s = "Hello World";
```

```
s.toUpperCase();
```

```
System.out.println(s);    // Hello World
```

- **Ergebnis kann in Variable gespeichert werden**

```
String s = "Hello World";
```

```
String t = s.toUpperCase();
```

```
System.out.println(t);    // HELLO WORLD
```

Zur Erinnerung

- **Zuweisungen zu String Variablen ändern nicht den String**
 - Die Variable verweist auf einen anderen String

- **Methoden (z.B. substring) liefern neuen String**

- Keine Modifikation des String für den sie aufgerufen wurden

```
String s = "Hello World";
```

```
s.toUpperCase();
```

```
System.out.println(s);    // Hello World
```

- **Ergebnis kann in Variable gespeichert werden**

```
String s = "Hello World";
```

```
s = s.toUpperCase();    // kann selbe Variable sein
```

```
System.out.println(s);    // HELLO WORLD
```

String als Parameter

```
public class StringParameters {  
    public static void main(String[] args) {  
        String friend = "Mark";  
        sayHello(friend);  
  
        sayHello("Peter");  
    }  
  
    public static void sayHello(String name) {  
        System.out.println("Welcome, " + name);  
    }  
}
```

Output:

```
Welcome, Mark  
Welcome, Peter
```

String als Parameter

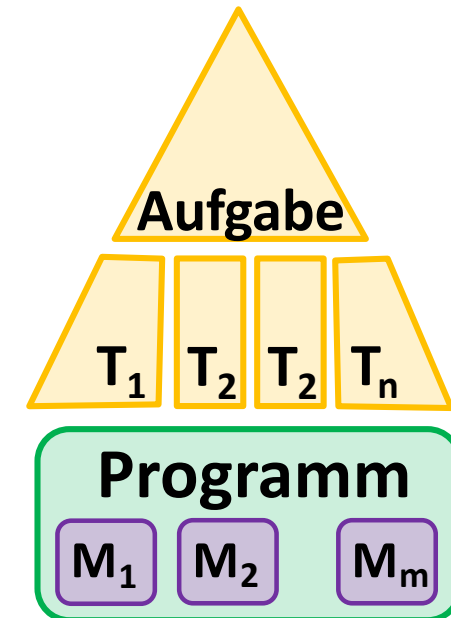
```
public class StringParameters {  
    public static void main(String[] args)  
        String friend = "Mark";  
        sayHello(friend);  
        sayHello(friend);  
}  
  
public static void sayHello(String name) {  
    System.out.println("Welcome, " + name);  
    name = "Incognito";  
}  
}
```

Output:

```
Welcome, Mark  
Welcome, Mark
```

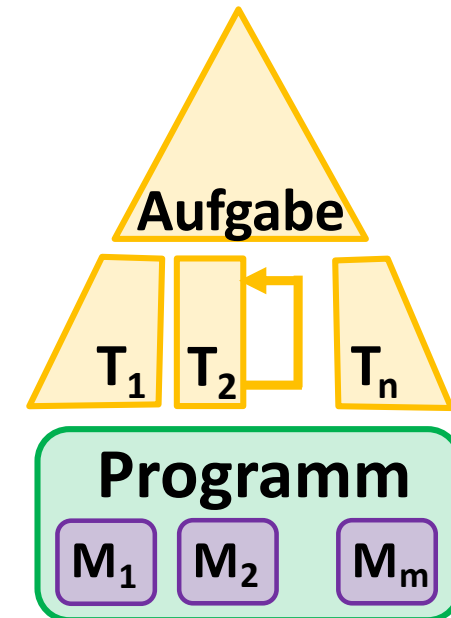
Zerlegen in Teilaufgaben

- Ziel: ... so dass Teilaufgaben T_i wiederverwendet werden können
 - Genauer: die Anweisungen für T_i können wiederverwendet werden
- Anweisungen für T_i : Methode M_i
 - Evtl. Hilfsmethoden M_j



Zerlegen in Teilaufgaben

- **Anweisungen für T_i : Methode M_i**
 - Evtl. Hilfsmethoden M_j
- **Methoden können Schleifen enthalten**
 - Beliebige Anweisungen
- **Methoden können in Schleifen aufgerufen werden**



Aufgabe

- **Schreiben Sie eine Methode `oneCount(String)` die berichtet wie oft das Zeichen 1 im Eingabe-String auftritt.**
 - Wenn Sie wollen können Sie sich vorstellen, dass die Eingabe-Strings Zahlen in Binärdarstellung sind
- **Beispiele**
 - `oneCount("001") → 1`
 - `oneCount("1010") → 2`

Aufgabe

- **Schreiben Sie eine Methode `oneCount(String)` die berichtet wie oft das Zeichen 1 im Eingabe-String auftritt.**
 - Wenn Sie wollen können Sie sich vorstellen, dass die Eingabe-Strings Zahlen in Binärdarstellung sind
- **Fragen**
 - Typ des Rückgabewertes: `int`
 - Sonderfälle
 - `oneCount("abc") → 0`
 - `oneCount("") → 0`

Aufgabe

- **Was für ein Parameter? String s**
- **Wie kann oneCount jedes Zeichen analysieren?**
 - **Loop**
 - Ein Zeichen? `substring(index, index+1)`
 - Prüfen ob Zeichen eine 1 ist? `"1".equals(s.substring(..))`
- **Wie wird Ergebnis berechnet?**
 - `int` Variable, um 1 erhöhen wenn Zeichen eine 1 ist

Lösung 1

