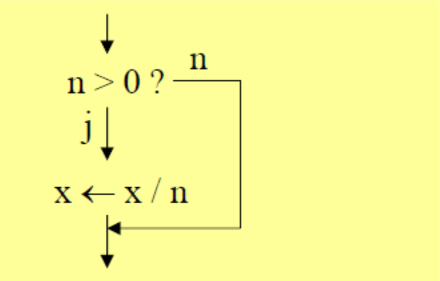
Conditions & Loops

Agenda

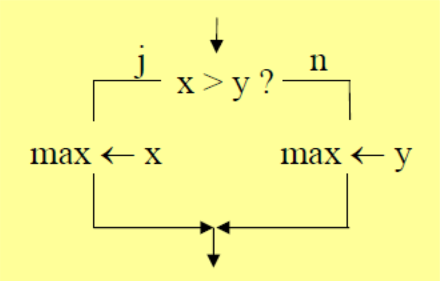
- Conditions
  - If – Else, Switch
- Loops
  - While, Do-While, For

If-Else



```
if (n > 0) x = x / n;
```

without else



```
if (x > y)
   max = x;
else
   max = y;
```

with else

Syntax

IfStatement = "**if**" "(" Expression ")" Statement ["**else**" Statement].

Blocks

If there is more than one statement in the if or the else part of a condition, we need to define blocks with {…}.

```
Statement = Assignment | IfStatement | Block | ... .
Block = "{" {Statement} "}".
```

Blocks

- Example

```
if (x < 0) {
 negNumbers++;
 System.out.print(-x);
} else {
 posNumbers++;
 System.out.print(x);
}
```

Indentation

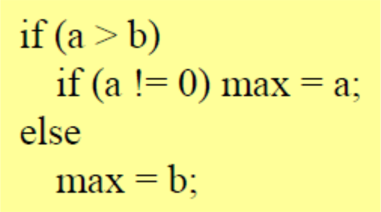
Best Practice:
{...} for single statements too

Indentations

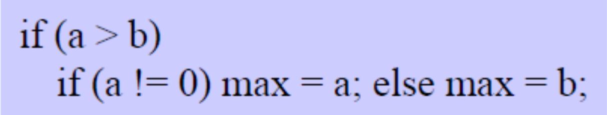
- For readability
  - visualize strucutre
- how much?
  - 1 tab oder 2 spaces

- Short If-statements in a single line:
  - `if (n != 0) x = x / n;`
  - `if (x > y) max = x; else max = y;`

Dangling Else



```
if (a > b)
   if (a != 0) max = a;
else
   max = b;
```



```
if (a > b)
   if (a != 0) max = a; else max = b;
```

- Two ifs, one else. Where does the else belong to?
- In Java else goes with the if immediately before it.

- Alternative: use blocks.

