



Programming Basics

Hoofdstuk 5

De Java programmeertaal

**DE HOGESCHOOL
MET HET NETWERK**

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook



Inhoud

1. Inleiding
2. Variabelen en letterlijke waarden
3. Operatoren
4. Uitdrukkingen, statements en blokken
5. Programmaverloop-statements
6. Methoden

1. Inleiding

In dit hoofdstuk:

- Taalregels van Java (*syntax*)
- Programmeeralgoritmen

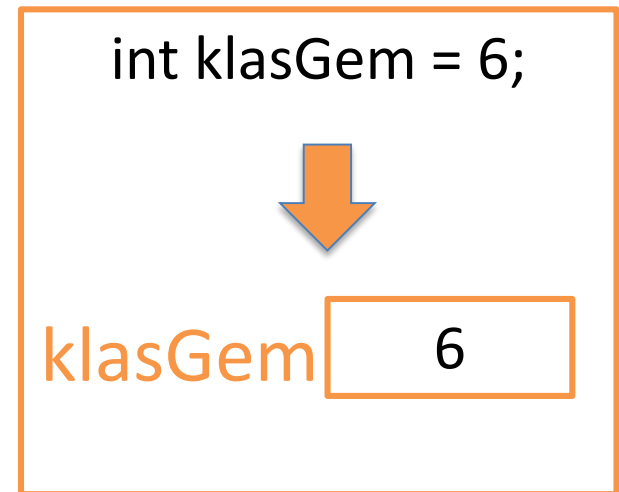


2. Variabelen en letterlijke waarden

Variabele: een geheugenlocatie met een naam waar je een waarde in kan stoppen

Elke variabele heeft een:

- **Type:** Vb: int, double, char, ...
- **Naam** (*identifier*)
 - lowerCamelCasing
 - kies zinvolle namen!vb: sum, klasGem, ...
- **Bereik** (*scope*): variabele kan maar gebruikt worden binnen een bepaald gebied van het programma



2.1 De declaratie van variabelen

Type name [= initialValue]; —————> [] d.w.z. is optioneel

```
int number;  
int number1, number2;  
int result = 10;  
int result1, result2;  
int res1 = 2, res2 = 3; /* niet aanbevolen */  
int quantity1, quantity2 = 7; /* quantity1  
    = unassigned (not initialized) */  
int sum = result + 2;  
float total;  
double salary = 1234.5;  
char letter = 'a';  
boolean geslaagd = true;
```

2.2 Het datatype

2.2.1 Het primitieve datatype

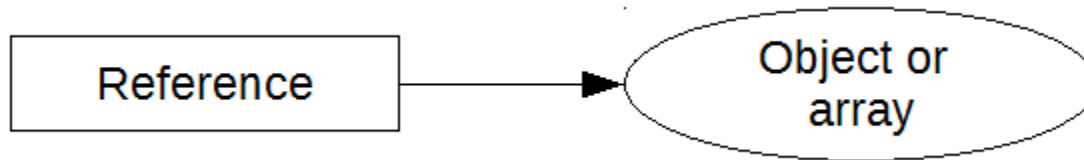
Type	Formaat
Gehele getallen	
byte	8 bit
short	16 bit
int	32 bit
long	64 bit
Reële getallen	
float	32 bit
double	64 bit
Andere types	
boolean	1 bit
char	16 bit

Bevat een enkele waarde. Bv. een getal, een letter of een logische (booleaanse) waarde: waar (true) – niet waar (false)

-2,... miljard → + 2,... miljard

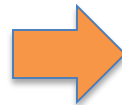
2.2.2 Het referentietype

Is een verwijzing (referentie) naar een object. Bevat het adres van het object.



Vb: `Button button = new Button();`

Creëert een
nieuw object
van de klasse
Button



De variabele
button is een
verwijzing naar
dit object

Een referentievariabele die naar geen enkel
object verwijst heeft de waarde **null**

2.3 Literals

= letterlijke waarden

```
int result = 10;  
double salary = 1234.5;  
char letter = 'a';  
boolean geslaagd = true;  
System.out.println("Hello World");
```


2.4 De naam

- beginnen met letter, \$ of _ (\$ en _ worden afgeraden)
- lowerCamelCasing
- uniek binnen scope
- geen gereserveerd woord (vb if, while, final, ...)

lijst van gereserveerde woorden: zie cursus



2.5 Final variables of constanten

```
final int CONSTANT;  
CONSTANT = 1;
```

of

```
final int CONSTANT = 1;
```

- Waarde van constante kan niet meer gewijzigd worden
- Hoofdletters
- Vermits CamelCasing niet mogelijk is (o.w.v. hoofdletters) gebruik je eventueel een underscore (_)

```
final int MY_CONSTANT = 1;
```



Opdracht 1: *Waarde toekennen aan variabele*

- Maak een programma waarin je variabelen van volgende datatypes een waarde (literal) geeft.
 - *boolean*
 - *char*
 - *byte*
 - *short*
 - *int*
 - *double*

float en long stellen we even uit totdat we typeconversie gezien hebben!



Opdracht 2: *Final variables*

- Voeg *final double PI = 3.14* toe. Tracht de waarde daarna te veranderen door *PI = 3.15* toe te voegen. Welke foutmelding krijg je?



2.6 Typeconversie

= gegevens van een bepaald datatype omzetten naar een ander datatype

- **Impliciete conversie** (automatisch door de compiler)

```
short aShort = 5;  
int anInteger;  
anInteger = aShort;
```

short
(16 bit)

past in!!!

int
(32 bit)



- **Expliciete conversie (casten)**

```
short aShort;  
int anInt = 600000;  
aShort = anInt;
```

int
(32 bit)

Past NIET
in!!!

short
(16 bit)

casten

= verantwoordelijkheid van
de programmeur!
Wanneer loopt het mis?



```
aShort = (short)anInt;
```

Opdracht 3a: *Variabelen casten*

- Maak van onderstaand codefragment een programma

```
byte aByte;  
short aShortInt = 1568;  
aByte = aShortInt;
```

- Welke fout geeft de compiler?
- Pas vervolgens expliciete typeconversie toe. Welke waarde wordt er afgedrukt?
- Wijzig de waarde van *aShortInt* naar 115. Welke waarde wordt er nu afgedrukt?





- **Constance gehele getallen** (vb. 12) zijn standaard van het type `int`.
 - Je kan van dergelijk getal een long maken door een `l` of `L` achter het getal te plaatsen (vb. `12L`)
- **Constance floating-points** (vb. 12.567) zijn standaard van het type `double`.
 - Je kan van dergelijk getal een float maken door een `f` of `F` achter het getal te plaatsen (vb. `12.567F`)

```
float aDecimalNumber = 123.456F;
```



Opdracht 3b: *Constanten casten*

Vervolledig Opdracht 1 met het datatype *float* en *long*. Ken een literal toe aan deze variabelen.