```
public static int oneCount (String s) {  
    int result = 0;  
    for (int i=0; i<s.length(); i = i + 1) {  
        if ("1".equals(s.substring(i, i+1))) {  
            result = result + 1;  
        }  
    }  
    return result;  
}
```

Anderer Ansatz

- **Was für Anweisungen können im Rumpf einer Methode `method()` auftreten?**

- Alle.
- Auch Aufrufe von Methoden.
- Auch Aufrufe der Methode `method()`.

index	0	1	2
Zeichen	1	0	1

- **Wie wird Ergebnis berechnet?**

- `oneCount(s.substring(0,1)) + oneCount(s.substring(1))`
- `(0 oder 1) + oneCount(Rest_des_Strings)`

Zerlegen einer (Teil)Aufgabe (Methode M_i)

- **Teilaufgabe T_1 : für Input X sofort lösbar**

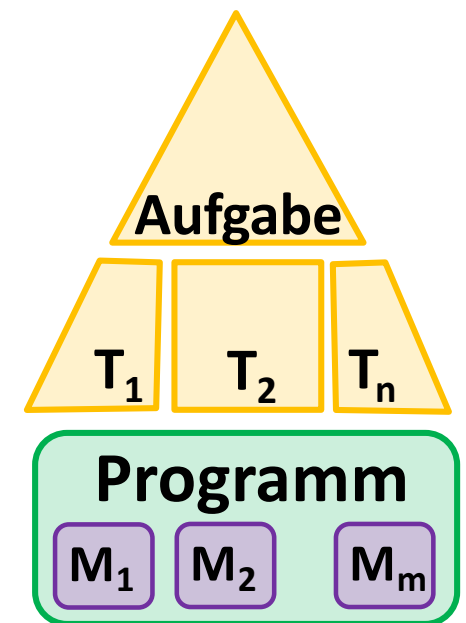
- Beispiel: X ein String der Länge 1

$\text{oneCount}(X) \rightarrow 1$ wenn X String "1" ist, sonst 0

- **Teilaufgabe T_2 :**

- Zerlege Input in zwei Teile X_1 und X_2
- Ergebnis kann (leicht) aus $M_i(X_1)$ und $M_i(X_2)$ berechnet werden
- Beispiel: X länger als 1 Zeichen

$\text{oneCount}(\text{Erstes_Zeichen}) + \text{oneCount}(\text{Rest_des_Strings})$



index	0	1	2
Zeichen	1	0	1

`oneCount(s.substring(0,1)) + oneCount(s.substring(1))`

index	0
Zeichen	1

index	0	1
Zeichen	0	1

`oneCount(s.substring(0,1)) + oneCount(s.substring(1))`

index	0
Zeichen	0

index	0
Zeichen	1

`oneCount(s.substring(0,1)) + oneCount("")`

index	0
Zeichen	1

index	
Zeichen	

Lösung 2

