Objekt-Orienteret Programmering

Control Flow

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellemann pmn@mmmi.sdu.dk

September 11, 2023

SDU

Part 0:
Boolean Typen

Boolean Typen ▷ Definition

Type der anvendes til at beskrive udsagn som værende sande eller falske.

Boolean værdier: {`true`, `false`}

Alternativt: {1, 0} eller {åben, lukket} eller {aktiv, inaktiv} eller ...

Der findes en primitiv type i java til at beskrive booleans: `boolean` :-)

Matematik	Java
$a = b$	`a == b`
$a \neq b$	`a != b`
$a > b$	`a > b`
$a < b$	`a < b`
$a \geq b$	`a >= b`
$a \leq b$	`a <= b`

Boolean Typen ▷ Boolske Expressions

Boolske expressions er expressions der evaluerer til enten `true` eller `false`.

Eksempler:

```
boolean lastLevel    = (level == levelCount);
boolean modernTime   = (year > 0);
boolean newBestScore = score >= bestScore;
```

Navn	**Eksempel**	**Sandt når**
NOT	`!value`	et udtryk ikke er sandt
AND	`value1 && value2`	begge udtryk er sande
OR	`value1 || value2`	mindst ét udtryk er sandt
XOR	`value1 ^ value2`	netop ét udtryk er sandt

Boolean Typen ▷ Sandhedstabel for NOT Operatoren

"modsat af værdien"

a	!a
0	1
1	0

"sand hvis og kun hvis begge sider er sande"

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Typen ▷ Sandhedstabel for OR Operatoren

"sand hvis mindst et af udtrykkene er sande"

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

"sand hvis netop ét udtryk er sandt (exclusive or)"

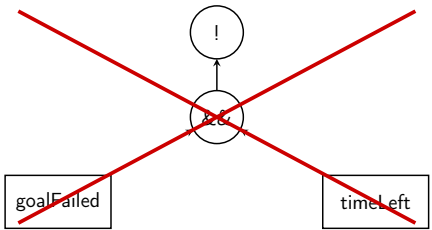
a	b	a && b
0	0	0
0	1	1
1	0	1
1	1	0

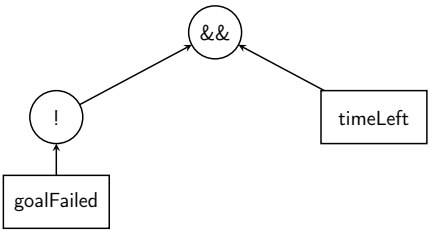
Boolean Typen ▷ Eksempel

```
boolean goalFailed = false;
boolean timeLeft   = false;
boolean success    = !goalFailed && timeLeft;
```

Boolean Typen ▷ Eksempel

```
boolean goalFailed = false;
boolean timeLeft   = false;
boolean success    = !goalFailed && timeLeft;
```





Hvorfor?

Operator precedens!

Part 1:
Valg i Expressions

Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

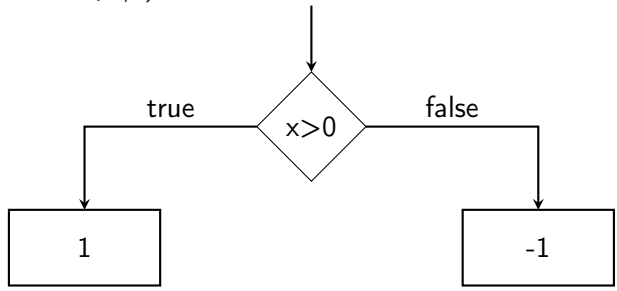
Syntaks:
(⟨*condition*⟩ ? ⟨*true − expr*⟩ : ⟨*false − expr*⟩)

Eksempel:

```
int y = (x>0 ? 1 : -1);
```

Hvad er værdien af y når

1. x ↦ 12?
2. x ↦ 0?
3. x ↦ true?