3. Operatoren

3.1 Rekenkundige operatoren

Binaire operatoren: operand1 **operator** operand2

Binaire operator	Betekenis	Prioriteit
*	Vermenigvuldiging	1
/	Delen	1
%	Modulo	1
+	Optellen	2
-	Aftrekken	2

Modulo berekent de restwaarde van een deling
Vb. $7 \% 2 = 1$

Als in een berekening operatoren van gelijke voorrang voorkomen, wordt de berekening van links naar rechts uitgevoerd. Wensen we de voorrang te wijzigen, dan gebruiken we haakjes.

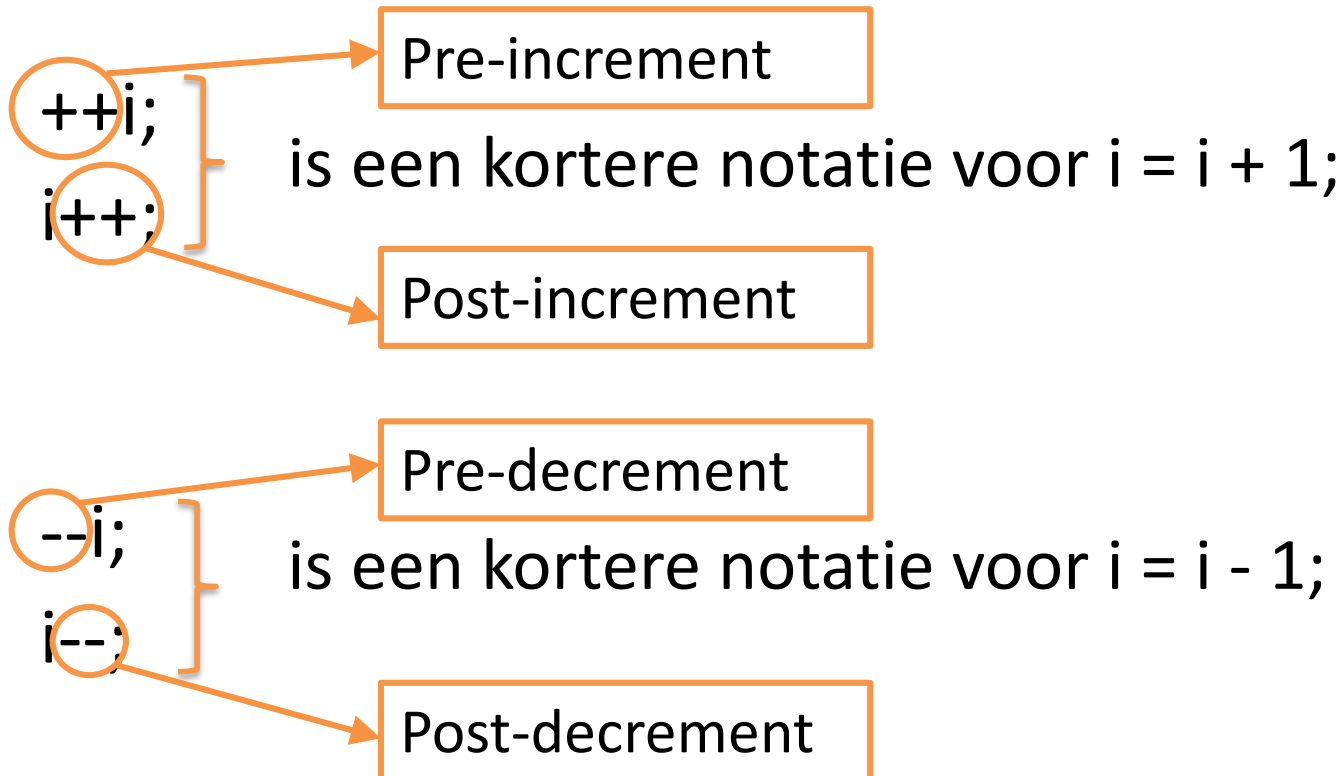


De + kan ook gebruikt wordt om tekst te concateneren.

```
public class SomVsConcat {  
  
    public static void main(String[] args) {  
        int waarde1 = 5;  
        int waarde2 = 7;  
        System.out.println(waarde1 + waarde2);  
                           // 12  
        System.out.println("concat: " + waarde1 + waarde2);  
                           // concat: 57  
        System.out.println("som: " + (waarde1 + waarde2));  
                           // som: 12  
    }  
}
```

Unaire operatoren:

Unaire operator	Betekenis
++	Verhoog met 1
--	Verlaag met 1



- ***Pre-increment:*** vb. $++i$;
Verhoogt eerst de waarde van i met 1 en gebruikt daarna de verhoogde waarde van i .
- ***Post-increment:*** vb. $i++$;
Gebruikt eerst de waarde van i en verhoogt ze daarna met 1.

Idem decrement



Voorbeeld:

aant = 10;

totAant = ++aant;

➔ De waarde van aant is: 11
De waarde van totAant is: 11

totAant = aant++;

➔ De waarde van aant is: 11
De waarde van totAant is: 10



Voorbeeld:

aant = 10;

totAant = --aant;

➔ De waarde van aant is: 9

De waarde van totAant is: 9

totAant = aant--;

➔ De waarde van aant is: 9

De waarde van totAant is: 10



Je kan deze pre- en post-operator combineren met (complexe) bewerkingen. Daardoor maak je het jezelf alleen maar moeilijk. Gebruik deze operatoren dus liefst in losstaande instructies! (Of vermijd deze operatoren te gebruiken)



Rangorde van de datatypes

Rangorde	Type
1	double
2	float
3	long
4	int
5	short
6	byte

Minimale type

Het datatype van het resultaat is minimaal *int* tenzij een van de operanden een type heeft dat hoger in rang is. In dat geval is het uiteindelijke type datgene van de operand met de hoogste rang.

Bijvoorbeeld:

byte + byte = int

byte + float = float

*short * byte = int*

*float * float = float*

int / int = int

int / double = double

!!! Hier maak je snel
een fout tegen.
Vb. $10 / 4 = 2$ i.p.v. 2.5



De enige uitzondering hierop zijn de operatoren ++ en --. Hierbij is het datatype van het resultaat gelijk aan het datatype van de operand.

Enkele voorbeelden:

```
int i;  
int n = 3;  
double d;  
i = n + 4;           // i krijgt de waarde 7  
i = n * 4;           // i krijgt de waarde 12  
i = 7 + 2 * 3;       // i krijgt de waarde 13  
n = n * (n + 2) * 4; // n krijgt de waarde 60  
d = 3.5 / 2;         // d krijgt de waarde 1.75  
n = 7 / 4;           // n krijgt de waarde 1
```

Opdracht 4:

Rekenkundige operatoren

a) Prioriteitsregels:

Wat is de waarde van *opl* na het uitvoeren van volgende instructies?

```
int a = 10;  
int b = 2;  
int c = 6;  
int opl;  
opl = c - a / b + 9;  
opl = 4 * a / c / b;  
opl = 4 * (a / c) / b;  
opl = a - c % 4 / b ;
```

b) Rangorde datatypes:

Wat is de waarde van *opl* na het uitvoeren van volgende instructies?

```
int a = 9;  
int b = 2;  
int c = 5;  
double opl;  
opl = c - a / b + 9;  
opl = 4 * a / c / b;  
opl = 4.0 * a / c / b;
```

c) Zijn onderstaande instructies mogelijk?

```
double a = 5;  
double b = 6;  
int opl = a + b;
```

```
short a = 5;  
short b = 6;  
short opl = a + b;
```



d)

```
int x, z;  
double y;
```

```
x = 7;
```

```
z = 2;
```

```
y = x / z;
```

Waarde van y?

```
int x, z;  
double y;
```

```
x = 7;
```

```
z = 2;
```

```
y = (double) x / z;
```

Waarde van y?

```
int x, z;  
double y;
```

```
x = 7;
```

```
z = 2;
```

```
y = (double) (x / z);
```

Waarde van y?