```
public static int oneCount (String s) {  
    int result = 0;  
    if (s.length()==0) return 0;  
    if (s.substring(0,1).equals("1")) {  
        result = 1;  
    }  
    return result + oneCount(s.substring(1));  
}
```

Lösung 2a

```
public static int oneCount (String s) {  
    if (s.length()==0) return 0;  
    if (s.length()==1 && "1".equals(s)) { return 1; }  
    else if (s.length()==1 && !("1".equals(s))) {  
        return 0;  
    } else {  
        return oneCount(s.substring(0,1)) +  
               oneCount(s.substring(1));  
    }  
}
```


Lösung 3

```
public static int oneCount (String s) {  
    if (s.length()==0) return 0;  
    if (s.length()==1) {  
        return ("1".equals(s) ? 1 : 0);  
    } else {  
        return oneCount(s.substring(0,1)) +  
            oneCount(s.substring(1));  
    }  
}
```

Lösung 4

```
public static int oneCount (String s) {  
    if (s.length()==0) return 0;  
    if (s.length()==1) {  
        return ("1".equals(s) ? 1 : 0);  
    } else {  
        return oneCount(s.substring(0,s.length()/2)) +  
            oneCount(s.substring(s.length()/2+1));  
    }  
}
```

Übersicht

- **2.8 Nochmals Schleifen**
 - 2.8.1 Kurzformen (für Aktualisierung)
 - 2.8.2 Kurzformen und bedingte («short-circuit») Ausführung
 - 2.8.3 Terminierung von Schleifen
 - 2.8.4 Input Werte zur Schleifenkontrolle
 - 2.8.5 Invarianten

2.8 Nochmals Schleifen

- Kurzform zur Aktualisierung des Loop Counters (Schleifenzählers)
- Tipps für korrekte Terminierung der Schleifen
- Hoare Tripel für Schleifen

2.8.1 Aktualisierung

```
for (int i = start ; i < bound; i = i + 1) {  
    // Statement  
}
```

Aktualisierung: `i` wird um 1 erhöht

```
for (int i = start ; i > bound; i = i - 1) {  
    // Statement  
}
```

Aktualisierung: `i` wird um 1 reduziert

Auch andere Aktualisierungen sind möglich aber diese hier treten häufig auf

Kurzformen für Zuweisungen

- **Zuweisungen der Form $j = j+1$ treten häufig auf**
 - Machen Programm unübersichtlich
 - Früher: unnötige Extra-Arbeit für Compiler und Computer
- **Kurzformen erlauben Inkrement (Addition von 1) und Dekrement (Subtraktion von 1)**
 - «increment» und «decrement» Operator
 - Veränderung immer um 1

Inkrement und Dekrement

Kurzform

variable++;

variable--;

Äquivalente ausführlichere Version

variable = variable + 1; //increment

variable = variable - 1; //decrement

Beispiele

int x = 2;

x++;

// x = x + 1;

// x now stores 3

double note = 4.5;

note--;

// note = note - 1;

// note now stores 3.5

Aktualisierung

```
for (int i = start ; i < bound; i++) {  
    // Statement  
}
```

Aktualisierung: `i` wird um 1 erhöht

```
for (int i = start ; i > bound; i--) {  
    // Statement  
}
```

Aktualisierung: `i` wird um 1 reduziert

`++` (und `--`) oft in Aktualisierungen des Loop Counters

Inkrement und Dekrement

Kurzform

variable++;

variable--;

Äquivalente ausführlichere Version

variable = variable + 1;

variable = variable - 1;

Variable wird *verwendet* und dann *verändert*

Dies gilt auch in Ausdrücken

Inkrement und Dekrement

Kurzform

variable++;

variable--;

Äquivalente ausführlichere Version

variable = variable + 1; //increment

variable = variable - 1; //decrement

Beispiele

```
int x = 2;
```

```
System.out.println(x++); // x = x + 1; x now stores 3
```

```
System.out.println(x++); // x = x + 1; x now stores 4
```

Output:

2

3

Inkrement und Dekrement

Kurzform

`variable++;`

`variable--;`

Äquivalente ausführlichere Version

`variable = variable + 1;`

`variable = variable - 1;`

Variable wird *verwendet* und dann *verändert*

Dies gilt auch in Ausdrücken und Zuweisungen

Beispiel