Short If

- (Expression)?Statement:Statement

```
int x = 3;
int y = 4;
int max = (x<y)?y:x;

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

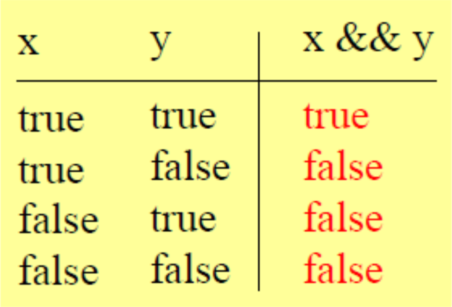
Comparison

- Compare two values
- Returns *true* or *false*

		Example
==	equal	x==3
!=	not equal	x!=y
>	larger than	4>3
<	smaller than	x+1<0
>=	larger or equal	x>=y
<=	smaller or equal	x<=y

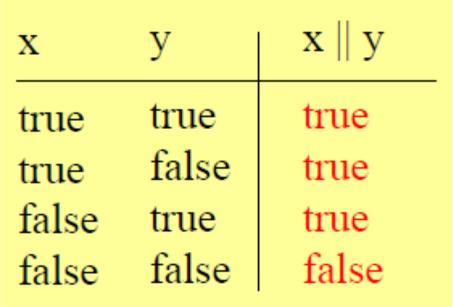
Combining Comparisons

&& logic AND



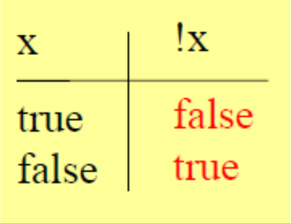
x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

|| logic OR



x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

! logic NOT



x	!x
true	false
false	true

- Example
  - `if (a >= 0 && a <= 10 || a >= 100 && a <= 110) b = a;`

Boolean Operators

- `!` Is stronger && and ||
- && is stronger than||

- brackets for association of clauses
  - `if (a > 0 && (b==1 || b==7)) ...`

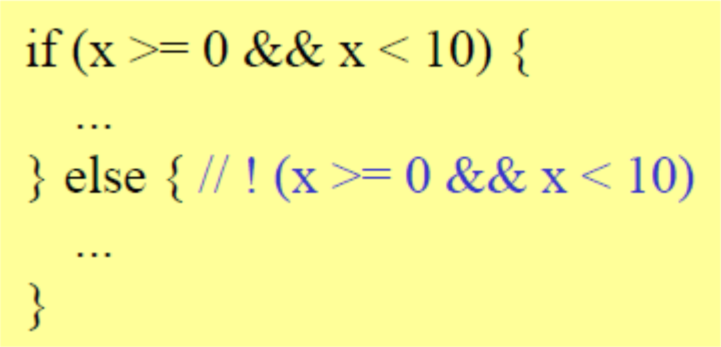
Data Type `boolean`

- data type (just like `int`)
    - values are *true* and *false*
- Examples

```
boolean p, q;
p = false;
q = x > 0;
p = p || q && x < 10;
```

DeMorgan Rules

- ! (a && b) ⇔ ! a || ! b
- ! (a || b) ⇔ ! a && ! b



```
if (x >= 0 && x < 10) {
    ...
} else { // ! (x >= 0 && x < 10)
    ...
}
```

⇨ ! (x >= 0) || ! (x < 10)

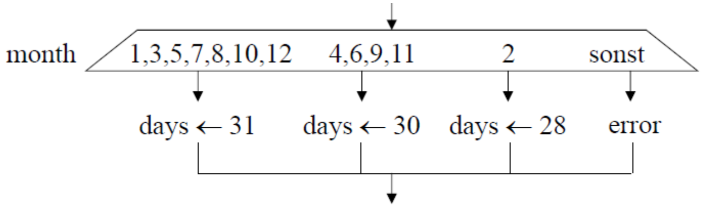
⇨ x < 0 || x >= 10

Examples `boolean` & `if`

- Expression is evaluated to `true` or `false`
  - `if (true) …`
  - `if (!true) …`
  - `if ((x >=1) == true) …`

Switch Statement

- Multiple branches



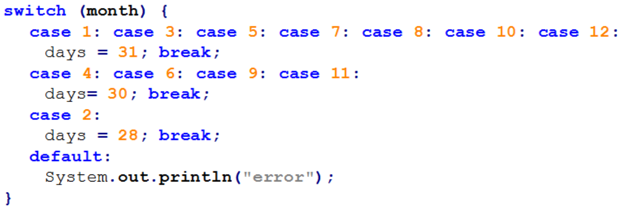
- In Java

```
switch (month) {
 case 1: case 3: case 5: case 7: case 8: case 10: case 12:
  days = 31; break;
 case 4: case 6: case 9: case 11:
  days= 30; break;
 case 2:
  days = 28; break;
 default:
  System.out.println("error");
}
```

Switch Statement

- Conditions
  - expression has to be integer, char or String
  - case labels have to be constants
  - case label data has to fit expression
  - case labels need to pair wise different
- Break statement
  - Jumps to the end of the switch block
  - If break is missing, everything after it is executed
    - typical error

Switch Expression