- e) Verbeter (indien nodig) onderstaande instructies.
Het type van de variabele mag niet aangepast worden.

```
int x;  
float z;  
x = 5;  
z = x * 5.2;
```

```
int x, y;  
long z;  
x = 5;  
y = 6;  
z = x + y;
```


f) Vul de gevraagde waarden in.

```
int i, iwaarde;  
float f, fwaarde;  
iwaarde = 22;  
fwaarde = 4.9F;  
f = iwaarde;           // f wordt? ...  
i = (int) fwaarde;     // i wordt? ...  
f = (float) ((5 + 6) / 2); // f wordt? ...  
f = (float) (5 + 6) / 2;  // f wordt? ...
```

g) Maak een programma met 2 variabelen *number1* en *number2*. Initialiseer deze variabelen met een bepaalde waarde. Druk vervolgens het volgende op het scherm af:

- de som
- het verschil
- het product
- de deling
- de rest

van beide getallen.



g) *vervolg*



Tracht te achterhalen waarvoor
<Ctrl> + <Space> dient!

Maak een variabele van het type char. Tel een waarde bij het karakter op en druk het af op het scherm (bekijk het resultaat!)



Een kleine uitbreiding:

Tot nu toe hebben we enkel integers ingelezen via het toetsenbord. Hoe los je dat op als andere datatypes wil inlezen via het toetsenbord?

I.p.v. `...nextInt()` kan je ook onderstaande varianten gebruiken:

- `...nextByte()`
- `...nextShort()`
- `...nextLong()`
- `...nextFloat()`
- `...nextDouble()`

Experimenteer hiermee in bv de voorgaande oefening.



Opdracht 5: *Datatypes en Casting*

- Vermenigvuldig 2 integers met de volgende waarden: 2 147 483 645 en 2 147 483 642. Ken het resultaat aan een derde integer toe en zie wat het resultaat is.
- Los het probleem op door het gebruik van de juiste datatypes en casting.

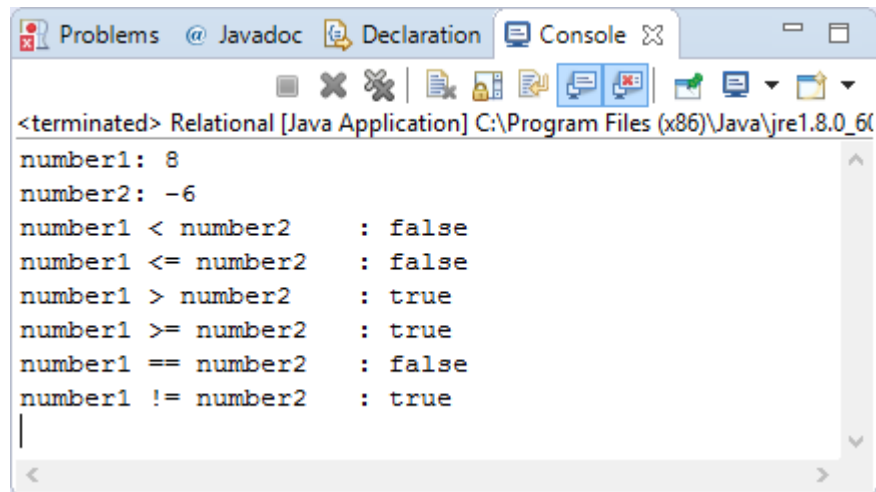


3.2 Relationele operatoren (= vergelijkingsoperatoren)

Relationele operator	Betekenis
>	groter dan
>=	groter of gelijk aan
<	kleiner dan
<=	kleiner of gelijk aan
==	gelijk aan
!=	verschillend van

Opdracht 6: *Relationele operatoren*

Maak een programma met 2 variabelen. Initialiseer deze variabelen met een bepaalde waarde. Ga vervolgens met alle relationele operatoren na welk de relatie is tussen de 2 getallen. Toon het resultaat op het scherm.



```
<terminated> Relational [Java Application] C:\Program Files (x86)\Java\jre1.8.0_60
number1: 8
number2: -6
number1 < number2      : false
number1 <= number2     : false
number1 > number2      : true
number1 >= number2     : true
number1 == number2     : false
number1 != number2     : true
```



Het wijzigen van een klassenaam, een variabele naam, ... doe je best met Refactor > Rename. Waarom?



Om een stuk code in commentaar te zetten: Selecteer de code: Source > Toggle Comment



Run = <Ctrl> + <F11>



3.3 Logische operatoren

Twee of meer relatiecondities kunnen gecombineerd worden m.b.v. logische operatoren

Logische operator	Resultaat
&&	true als beide condities true zijn
	true als 1 van beide condities true is
!	is het tegenovergestelde

Stel dat L1 en L2 enkelvoudige relatiecondities zijn, dan geeft de volgende tabel de waarde van de samengestelde relatieconditie.

L1	L2	L1 && L2	L1 L2	!L1
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Besluiten:

- L1 en L2 is alleen true als **beiden true zijn**
- L1 of L2 is enkel false als **beiden false zijn**
- !L1 is **het tegenovergestelde**



Indien bij de evaluatie van de 1^e operand het resultaat reeds bepaald is → 2^e operand wordt niet meer geëvalueerd. Dit kan van belang zijn wanneer de 2^e operand uit een bewerking bestaat.

= short circuit evaluation

- Vb1: Bij de &&-operator is de voorwaarde true als beide voorwaarden true zijn.

$$a > 0 \ \&\& \ b / a > 5$$

Als de eerste voorwaarde false is, heeft het in dit geval geen zin om de 2^e voorwaarde nog na te kijken. Als je de &&-operator gebruikt, wordt de 2^e voorwaarde alleen nagegaan als de 1^e true is.

- Vb2: Bij de ||-operator is de voorwaarde true als minstens 1 van beide voorwaarden true is.