```
int x = 2;
```

```
int y;
```

```
y = x++;
```

Inkrement und Dekrement

Kurzform

`variable++;`

`variable--;`

Äquivalente ausführlichere Version

`variable = variable + 1;`

`variable = variable - 1;`

Variable wird verwendet und dann verändert

Dies gilt auch in Ausdrücken und Zuweisungen

Beispiel

`int x = 2;`

`int y;`

`y = x++;`

`int temp = x;`

`x++;`

`y = temp;`

//x:

//y:

Inkrement und Dekrement

Kurzform

`variable++;`

`variable--;`

Äquivalente ausführlichere Version

`variable = variable + 1;`

`variable = variable - 1;`

Variable wird verwendet und dann verändert

Dies gilt auch in Ausdrücken und Zuweisungen

Beispiel

`int x = 2;`

`int y;`

`y = x++;`

`int temp = x;`

`x = x + 1;`

`y = temp;`

//x:

//y:

Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. int, long, oder double)

RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird berechnet
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiele LHS: int k

```
int i = 3;
```

```
int j = 7;
```

RHS

Resultat:

9

9

3+5

8

i+2

5

i++

3

// i: 4

j-- + j%4

9

// j: 6

Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. int, long, oder double)

RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird berechnet
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiel

```
int i = 3;
```

```
int j = i++;
```

Achtung: Zwei Variablen werden verändert!

Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. `int`,
`long`, oder `double`)

RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird *berechnet*
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiel

```
int i = 3;  
int j = i++;
```


Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. int, long, oder double)

RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird *berechnet*
 1. RHS: 3
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiel

```
int i = 3;  
int j = i++;  
      = 3; //update i!!
```

Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. `int`,
`long`, oder `double`)

RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird *berechnet*
 1. RHS: 3
 2. Addiere 1 zu Variable `i`
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiel

```
int i = 3;  
int j = i++;  
        = 3; //update i!!  
           // i+1
```

Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. `int`,
`long`, oder `double`)


RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird *berechnet*
 1. RHS: 3
 2. Addiere 1 zu Variable `i`
 3. *Speichere Variable `i`*
2. Resultat (Wert) wird in Variable (LHS) gespeichert

Beispiel

```
int i = 3;  
int j = i++;  
        = 3; //update i!!  
           // i+1  
           // i=
```



Zuweisungen (Assignment Statement)

LHS = RHS;

LHS: Eine Basistyp Variable (z.B. int, long, oder double)

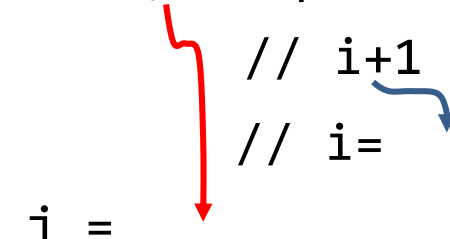
RHS: Ein Ausdruck

Ablauf:

1. Rechte Seite (RHS) wird berechnet
 1. RHS: 3
 2. Addiere 1 zu Variable i
 3. Speichere Variable i
2. *Resultat* (Wert) wird in Variable (LHS) *gespeichert*

Beispiel

