Introductory Programming
Imperative Programming II, sections 3.1-3.5

Anne Haxthausen[a]

IMM, DTU

1. Control flow (section 3.1)
2. Conditional statements (**if** and **switch**) (sections 3.2 and 3.3)
3. Block statements (section 3.2)
4. Boolean expressions (sections 3.2, 3.4)
5. More operators (read on your own: section 3.5)

a. Parts of this material are inspired by/originate from a course at ITU developed by Niels Hallenberg and Peter Sestoft on the basis of a course at KVL developed by Morten Larsen and Peter Sestoft.

Control flow of programs

control flow = order in which statements are executed

- Unless otherwise specified, a program is executed *linearly*:
  - Java executes the statements of the `main` method one by one
- The execution order can be controlled by two special kinds of statements:
  - *conditionals* (valg-sætninger) give the choice between two or more statements: **if**, **if-else** and **switch**.
  - loops (løkker/gentagelses-sætninger) repeat a statement: **while**, **do** and **for**.

if statements

if statements are useful in situations where you can say:

"if ... then ..."

Example:

```
double toptax = 0.0;

if (income > 267000)
   toptax = (income-267000) * 0.15;
```


if-else statements

if-else statements are useful in situations where you can say:

"if ... then ... otherwise ..."

Example:

```
double toptax;

if (income > 267000)
   toptax = (income-267000) * 0.15;
else toptax = 0.0;
```

if-else statements

Syntax:

if (*expression*)
  statement1
else
  statement2

Semantics:

(1) the value of *expression* is evaluated

(2) if it is **true**, *statement1* is executed, otherwise *statement2* is executed

Blocks

A sequence of statements can be grouped into one statement, a so-called *block*, by enclosing the statements by `{ ... }`.

This can for instance be useful in if-else statements where only a single statement is allowed in each of the branches.

Example:

```
if (income > 267000)
   toptax = (income - 267000) * 0.15;
else
{
  toptax = 0.0;
  System.out.println("Better luck " +
                     "at next salary negotiations");
}
```


The else problem

What is the result of the following, if X is 4:

```
if (x==3)
   if (y < 10) System.out.println("branch A");
else System.out.println("branch B");
```

Rule: else is matched to the closest unmatched **if**.

Use braces (tuborg-klammer) to specify which **if** an **else** belongs to, e.g.:

```
if (x==3)
   { if (y < 10) System.out.println("branch A"); }
else
   {System.out.println("branch B");}
```

What is the result now?

It is better to use extra braces than making logical errors!

International grading scale

Score in %	Grade
95–100	A
90–94	A-
87–89	B+
83–86	B
80–82	B-
67–69	D+
⋮	⋮
0–59	F

Exercise: Write a program that converts a score to a grade.

Solution using nested if-else

```
int    score; // 0-100
String grade; // A, A-, ..., D-, F

if (score >= 95)
   grade = "A";
else
{
   if (score >= 90)
      grade = "A-";
   else
   {
      if (score >= 87)
         grade = "B+";
      else // etc.
         ...
   }
}
```


switch statements

switch is useful, when you need more than two branches.

Conversion of grades using switch

```
int    score; // 0-100
String grade; // A, A-, ..., D-, F

switch (score)
{
    case 100: case 99: case 98: case 97: case 96:  case 95:
       grade = "A";
       break;
    case 94: case 93: case 92: case 91: case 90:
       grade = "A-";
       break;
    // etc.
    default: // 0 - 59
       grade = "F";
}
```


Boolean expressions

We have previously looked at examples of expressions that denote

- numerical values (of type **int, double, ...**)
- strings of characters (of type `String`)

A *Boolean expression* denotes a truth value (of type **boolean**). Boolean expressions include:

- **true** and **false**
- relational operators (sammenligningsoperatorer) applied to expressions of the same type (e.g. `income > 267000`)
- Boolean operators applied to Boolean expressions (e.g. `!found && (x < 10)`)

Relational operators and their precedence

Operator	Meaning	Example
<	less than	x < 60
<=	less than or equal to	x <= 60
>	greater than	x > 60
>=	greater than or equal to	x >= 60
==	equal to	x == 60
!=	not equal to	x != 60

The result type is **boolean**, i.e. the result is **true** or **false**.

The arithmetic operators *, /, %, +, - "bind tighter" than <, <=, >, >=.

The relational operators <, <=, >, >= "bind tighter" than == and !=.

Example: what are the values of the following Boolean expressions?

Assume x = 2 and y = 4.

Boolean expression	Value
false	
true	
true == false	
x != y	
x < 3 + y	
y < x + 3	
(x + y > 3) == false	
false != x < 3	
x == y == false	

Comparing chars

Like numbers, two characters can be compared:

Example:

```
char ch1= 'a', ch2 = 'c';

if (ch1 < ch2)
   System.out.println(ch1 +
                      " is less than " +
                      ch2);
else
   System.out.println(ch1 +
                      " is greater than or equal to" +
                      ch2);
```

gives:

```
a is less than c
```


Comparing strings

Strings can be compared by the methods `equals` and `compareTo`:

Example:

```
String name1="Henrik", name2="Anne";

System.out.println("Are the names equal? " +
                   name1.equals(name2));

if (name1.compareTo(name2) < 0)
   System.out.println("name1 comes before name2");
else if (name1.compareTo(name2) > 0)
        System.out.println("name2 comes before name1");
```

gives

```
Are the names equal? false
name2 comes before name1
```

Comparing floating point numbers

Two floating point numbers are equal, if *all* the bits of their internal representations are the same.

Often it is the case that it is sufficient to check whether two numbers are close to each other:

```
double x=3.00002, y=3.00001;
double v=3.02, w=3.01;
final double TOLERANCE = 1E-3;

if (Math.abs(x-y) < TOLERANCE)
   System.out.println("x and y are essentially equal");

if (Math.abs(v-w) > TOLERANCE)
   System.out.println("but v and w are not!");
```

gives

```
x and y are essentially equal
but v and w are not!
```


Boolean operators and their precedence

Operator	Meaning	Example
!	Not	`!(x == 60)`
&&	And	`0 <= x && x < 60`
||	Or	`x < 0 || x >= 60`

The types of the operands must be **boolean**. The result type is **boolean**.

The operator `!` binds tighter than && which binds tighter than `||`.

`!` binds tighter than the relational operators and the arithmetic operators.

&& and `||` binds less tight than the relational operators and the arithmetic operators.

Truth tables for the Boolean operators:

a	! a
false	true
true	false

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

The Boolean expressions are evaluated from left to right.

The operators && and `||` evaluate no more than necessary:

- If *expression1* is false in *expression1* && *expression2* then *expression2* is not evaluated.
- If *expression1* is true in *expression1* `||` *expression2* then *expression2* is not evaluated.

Example: what are the values of the following Boolean expressions?

Assume x = 2 and y = 4.

Boolean expression	**Value**
`! false`	
`! true`	
`! true == false`	
`! (true == false)`	
`true && false`	
`false || true`	
`x + y > 3 && x < y`	
`x + y == 3 || x < 4`	
`x < y && (3*4 == 2*6-1*2+2) == !(3<x)`	

More operators

Assume given **int** `count = 0;`

```
count++;
```

is equivalent to

```
count = count + 1;
```

Read more about this in section 3.5.

Advice: increase the readability – write the full expressions!

Prepare for next week

- Read sections 3.6-3.8, 3.0, 3.9.
- Prepare the exercises.