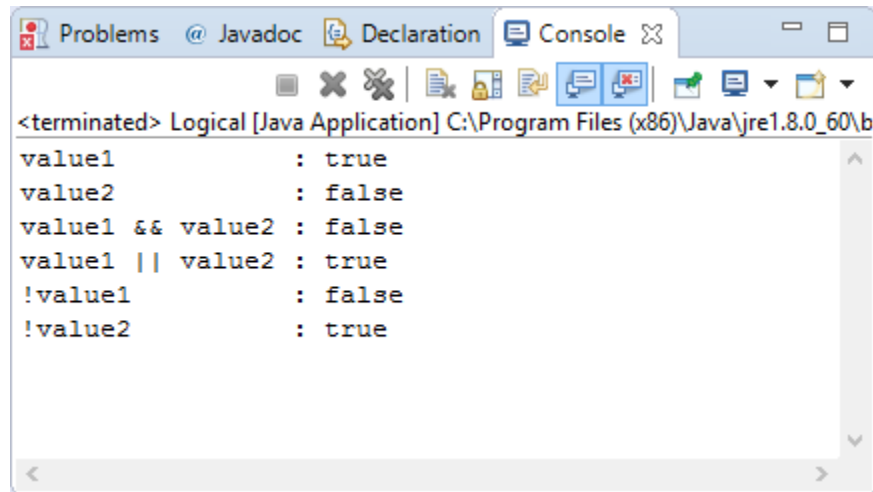
$$a < 0 \ || \ a > 100$$

Bij de ||-operator wordt de 2^e voorwaarde alleen nagegaan als de 1^e voorwaarde false is.



Opdracht 7: *Logische operatoren*

Maak een programma met 2 boolean variabelen met een initiële waarde. Gebruik de logische operatoren en toon het resultaat op het scherm (het is niet nodig daarvoor een if te gebruiken).



```
<terminated> Logical [Java Application] C:\Program Files (x86)\Java\jre1.8.0_60\b
value1           : true
value2           : false
value1 && value2  : false
value1 || value2 : true
!value1          : false
!value2          : true
```

3.4 Toekenningsoperatoren

Toekenningsoperator	Resultaat
=	De variabele links van het =-teken krijgt de waarde van hetgeen rechts van het =-teken staat.

Indien het datatype rechts van het =-teken van een hogere rangorde is, zal de compiler een foutmelding geven. Een expliciete typconversie (casting) is dan vereist.

Vb.

```
int get1;  
long get2 = 100;  
get1 = (int)get2;
```

De toekenningoperator wordt vaak gecombineerd met andere operatoren.

Gecombineerde toekenningoperator	Gebruik	Gelijk aan
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Bij het gebruik van de gecombineerde toekenningoperator geldt de regel van het minimale datatype niet.

Vb.

```
byte b1 = 5;  
byte b2 = 7;  
b1 = b1 + b2; // Wrong  
b1 += b2;     // Good
```



Opdracht 8: *Toekenningoperator*

Maak een programma met 2 variabelen en pas de gecombineerde toekenningoperator toe.



3.7 Prioriteitsregels

Operator	Prioriteit
!	1
* / %	2
+ -	3
< > <= >=	4
== !=	5
&&	6
	7
= += -= *= /= %=	8

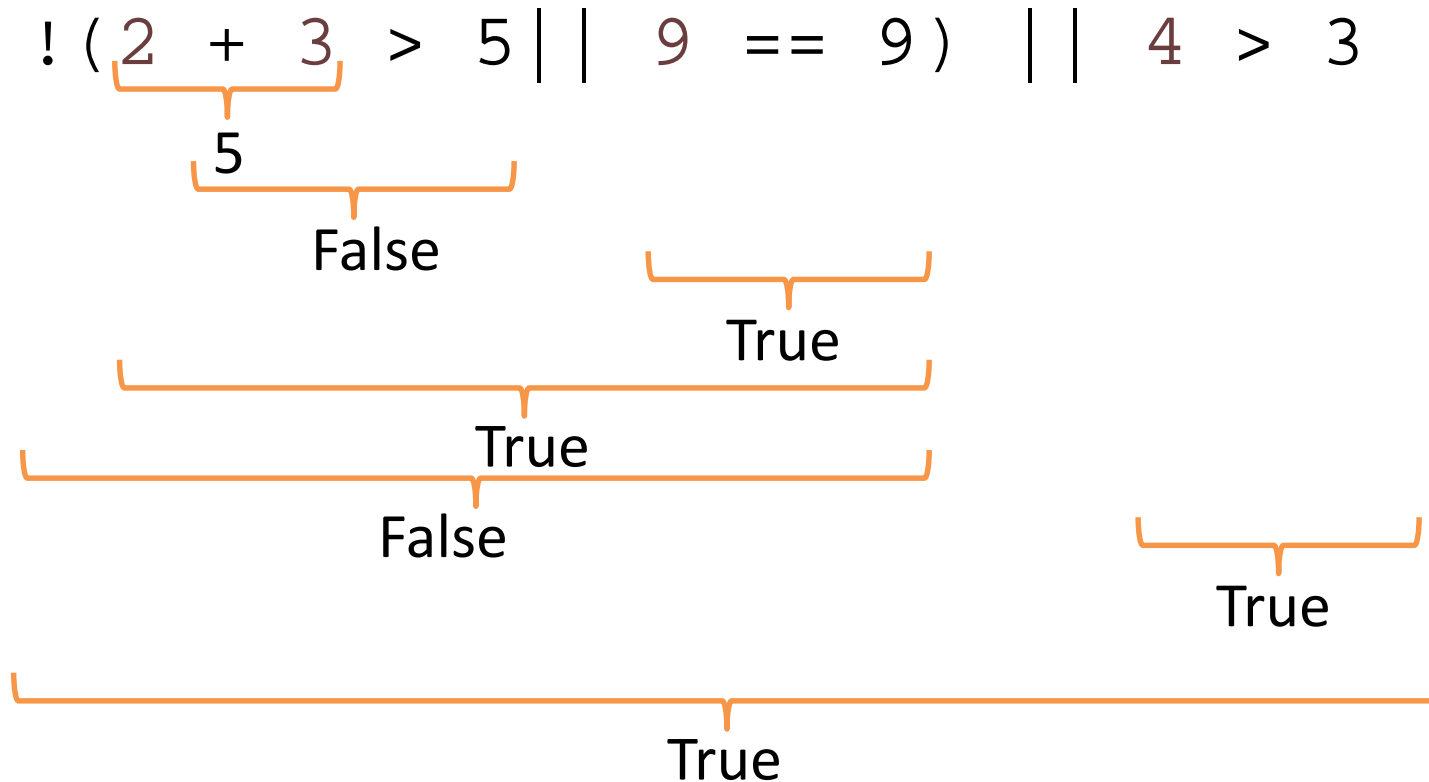
Bewerkingen met dezelfde prioriteit worden van links naar rechts uitgevoerd.

Als je de prioriteit van de bewerkingen wil wijzigen gebruik je haakjes.

Tip: gebruik ook haakjes om de leesbaarheid te vergroten!



Voorbeeld:



➔ Het resultaat van deze uitdrukking is True.