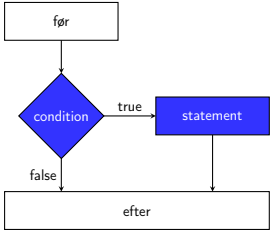
Part 2:
Valg i Statements

Valg i Statements ▷ If og Else

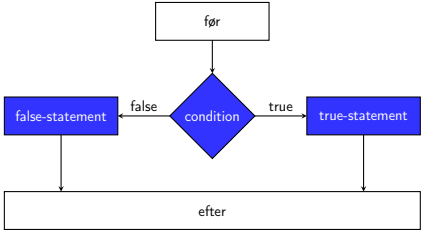
Ud fra evalueringen af et boolsk expression kan et program – på kørselstidspunktet – bestemme hvorvidt et statement skal udføres.

Syntaks:

if (⟨*condition*⟩)
    ⟨*statement*⟩



if (⟨*condition*⟩)
    ⟨*true − statement*⟩
else
    ⟨*false − statement*⟩



Valg i Statements ▷ Branching Eksempel

Eksempel:

```
System.out.println("Input is "+i);
if (i%2==1)
  i++;
System.out.println("Output is "+i);
```

Tolkes: Læg én til `i` hvis `i` modulo `2` er `1` (altså, hvis `i` er et ulige tal).

Bemærk: Branches (her ved hjælp af `if`) indfører valg i udførelsen af et program.

Part 3:
Gruppering af Statements

Gruppering af Statements ▷ Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Syntaks:

{
  ⟨*statement* − *sekvens*⟩
}

Husk: Statements adskildt af semi-kolon'er er en sekvens af statements.

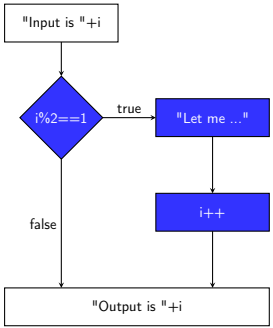
Det er meget anvendeligt, for nu kan vi erstatte vores enkelt-linje statements med en block, og dermed flere statements!

Note: Blocks implementerer konceptet om "sekvenser" i vores programmer.

Gruppering af Statements ▷ Branching med Blocks (1/2)

Eksempel:

```
System.out.println("Input is "+i);
if (i%2==1) {
  System.out.println("Let me adjust your value ...");
  i++;
}
System.out.println("Output is "+i);
```



Gruppering af Statements ▷ Branching med Blocks (2/2)

Eksempel:

```
double radius = 12.0;
if (radius < 0.0) {
  System.out.println("Incorrect input");
} else {
  area = 3.14159 * radius * radius;
  System.out.println("Area is " + area);
}
```

Gruppering af Statements ▷ "Dangling Else" Problemet (1/2)

```
int i = 1, j = 2, k = 3;
if (i < j)
  if (i > k)
    System.out.println("A");
else
  System.out.println("B");
```

Hvad bliver der printet?

Der bliver printet "B" da ...

Gruppering af Statements ▷ "Dangling Else" Problemet (2/2)

Java tolker koden sådan her:

```
int i = 1, j = 2, k = 3;
if (i < j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

Ud over at muliggøre sekvenser, løser blocks også andre problemer; herunder "dangling else"-problemet.

Derfor: **Brug altid blocks, også ved inkelt-linje statements!**

Part 4:
Multiple Valgmuligheder i Statements

Multiple Valgmuligheder i Statements ▷ Else-If Chained Branches

```
char direction = 'n';
if (direction=='N') {
  System.out.println("Going north ...");
} else if (direction=='S') {
  System.out.println("Going south ...");
} else if (direction=='E') {
  System.out.println("Going east ...");
} else if (direction=='W') {
  System.out.println("Going west ...");
} else {
  System.out.println("I don't understand ?!?");
}
```

Multiple Valgmuligheder i Statements ▷ Else-If Chained Branches

```
char direction = 'N';
if (direction=='N') {
  System.out.println("Going north ...");
} else if (direction=='S') {
  System.out.println("Going south ...");
} else if (direction=='E') {
  System.out.println("Going east ...");
} else if (direction=='W') {
  System.out.println("Going west ...");
} else {
  System.out.println("I don't understand ?!?");
}
```

Multiple Valgmuligheder i Statements ▷ Switch Statements

switch (⟨*expr*⟩) {
case ⟨*expression*⟩ :
⟨*statement* − *sekvens*⟩ ;
break;
case ⟨*expression*⟩ :
⟨*statement* − *sekvens*⟩ ;
break;
case ⟨*expression*⟩ :
⟨*statement* − *sekvens*⟩ ;
break;
default :
⟨*statement* − *sekvens*⟩ ;
}

Notes:

- Hvis ikke **break** anvendes, fortsættes til næste case.
- **default** er valgfri og "fanger" alt der ikke rammer en case.

Multiple Valgmuligheder i Statements ▷ Switch Eksempel