```
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
     days = 31; break;
   case 4: case 6: case 9: case 11:
     days= 30; break;
   case 2:
     days = 28; break;
   default:
     System.out.println("error");
}
```

Switch-Syntax

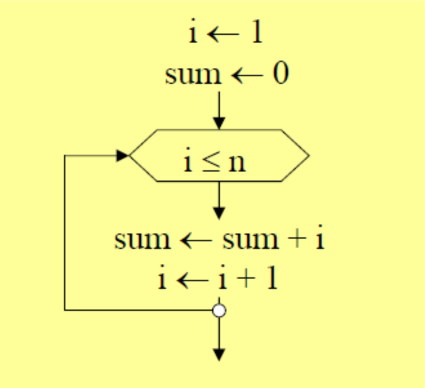
```
Statement = Assignment | IfStatement | SwitchStatement | ... | Block.
SwitchStatement = "switch" "(" Expression ")" "{" {LabelSeq StatementSeq} "}".
LabelSeq = Label {Label}.
StatementSeq = Statement {Statement}.
Label = "case" ConstantExpression ":" | "default" ":".
```

While Loop

- Loops a sequence of statements
- As long as a condition evaluates to `true`.

Statemen

WhileSta



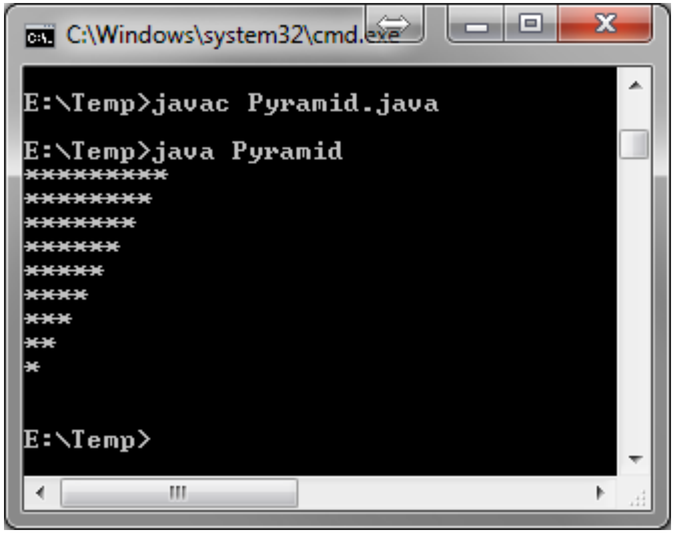
```
i = 1;
sum = 0;
while ( i <= n ) {
    sum = sum + i;
    i = i + 1;
}
```

continuation condition

loop body

While Loop

```
class Pyramid {
  public static void main (String[] arg) {
        int i = 10;
        while (i-->0) {
              int j = 0;
              while (j++<i) {
                    System.out.print("*");
              }
              System.out.println();
        }
  }
}
```



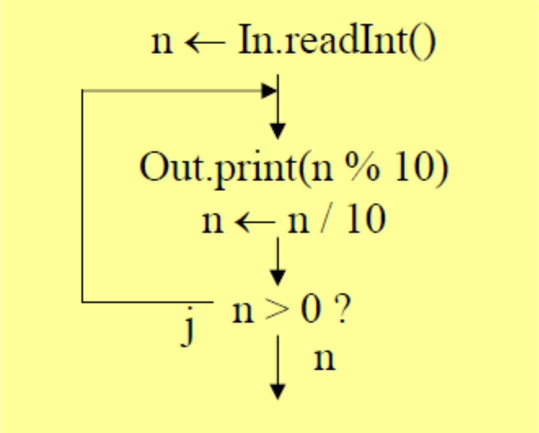
```
E:\Temp>javac Pyramid.java

E:\Temp>java Pyramid
**********
*********
********
*******
******
*****
****
***
**
*