Opdracht 9:

Relationele operatoren, logische operatoren en prioriteitsregels

Zijn volgende logische uitdrukkingen true of false?

```
int x = 30;  
int y = 40;  
int a = 10;  
int b = 5;  
x + 2 < y || !(y == 40 && a * 2 > 5)  
!(x + y < 10 || a == 9) && b > 3  
x + y != 100 || (a == 1 && b > 3)
```

4. Uitdrukkingen, statements en blokken

codeblok =

```
{  
    int num1 = 5;  
    int num2 = 10;  
    int sum;  
    sum = num1 + num2;  
    System.out.println(sum);  
}
```

= uitdrukking
(resulteert in
een eenvoudige
waarde)

= statements (eindigen op ;)



Codeblok

- = groepering van een aantal statements tussen accolades
- heeft invloed op het bereik (scope) van een variabele (= gebied waar men gebruik kan maken van een variabele)



```
{  
    // Codeblok 1  
    ...  
    {  
        // Codeblok 2  
        int anInteger;  
        ...  
        {  
            // Codeblok 3  
            ...  
        }  
        ...  
    }  
    ...  
    {  
        // Codeblok 4  
        ...  
    }  
    ...  
}
```

Deze variabele kan enkel in Codeblok 2 en Codeblok 3 gebruikt worden.

Best laat je ieder codeblok inspringen.

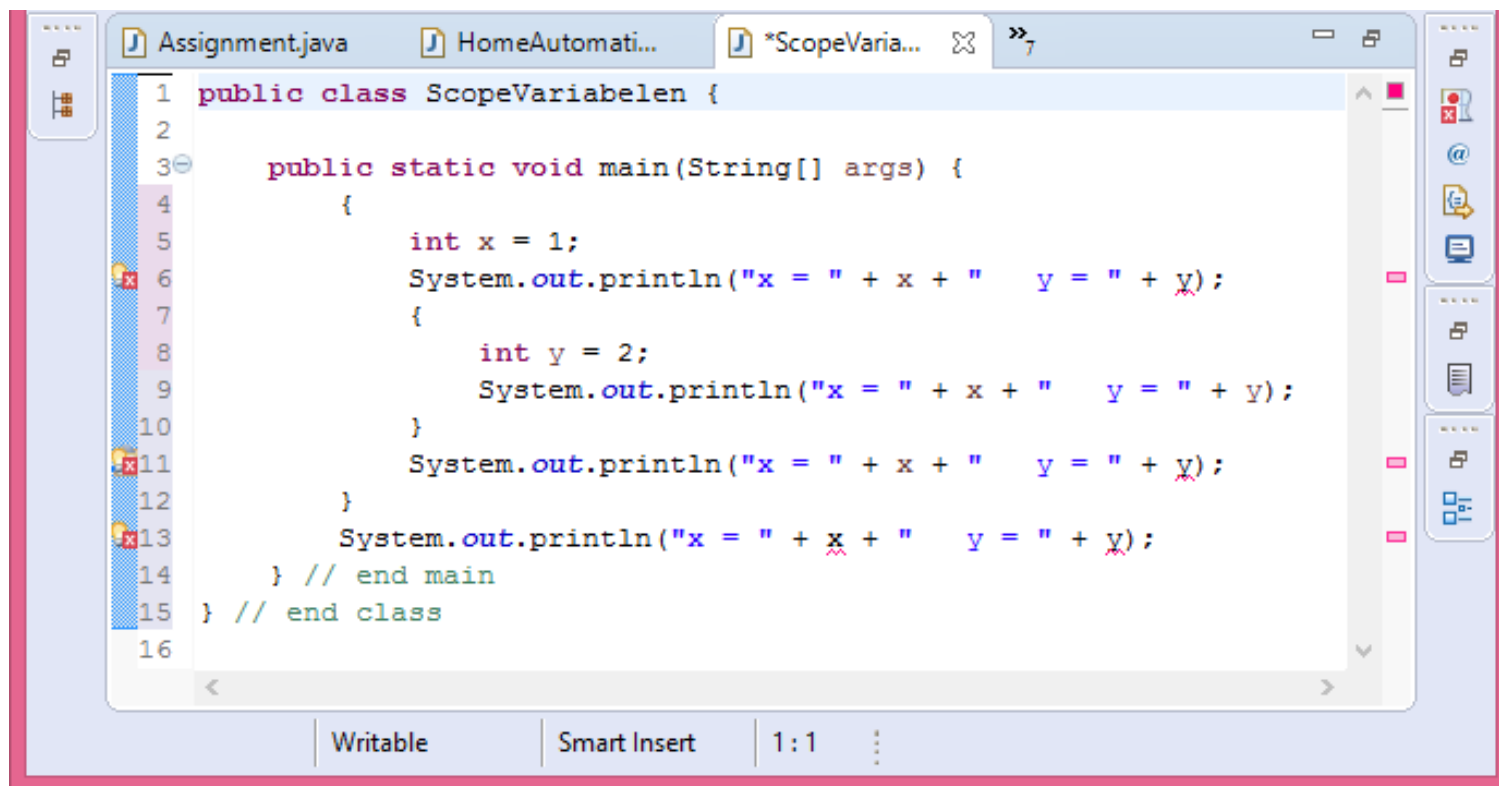


Automatisch inspringen:
Source > Format



Opdracht 10: *Het bereik van variabelen*

Waarom geeft de compiler een foutindicatie in lijn 6, 11 en 13?



The screenshot shows an IDE window with a Java file named `Assignment.java`. The code defines a class `ScopeVariabelen` with a `main` method. The code is as follows:

```
1 public class ScopeVariabelen {
2
3     public static void main(String[] args) {
4         {
5             int x = 1;
6             System.out.println("x = " + x + "    y = " + y);
7         }
8         int y = 2;
9         System.out.println("x = " + x + "    y = " + y);
10    }
11    System.out.println("x = " + x + "    y = " + y);
12 }
13 System.out.println("x = " + x + "    y = " + y);
14 } // end main
15 } // end class
16
```

The IDE highlights three lines with red 'x' icons, indicating compilation errors:

- Line 6: `y` is used before it is declared.
- Line 11: `y` is used before it is declared (in the outer scope).
- Line 13: `x` is used after it has been declared and its scope has ended.

The IDE interface includes a toolbar on the left with icons for file operations, a right sidebar with tool windows, and a status bar at the bottom with options like 'Writable', 'Smart Insert', and '1:1'.

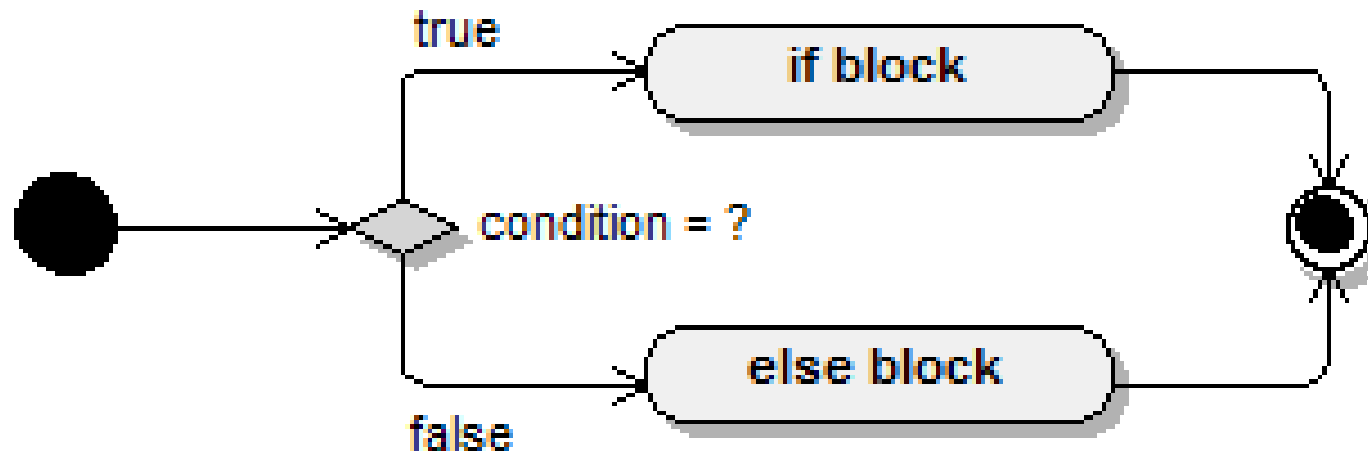
5. Programmaverloop-statements

5.1 Inleiding

- *if ... else* (selectie)
- *switch* (selectie)
- *while en do ... while* (herhaling)
- *for* (herhaling)