```
char direction = 'S';
switch (direction) {
case 'N':
  System.out.println("Going north ...");
  break;
case 'S':
  System.out.println("Going south ...");
  break;
case 'E':
  System.out.println("Going east ...");
  break;
case 'W':
  System.out.println("Going west ...");
  break;
default:
  System.out.println("I don't understand ?!?");
}
```

Part 5:
Gentagelse af Statements

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 2 3 gange:

```
System.out.println("Hello World!");
System.out.println("Hello World!");
System.out.println("Hello World!");
```

Hvad hvis vi vil printe det `n` gange?

Så bruger vi et *loop*!

Idag skal vi lære om tre typer loops:

1. while
2. do-while
3. for

Part 5.1:
Loops med “while” Statements

Loops med “while” Statements ▷ Introduktion

Det mest basale type af programloop i Java.

Afvikler et *statement* så længe en *condition* er sand.

Typisk anvendt henover en *block*, hvor en sekvens afvikles så længe en *condition* er sand.

Loops med "while" Statements ▷ Syntaks

Syntaks:
while (⟨*condition*⟩) ⟨*statement*⟩

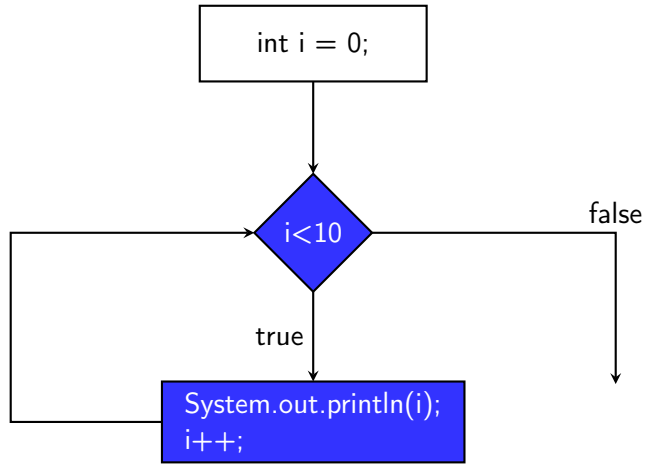
Loopets *"continuation condition"* testes i starten af hvert gennemløb.

Hvis denne *condition* er sand, afvikles *statement*; ellers ikke.

Statementet afvikles altså $0 - \infty$ gange.

Loops med “while” Statements ▷ Eksempel

```
int i = 0;
while (i<10) {
  System.out.println(i);
  i++;
}
```



Loops med “while” Statements ▷ Øvelse

```
int i = 0;
while (i < 4) {
  i++;
  System.out.println("i is now " + i);
}
```

Efter afvikling af denne kode har `i` værdien `4`.

Spørgsmål: Hvad står der i den sidste linje som programmet udskriver?

Part 5.2:
Loops med “do-while” Statements

Loops med “do-while” Statements ▷ Introduktion

Som et “while”-loop, men evaluerer continuation condition **efter** statement.

- Konsekvens: Statementet afvikles altid én gang indledningsvist.

Anvendes som while oftest til at modifcere en variable i statement så længe continuation condition er sand; dog mindst én gang.

Loops med "do-while" Statements ▷ Syntaks

Syntaks:
do ⟨*statement*⟩ *while* (⟨*condition*⟩)

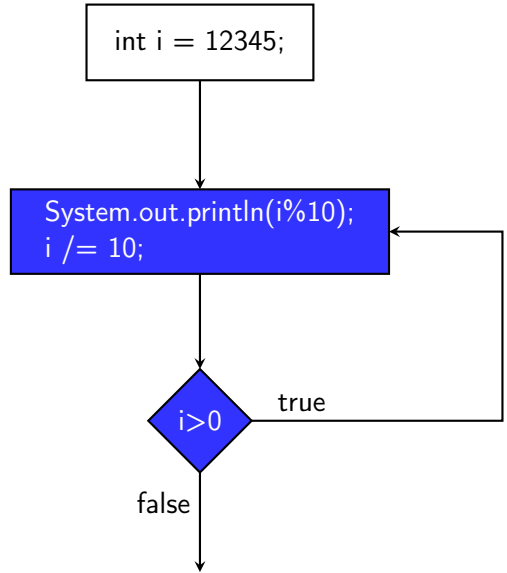
Continuation condition testes første gang efter én gennemkørsel.

Statement afvikles herefter indtil continuation condition er falsk.

Statementet afvikles altså $1 - \infty$ gange.

Loops med “do-while” Statements ▷ Eksempel