```
int i = 3;
int j = i++;
        = 3; //update i!!
           // i+1
           // i=
j =
```



Inkrement **und** Dekrement **Puzzles**

- Unser Ziel ist es *verständliche* Programme zu schreiben

Inkrement und Dekrement Puzzles

- Unser Ziel ist es *verständliche* Programme zu schreiben
- ... und nicht Puzzles zu konstruieren!
- Sie sollten ++ und -- (er)kennen
 - Auch in komplexen Ausdrücken
 - Ihre Entscheidung ob Sie es verwenden (aber wenn dann richtig)
- Diese Operatoren sind *nicht* so effizient dass wir dafür die Klarheit eines Programmes opfern wollen.

Weitere Kurzformen

- Erlauben Verwendung des Wertes einer Variable gefolgt von einer Modifikation (Zuweisung)

Kurzform

variable += value;

variable -= value;

variable *= value;

variable /= value;

variable %= value;

Äquivalente ausführlichere Version

variable = variable + value;

variable = variable - value;

variable = variable * value;

variable = variable / value;

variable = variable % value;

- Modifikation mit beliebigen Werten (nicht nur 1)

Weitere Kurzformen

Beispiele

<code>x += 3;</code>	<code>// x = x + 3;</code>
<code>note -= 0.5;</code>	<code>// note = note - 0.5;</code>
<code>number *= 2;</code>	<code>// number = number * 2;</code>

Warnung:

<code>x += 1;</code>	<code>// x = x + 1;</code>
<code>x =+ 1;</code>	<code>// x = + 1;</code>

Weitere Kurzformen – manchmal nützlich

- `x++` und `j--` heissen Post-Increment bzw. Post-Decrement Operator, da die Veränderung (von `x` und `j`) gemacht wird *nachdem* der Wert (von `x` oder `j`) gelesen («gebraucht») wurde.
- Es gibt auch Operatoren, die die Veränderung (Increment oder Decrement) durchführen *bevor* der Wert gelesen wurde; dies sind der Pre-Increment bzw. Pre-Decrement Operator: `++j` oder `--x`.

Beispiele

```
int x = 2;
```

```
System.out.println(++x); // x = x + 1; x now stores 3
```

```
System.out.println(++x); // x = x + 1; x now stores 4
```

Output:

3

4

Weitere Kurzformen – manchmal unnötig

- `x++` und `j--` heissen Post-Increment bzw. Post-Decrement Operator, da die Veränderung (von `x` und `j`) gemacht wird *nachdem* der Wert (von `x` oder `j`) gelesen («gebraucht») wurde.
- Es gibt auch Operatoren, die die Veränderung (Increment oder Decrement) durchführen *bevor* der Wert gelesen wurde; dies sind der Pre-Increment bzw. Pre-Decrement Operator: `++j` oder `--x`.

Beispiele

```
int x = 2;
```

```
System.out.println(++x); // x = x + 1; x now stores 3
```

```
System.out.println(++x); // x = x + 1; x now stores 4
```

Output:

3

4

2.8.2 Bedingte Auswertung und Kurzformen

- Für `&&` und `||` müssen nicht immer beide Operanden ausgewertet werden, um das Ergebnis zu ermitteln
- Java *beendet* die Auswertung eines booleschen Ausdrucks sobald das Ergebnis fest steht.
 - `&&` und `||` sind links-assoziativ
 - Ausdrücke werden von links nach rechts, gemäss Präzedenz und Assoziativität ausgewertet
 - `&&` stoppt sobald ein Teil(ausdruck) `false` ist
 - `||` stoppt sobald ein Teil(ausdruck) `true` ist

Bedingte Auswertung: Vorsicht

- Was ist der Wert von count am Ende des Codesegments?

```
// look closely
int count = 0;
Scanner console = new Scanner(System.in);
for (int i = 0; i<4; i++) {
    System.out.print("Eingabe Zahl: ");
    int wert = console.nextInt();

    if ((wert != 0) && (count++ < 9)) {
        System.out.println("Hit");
    }
} // count: Anzahl Werte ungleich 0, nicht Iterationen
```

- Vorsicht bei ++/--

Bedingte Auswertung: Vorsicht

- **Die logischen Operatoren sind nicht kommutativ wenn die Auswertung den Zustand des Programms verändern kann.**
 - `(expr1 && expr2)` nicht immer gleich `(expr2 && expr1)`
- **Vorsicht bei Operatoren mit Nebenwirkungen («side effects»)**
 - Offensichtliche Nebenwirkungen: z.B. `int x,y; x++ y--` o. ä.
 - Nicht sofort offensichtlich:
 - Methoden oder Funktionen, die Zustand des Programms ändern (werden wir später kennenlernen)
 - Operationen die Zustand des Systems ändern (wie z.B. `x/0` – Fehler!)

Kurzformen - Recap

- **Unser Ziel ist es, verständliche Programme zu schreiben.**

- Vorsicht bei Kurzformen und bedingter Auswertung
 - Oft sinnvoll um kompakt Laufzeitfehler zu vermeiden
- Gebrauch erlaubt, nicht erzwungen

- **Was wird gedruckt?**