5.2 Het *if* ... *else* statement



else block is optioneel

```
package examples.algorithms.selection;

import java.util.*;

public class Age {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Enter your age:");
        int age = keyboard.nextInt();

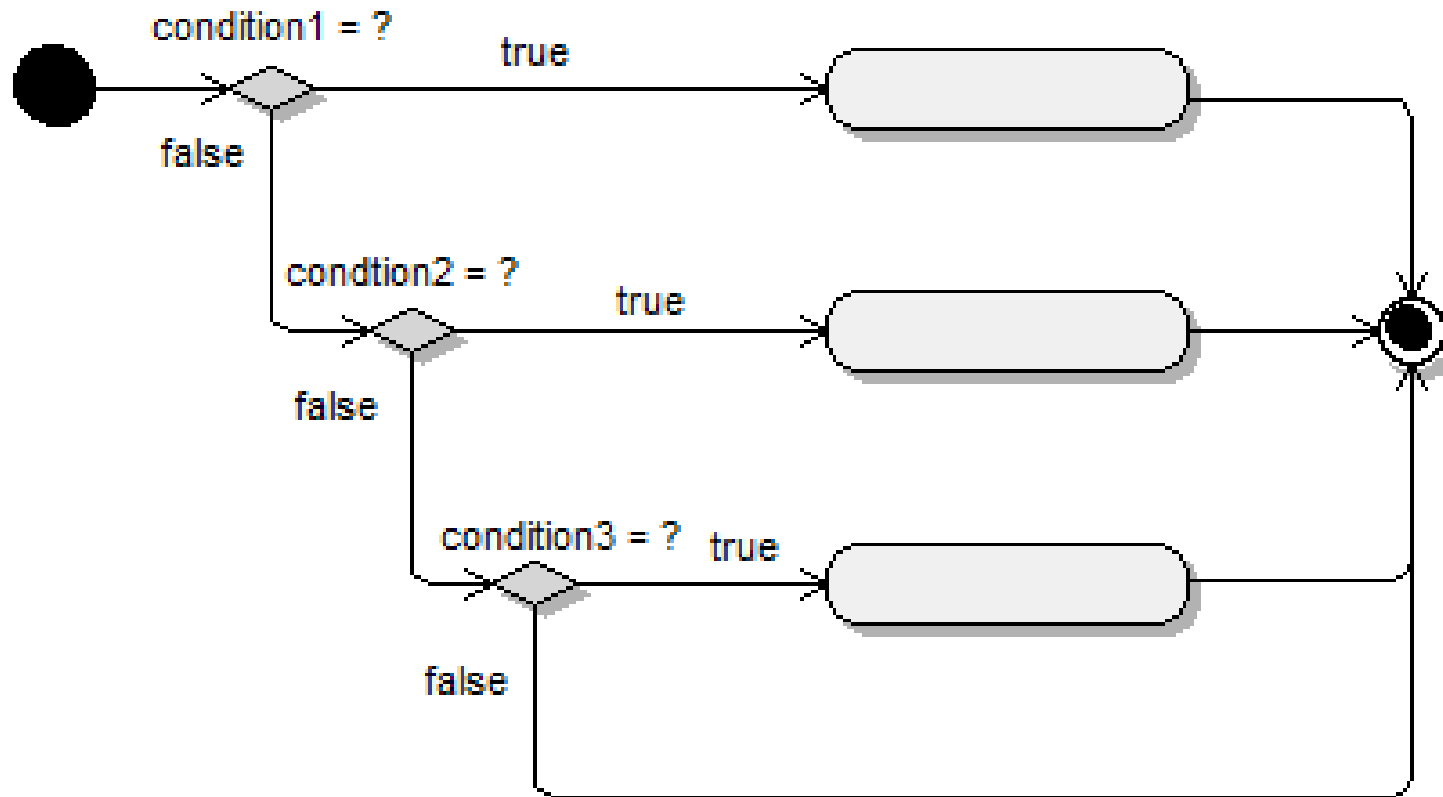
        if (age >= 18) {
            System.out.println("You are an adult");
        } else {
            System.out.println("You are a child");
        }

        keyboard.close();
    }
}
```

Het is een goede gewoonte om
altijd een codeblok te gebruiken.
Ook al is er maar 1 statement bij
de *if* of de *else*.



Geneste *if ... else*



```
if (age >= 18) {  
    System.out.println("You are an adult");  
} else {  
    if (age >= 10) {  
        System.out.println("You are a teenager");  
    } else {  
        if (age >= 2) {  
            System.out.println("You are a child");  
        } else {  
            System.out.println("You are a baby");  
        }  
    }  
}  
}
```



Opdracht 11: *if ... else*

a)

```
if (a > 100) {  
    System.out.println("a > 100");  
} else {  
    if (a < 10) {  
        System.out.println("a < 10");  
    } else {  
        System.out.println("a >= 10");  
    }  
}
```

```
if (a > 100) {  
    System.out.println("a > 100");  
}  
if (a < 10) {  
    System.out.println("a < 10");  
} else {  
    System.out.println("a >= 10");  
}
```

Zijn deze programma-
stukken equivalent?
Indien niet, leg uit
waarom niet.



b) Zijn volgende programma-stukken equivalent?
Indien niet, leg uit waarom niet.

```
if (x != 1) {  
    b = x;  
}  
if (x == 2) {  
    a = x;  
}
```

```
if (x != 1) {  
    b = x;  
} else {  
    if (x == 2) {  
        a = x;  
    }  
}
```

c) Wat zijn de waarden van a en b na het uitvoeren van volgende opdrachten?

```
a = 5;  
b = 3;  
if (a < b)  
    a = 2 * a;  
    b = a + b;
```

```
a = 5;  
b = 3;  
if (a < b){  
    a = 2 * a;  
    b = a + b;  
}
```

```
a = 3;  
b = 5;  
if (a < b)  
    a = 2 * a;  
    b = a + b;
```

```
a = 3;  
b = 5;  
if (a < b){  
    a = 2 * a;  
    b = a + b;  
}
```