E:\Temp>
```

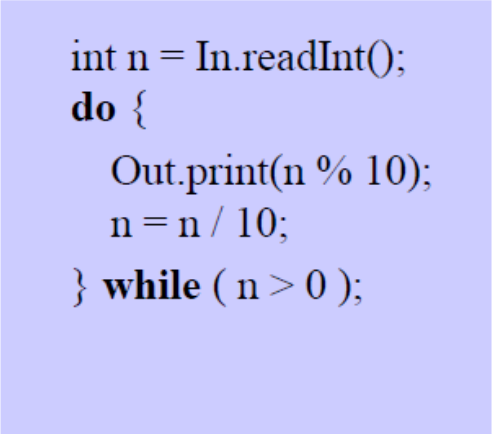
Termination

- Loops should terminate
  - no endless loop `while (true) { ... }`
- Common problems for endless loops
  - variable in continuation condition is not changed
  - continuation condition never evaluates to `false`
    - eg. `while (x!=0) { x -= 5; }`
- Approach: model & test for typical problems

Do-While Loop

- Continuation condition is tested at the end of the loop
- Loop body is run at least once





```
int n = In.readInt();
do {
   Out.print(n % 10);
   n = n / 10;
} while ( n > 0 );
```

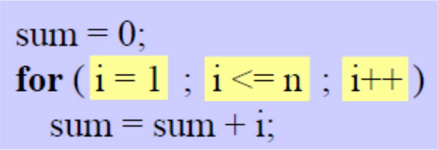
proof of concept

n	n % 10
~~123~~	3
~~12~~	2
~~1~~	1
0	

```
St                                                                       ent
  | ... | Block.
DoWhileStatement = "do" Statement "while" "(" Expression ")" ";".
```

For Loop (Counting Loop)

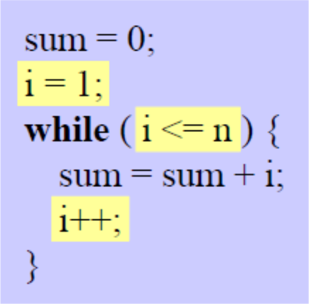
- Used if number of iterations is known beforehand



```
sum = 0;
for ( i = 1  ;  i <= n  ;  i++ )
   sum = sum + i;
```

1) Initialisation
2) Continuation condition
3) Update

.. is actually short for



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

For Loop Examples

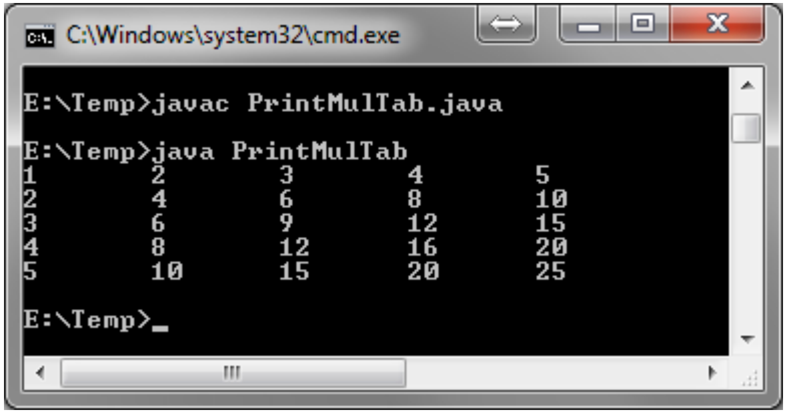
for (i = 0; i < n; i++)	i: 0, 1, 2, 3, …, n-1
for (i = 10; i > 0; i--)	i: 10, 9, 8, 7, …, 1
for (int i = 0; i <= n; i = i + 1)	i: 0, 1, 2, 3, …, n
for (int i = 0, j = 0; i < n && j < m; i = i + 1, j = j + 2)	i: 0, 1, 2, 3, … j: 0, 2, 4, 6, …
for (;;) …	Endless loop

For Loop Definition

```
ForStatement = "for" "(" [ForInit] ";" [Expression] ";"
 [ForUpdate] ")" Statement.

ForInit = Assignment {"," Assignment} | Type VarDecl {","
 VarDecl}.