```
int x = 2;  
System.out.println(++x + x++ + " " + x + ++x + x);
```

2.8.3 Terminierung von Schleifen

Eine triviale Aufgabe ...

- Schreiben Sie eine Methode `printNumbers` die die Zahlen von 1 bis N durch Komma getrennt ausgibt.

Beispiel:

Obergrenze N eingeben: 5

sollte ergeben:

1, 2, 3, 4, 5

Lösungsansatz

```
public static void printNumbers() {  
    Scanner console = new Scanner(System.in);  
    System.out.print("Obergrenze N eingeben: ");  
    int max = console.nextInt();  
  
}
```

Welche Schleifen liefern gewünschten Output?

Poll

```
public static void printNumbers() {  
    Scanner console = new Scanner(System.in);  
    System.out.print("Obergrenze N eingeben: ");  
    int max = console.nextInt();
```

```
// Option A  
for (int i = 1; i <= max; i++) {  
    System.out.print(i + ", ");  
}  
System.out.println(); // to end the line of output
```

```
// Option B  
for (int i = 1; i <= max; i++) {  
    System.out.print(", " + i);  
}  
System.out.println(); // to end the line of output
```

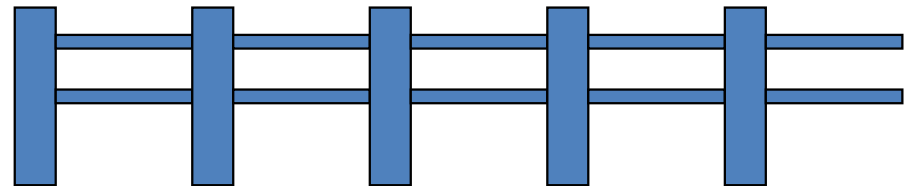
```
}
```

```
}// printNumbers
```

Gartenzaun Analogie

- Wir geben n Zahlen aus aber brauchen nur $n - 1$ Kommas.
- Ähnlich dem Bau eines Weidezaunes mit Pfosten und Querstreben
 - Wenn wir – wie in der 1. fehlerhaften Lösung – Pfosten und Streben installieren dann hat der letzte Pfosten in der Luft hängende Streben.

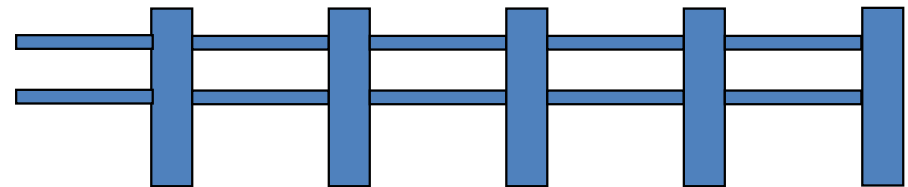
```
for (Länge des Zauns) {  
    Betoniere Pfosten.  
    Installiere Querstreben.  
}
```



Gartenzaun Analogie

- Wir geben n Zahlen aus aber brauchen nur $n - 1$ Kommas.
- Ähnlich dem Bau eines Weidezaunes mit Pfosten und Querstreben
 - Wenn wir – wie in der 2. fehlerhaften Lösung – Streben und Pfosten installieren dann hat der erste Pfosten in der Luft hängende Streben.

```
for (Länge des Zauns) {  
    Installiere Querstreben.  
    Betoniere Pfosten.  
}
```



Schleife

- Fügen Sie eine Anweisung ausserhalb der Schleife hinzu um den ersten «Pfosten» zu platzieren

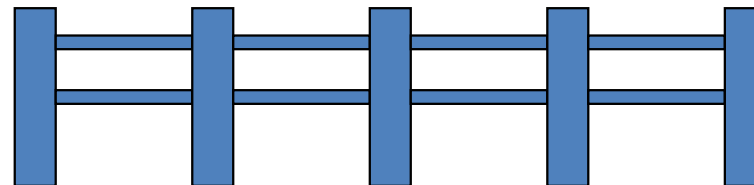
Betoniere Pfosten.

for (Länge des Zauns - 1) {

Installiere Querstreben.

Betoniere Pfosten.

}



Lösungen basierend auf dieser Idee