- d) Maak een programma dat de **BMI** van een persoon berekent. Vraag de gebruiker naar zijn lengte en gewicht en bereken de BMI. Steek deze waarde in een afzonderlijke variabele. Geef vervolgens medisch advies.

lager dan 18: ondergewicht
18 tot 25: ok
25 tot 30: overgewicht
30 tot 40: obesitas
40 en hoger: ziekelijk overgewicht

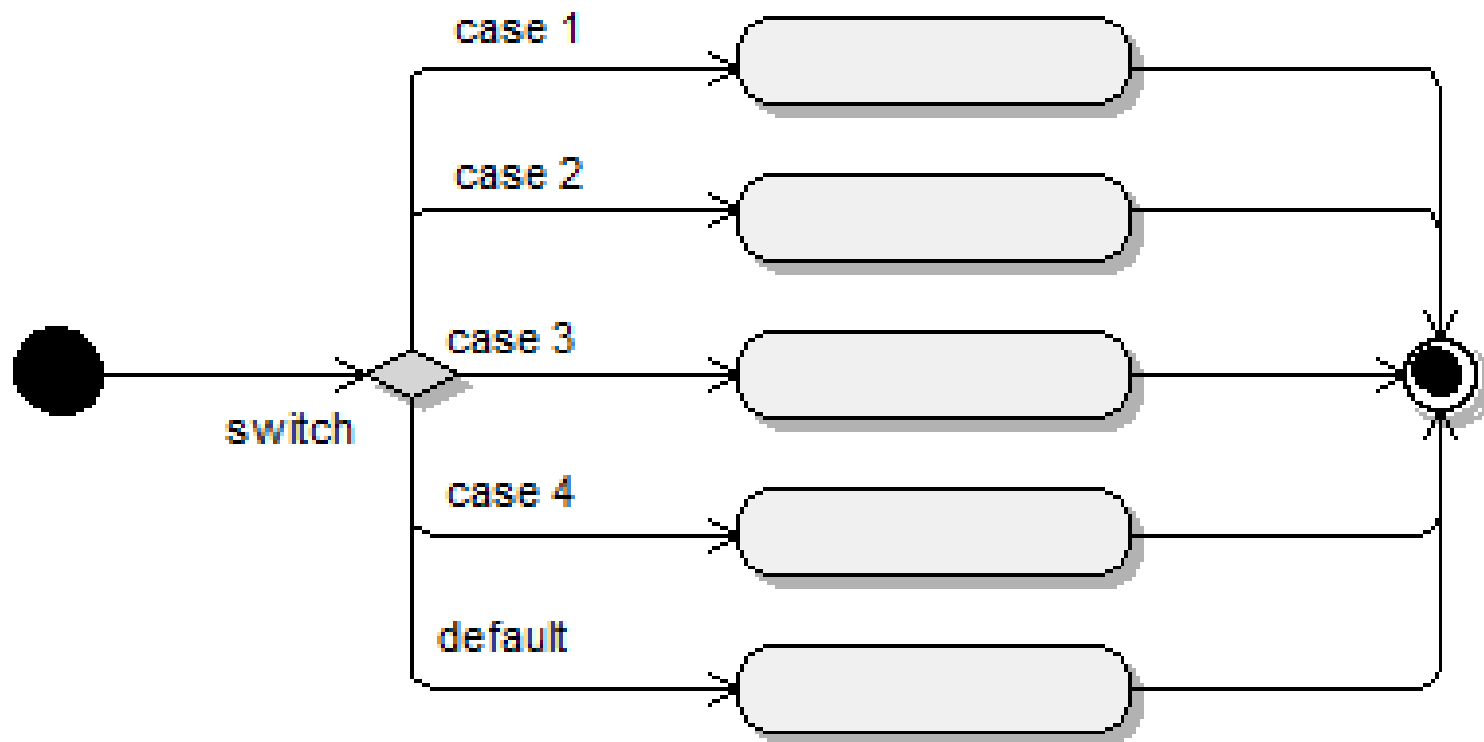
$$\text{BMI} = \frac{\text{gewicht in kg}}{(\text{lengte in m}) * (\text{lengte in m})}$$



- e) Schrijf telkens een programma voor volgende vier situaties waarbij het jaarlijkse lidgeld voor een sportvereniging dient berekend te worden. De burgerlijke staat (dit is een cijfer van 1 tot en met 3: 1 = ongehuwd; 2 = gehuwd; 3 = weduwe(naar)) en de leeftijd dienen ingegeven te worden.
- a) ongehuwd: 25 euro; gehuwd: 20 euro; weduwe(naar): 15 euro.
 - b) ongehuwd jonger dan 30: 25 euro; alle overige betalen 15 euro.
 - c) alle leden jonger dan 30 en alle ongehuwden: 25 euro; alle overige betalen 15 euro.
 - d) ongehuwd: 25 euro; gehuwd jonger dan 30: 20 euro; alle overige betalen 15 euro.



5.3 Het *switch* statement



```
switch (value) {  
    case literal1:  
        statements;  
        break;  
    case literal2:  
        statements;  
        break;  
    ...  
    default:  
        statements;  
}
```

= a non-long integer: d.w.z.
een byte, short, int, char of
String

Zien we in hoofdstuk 6

Breekt de uitvoering af en
gaat verder na de accolade
van het *switch*- blok.

```
if (dagNr == 1) {  
    System.out.println("Maandag");  
} else {  
    if (dagNr == 2) {  
        System.out.println("Dinsdag");  
    } else {  
        if ...  
        ... else {  
            if (dagNr == 7) {  
                System.out.println("Zondag");  
            }  
        }  
    }  
}  
}
```



```
switch (dagNr) {  
    case 1:  
        System.out.println("Maandag");  
        break;  
    case 2:  
        System.out.println("Dinsdag");  
        break;  
    ...  
    case 7:  
        System.out.println("Zondag");  
}
```



case 6:

```
System.out.println("Hoera");  
System.out.println("Zaterdag");  
break;
```

Meerdere opdrachten
binnen case

case 6:

case 7:

```
System.out.println("Weekend");  
break;
```

= fall through

Code valt doorheen alle cases
totdat een *break* de val stopt.