ForUpdate = Assignment {"," Assignment}.
```

For Loop Example

```
class PrintMulTab {
 public static void main (String[] arg) {
    int n = 5;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            System.out.print(i * j + "\t");
        }
        System.out.println();
    }
 }
}
```



```
E:\Temp>javac PrintMulTab.java

E:\Temp>java PrintMulTab
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25

E:\Temp>_
```

Termination of Loops

- Terminate with keyword *break*

```
while (In.done()) {
  sum = sum + x;
  if (sum > 1000) {
    Out.println(”too big");
    break;
  }
  x = In.nextNumber();
}
```

- However, it’s better to use the continuation condition

```
while (In.done() && sum < 1000) {
  sum = sum + x;
  x = In.nextNumber();
}
if (sum > 1000)
   System.out.print("too big");
```

Termination of Outer Loops

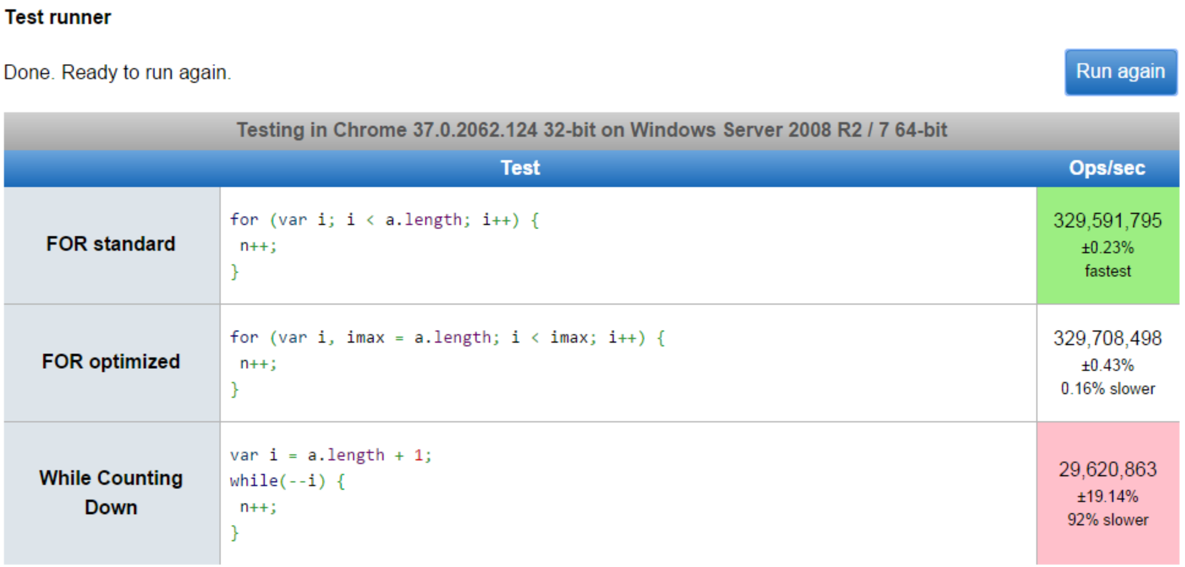
```
outer: // Label!
for (;;) { // endless loop!
 for (;;) {
     ...
     if (...) break;     // terminates inner loop
     else break outer;   // terminates outer loop
     ...
 }
}
```

Loop Termination

- When to use `break`
    - on errors (performance!)
    - multiple exit points within a loops
    - real endless loops (eg. in real time systems)

Which Type of Loop When?

- Selection based on “Convenience”
  - counting, condition at begin or end ..
- Selection based on performance
  - (s.u. für Javascript, http://jsperf.com/fun-with-for-loops/8)



Test runner

Done. Ready to run again.

Testing in Chrome 37.0.2062.124 32-bit on Windows Server 2008 R2 / 7 64-bit

Test		Ops/sec

FOR standard	`for (var i; i < a.length; i++) {`
	` n++;`
	`}`
		329,591,795
		±0.23%
		fastest

FOR optimized	`for (var i, imax = a.length; i < imax; i++) {`
	` n++;`
	`}`
		329,708,498
		±0.43%
		0.16% slower

While Counting Down	`var i = a.length + 1;`
	`while(--i) {`
	` n++;`
	`}`
		29,620,863
		±19.14%
		92% slower