```
System.out.print(1);  
for (int i = 2; i <= max; i++) {  
    System.out.print(", " + i);  
}  
System.out.println();    // to end the line
```

Alternative: 1. oder letzter Durchlauf durch die Schleife kann verändert werden:

```
for (int i = 1; i <= max - 1; i++) {  
    System.out.print(i + ", ");  
}  
System.out.println(max);    // to end the line
```

Lösung (eine Möglichkeit)

```
public static void printNumbers() {  
    Scanner console = new Scanner(System.in);  
    System.out.print("Obergrenze N eingeben: ");  
    int max = console.nextInt();  
  
    System.out.print(1);  
    for (int i = 2; i <= max; i++) {  
        System.out.print(", " + i);  
    }  
    System.out.println(); // to end the line  
}
```

«off-by-one» Error (*Um-Eins-Daneben-Fehler*)

- Die Schleife wurde einmal zuviel (oder einmal zuwenig) durchlaufen.
- «Zaunpfahlproblem» – es gibt sogar eine D Wikipedia Seite (Inhalt ohne Gewähr)

Terminierung von Loops

- Verwandeln Sie die Methode `printNumbers` in eine neue Methode `printPrimes` die alle *Primzahlen* (durch Komma getrennt) bis zur Obergrenze *max* ausgibt ($max \geq 2$).
 - Beispiel: `printPrimes` mit Eingabe 50 ergibt:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
- Eine Primzahl p kann in genau zwei Faktoren zerlegt werden: p und 1

```

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

```

```

public static void printPrimes(int limit) {
    // Prints all prime numbers from 2 up to the given limit
    // limit >= 2
    System.out.print("2");

    for (int candidate = 3; candidate <= limit; candidate++) {
        // Determine if candidate is prime
        // Count factors!  2: prime, >2 not prime
        int count = 0;
        for (int j = 1; j<=candidate; j++) {
            if (candidate % j == 0) {
                count++;
            }
        }
        if (count == 2) {
            System.out.print(", " + candidate);
        }
    }
    System.out.println(); // to end output
}

```

2.8.4 Input Werte zur Schleifen Kontrolle

- **Interessantes Beispiel eines unbestimmten Loops**
 - Kandidat für while-Schleife
- **Wert wird nicht (nur) zur Berechnung verwendet sondern kontrolliert auch den Loop (d.h. die Terminierung)**
 - Wert ist (zusätzlich) Hinweis

Werte die Hinweise sind ...

- **Hinweiszeichen (Sentinel) («sentinel»):** Ein Wert der das Ende eine Reihe anzeigt
 - *sentinel loop*: Schleife deren Rumpf ausgeführt wird bis ein Sentinel gesehen wurde
- **Beispiel: Ein Programm soll Zahlen einlesen bis der Benutzer eine 0 eingibt; dann soll die Summe aller eingegebenen Zahlen ausgegeben werden.**
 - (In diesem Beispiel ist 0 das Hinweiszeichen/der Sentinel.)

Werte die Hinweise sind ...

- **Beispiel: Ein Programm soll Zahlen einlesen bis der Benutzer eine 0 eingibt; dann soll die Summe aller eingegebenen Zahlen ausgegeben werden.**
 - (In diesem Beispiel ist 0 das Hinweiszeichen/der Sentinel)

```
Enter a number (0 to quit): 10  
Enter a number (0 to quit): 20  
Enter a number (0 to quit): 30  
Enter a number (0 to quit): 0  
The sum is 60
```

Fehlerhafte Lösung

- Was ist an diesem Programm schlecht?

```
Scanner console = new Scanner(System.in);
int sum = 0;
int number = 1;    // "dummy value", anything but 0

while (number != 0) {
    System.out.print("Enter a number (0 to quit): ");
    number = console.nextInt();
    sum = sum + number;
}
System.out.println("The total is " + sum);
```

Ein anderes Hinweiszeichen ...

- **Ändern Sie das Programm so dass -1 der Sentinel ist.**

```
Scanner console = new Scanner(System.in);
int sum = 0;
int number = 1;    // "dummy value", anything but 0

while (number != -1) {
    System.out.print("Enter a number (0 to quit): ");
    number = console.nextInt();
    sum = sum + number;
}
System.out.println("The total is " + sum);
```


Ein anderes Hinweiszeichen ...

- Ändern Sie das Programm so dass -1 der Sentinel ist.
 - Example log of execution:

```
Enter a number (-1 to quit): 15  
Enter a number (-1 to quit): 25  
Enter a number (-1 to quit): 10  
Enter a number (-1 to quit): 30  
Enter a number (-1 to quit): -1  
The total is 79
```

Ein anderes Hinweiszeichen ...

- **Setzen Sie den Sentinel auf -1:**

```
Scanner console = new Scanner(System.in);
int sum = 0;
int number = 1;    // "dummy value", anything but -1

while (number != -1) {
    System.out.print("Enter a number (-1 to quit): ");
    number = console.nextInt();
    sum = sum + number;
}
System.out.println("The total is " + sum);
```

- **Jetzt ist das Result falsch. Warum?**

The total is 79

Fehlerhafte Lösung – 0 → -1

■ Was ist an diesem Programm falsch?

```
Scanner console = new Scanner(System.in);
int sum = 0;
int number = 1;    // "dummy value", anything but -0

while (number != -0) {
    System.out.print("Enter a number (-0 to quit): ");
    number = console.nextInt();
    sum = sum + number;
}
System.out.println("The total is " + sum);
```

Das Problem mit diesem Programm

- **Unser Programm folgt diesem Muster:**

```
summe = 0
while (input ist nicht der sentinel) {
    drucke prompt; lese input
    addiere input zu summe
}
```

- **Beim letzten Durchlauf durch den Rumpf wird der Sentinel -1 zur Summe addiert:**

Das Problem mit diesem Programm

- **Beim letzten Durchlauf durch den Rumpf wird der Sentinel -1 zur Summe addiert:**
 - drucke prompt; lese input (-1)
 - addiere input (-1) zu summe
- **Beispiel inkorrektter Terminierung (off-by-one error, Zaunpfahlproblem):**
 - Müssen N Zahlen lesen aber nur die ersten $N-1$ addieren.

Lösung

summe = 0

drucke prompt; lese input

// setzen eines pfostens

while (input ist nicht der sentinel) {

addiere input zu summe

// installation querstrebe

drucke prompt; lese input

// setzen eines pfostens

}

- **Schleifen mit einem Sentinel folgen oft diesem Muster.**

Beispiel mit Sentinel

```
Scanner console = new Scanner(System.in);
int sum = 0;

// pull one prompt/read ("post") out of the loop
System.out.print("Enter a number (-1 to quit): ");
int number = console.nextInt();

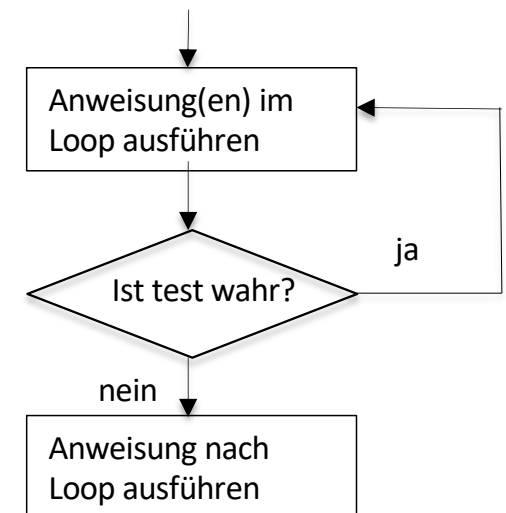
while (number != -1) {
    sum = sum + number;    // moved to top of loop
    System.out.print("Enter a number (-1 to quit): ");
    number = console.nextInt();
}

System.out.println("The total is " + sum);
```

do-while-Schleife

- **do-while-Schleife:** Führt test am Ende des Schleifenrumpfes aus um zu entscheiden, ob ein weiterer Durchlauf nötig ist
 - Stellt sicher dass der Rumpf { ... } mindestens einmal ausgeführt wird.

```
do {  
    statement(s);  
} while (test);  
// naechste Anweisung
```



do-while-Schleife

■ Beispiel:

// Example: prompt until correct PIN is typed

```
int input;
```

```
do {
```

```
    System.out.print("Type your PIN: ");
```

```
    input = console.nextInt();
```

```
} while (input != userPinCode);
```