```
int i = 12345;
do {
  System.out.println(i%10);
  i /= 10;
} while (i>0);
```



Gentagelse af Statements ▷ While som Erstatning for Do-While

```
do {
  <statement>
} while ( <condition> );
```

```
<statement>
while ( <condition> ) {
  <statement>
}
```

Spørgsmål: Hvilken variant er mest elegant, og hvorfor?

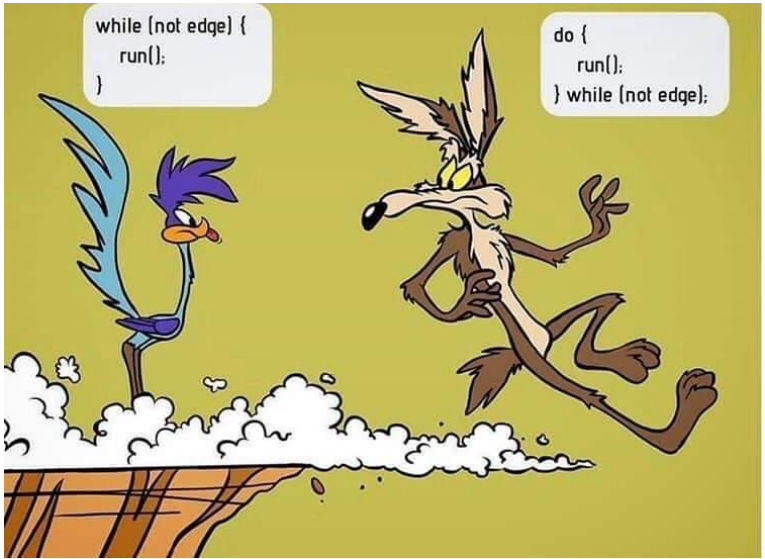
Gentagelse af Statements ▷ Do-While som Erstatning for While

```
while ( <condition> ) {
  <statement>
}
```

```
if ( <condition> ) {
  do {
    <statement>
  } while ( <condition> );
}
```

Spørgsmål: Hvilken variant er mest elegant, og hvorfor?

Gentagelse af Statements ▷ Do-While vs While



Part 5.3:
Loops med “for” Statements

Loops med "for" Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – "en opskrift" – som ofte ses i programmer.

Mønsteret består af:

- En loop **body** som indeholder de statements der skal loopes over.
- Et **initialization statement** der afvikles én gang i begyndelsen.
- En **continuation condition** der bestemmer hvorvidt et loop skal fortsættes.
- Et **update statement** der afvikles i slutningen af hvert gennemløb af loopet.

Loops med "for" Statements ▷ Syntaks

Syntaks:
for (⟨*init* − *statement*⟩ ; ⟨*condition*⟩ ; ⟨*update* − *statement*⟩) ⟨*body* − *statement*⟩

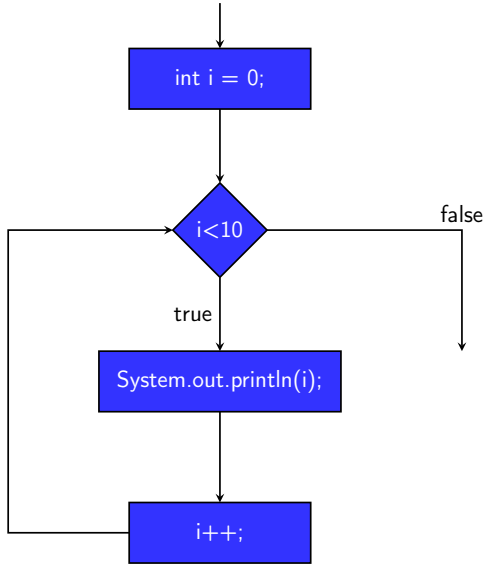
Continuation condition testes i starten af hvert gennemløb. Kroppen afvikles $0 - \infty$ gange.

"Syntaktisk sukker" for:

```
<init-statement>
while (condition) {
  <body-statement>
  <update-statement>
}
```

Loops med “for” Statements ▷ Eksempel

```
for (int i=0 ; i<10 ; i++) {
  System.out.println(i);
}
```



Loops med “for” Statements ▷ For vs While

```
for (years=0 ; years<5 ; years++) {
  interest = principal * rate;
  principal += interest;
  System.out.println(principal);
}
```

```
years = 0; // init
while (years<5) { // condition
  interest = principal * rate;
  principal += interest;
  System.out.println(principal);

  years++; // update
}
```

Loops med "for" Statements ▷ Flere Veje til Rom . . .

Spørgsmål:
Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

```
for (int i=1 ; i<=10 ; i++) {
  System.out.println(2*i);
}
```

```
for (int i=2 ; i<=20 ; i = i+2) {
  System.out.println(i);
}
```

```
for (int i=2 ; i<=20 ; i++) {
  if (i%2==0) // is i even?
  {
    System.out.println(i);
  }
}
```

```
for (int i=1 ; i<=1 ; i++) {
  System.out.println("2 4 6 8 10 12 14 16 18 20");
}
```

Part 6:
Indlejrede Loops

Indlejrede Loops ▷ Eksempel

Da loops selv er statements, kan vi også *indlejre* (eng: neste) dem.

```
for (int y=1 ; y<=10 ; y++) {
  for (int x=1 ; x<=y ; x++)
  {
    System.out.printf("%4d", x*y);
  }
  System.out.println("");
}
```

Indlejrede Loops ▷ Resultat

Da loops selv er statements, kan vi også *indlejre* (eng: neste) dem.

```
 1
 2   4
 3   6   9
 4   8  12  16
 5  10  15  20  25
 6  12  18  24  30  36
 7  14  21  28  35  42  49
 8  16  24  32  40  48  56  64
 9  18  27  36  45  54  63  72  81
10  20  30  40  50  60  70  80  90 100
```

Part 7:
Flow Control

Flow Control ▷ Break

`break` afslutter loopet og hopper til statement umiddelbart efter loopet.

```
int i;
for (i=0 ; i<10 ; i++) {
  if (i==5)
  {
    break;
  }
}
System.out.println(i);
```

Resultat: 5

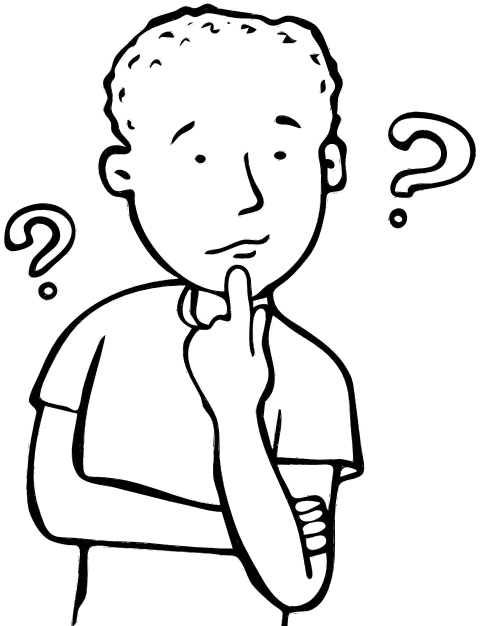
Flow Control ▷ Continue

`continue` springer over den resterende del af body for dette gennemløb.

```
int i;
int sum = 0;
for (i=0 ; i<4 ; i++) {
  if (i==2) {
    continue;
  }
  sum += i;
}
System.out.println(sum);
```

Resultat: 4

Spørgsmål?



https://openclipart.org/detail/236687/boy-thinking-of-question