Meerdere cases bij
(een) opdracht(en)

switch (dagNr) {

...

default:

```
System.out.println("Verkeerde dagnr");
```

}

Default is
optioneel



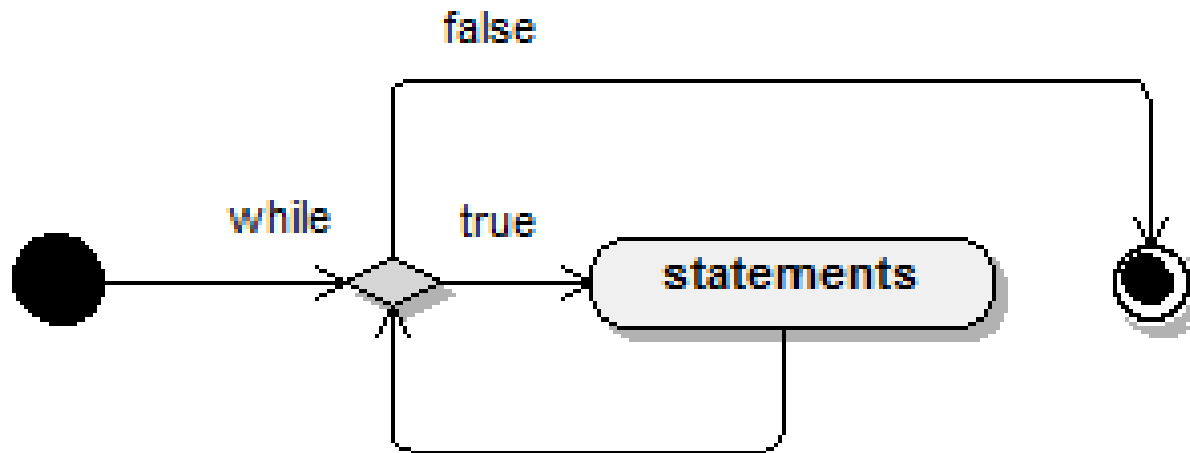
Opdracht 12: *switch*

Zet volgend stukje Java code om naar een switch statement.

```
if (x == 1){
    System.out.println("het getal is 1");
} else {
    if ((x == 2) || (x == 3)){
        System.out.println("het getal is 2 of 3");
    } else {
        System.out.println("het getal is niet 1, 2 of 3");
    }
}
```

5.3 Het *while* en *do ... while* statement

while (test vooraf)

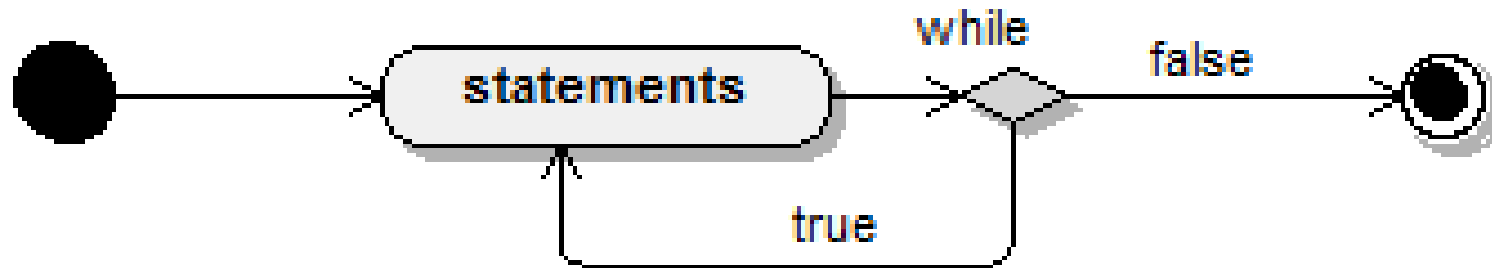



```
int personeelsNr;  
int aantalUrenGewerkt;  
Scanner in = new Scanner(System.in);  
System.out.println("Geef personeelsNr (0 om te eindigen): ");  
personeelsNr = in.nextInt();  
while (personeelsNr != 0){  
    System.out.println("Geef aantalUrenGewerkt: ");  
    aantalUrenGewerkt = in.nextInt();  
    ...  
    System.out.println("Geef personeelsNr (0 om te eindigen): ");  
    personeelsNr = in.nextInt();  
}
```

- Test vooraf
- De waarde van de variabele in de conditie moet altijd gewijzigd worden in de body van de while! Waarom?
- Meestal krijgt deze variabele een nieuwe waarde als 'laatste statement' van de body van de while. Waarom?



do ... while (test achteraf)



```
int count = 1;  
do {  
    System.out.println(count);  
    count++;  
} while (count < 5);
```

- Body wordt minimaal 1x uitgevoerd
- Test gebeurt achteraf



Opdracht 13: *while*

a) Wat is het resultaat van onderstaande opdrachten?

```
Scanner keyboard = new Scanner(System.in);
int a;
System.out.println("Geef een getal in: ");
a = keyboard.nextInt();
while (a < 5)
    System.out.println("Geef een ander getal in: ");
    a = keyboard.nextInt();
```

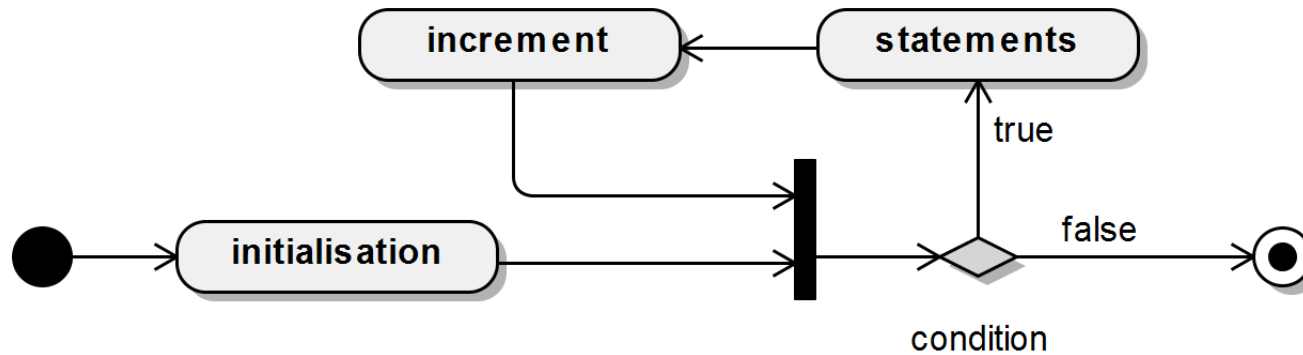
```
Scanner keyboard = new Scanner(System.in);
int a;
System.out.println("Geef een getal in: ");
a = keyboard.nextInt();
while (a < 5) {
    System.out.println("Geef een ander getal in: ");
    a = keyboard.nextInt();
}
```



b) Maak een programma dat de gebruiker vraagt om een getal in te geven tussen 0 en 10. Indien de gebruiker een ongeldig getal ingeeft, herhaal je de vraag.



5.3 Het for statement (of zelftellende lus)



1
for (*initialisation*; *condition*; *increment*) {
statements; 3
}

2 5 4



condition
moet 'true'
zijn vooraleer
de body van
de lus
uitgevoerd
wordt!

```
for (int tel = 0; tel <= 5; tel++) {  
    System.out.println(tel);  
}
```

Hoe vaak wordt bovenstaande lus uitgevoerd?

Opdracht 14: *for*

a) Hoe vaak wordt onderstaande lus uitgevoerd?

```
int getal1 = 7;  
int getal2 = 6;  
int grens = getal1 + getal2 ;  
for (int tel = 0; tel == grens; tel++) {  
    System.out.println(tel);  
}
```


b) Wat is het resultaat van onderstaande opdrachten?

```
Scanner keyboard = new Scanner(System.in);
int b, dubbel;
for (int a = 1; a <= 5; a++)
    System.out.println("Geef een getal in: ");
    b = keyboard.nextInt();
    dubbel = b * 2;
    System.out.println("Het dubbel is: " + dubbel);
```

```
Scanner keyboard = new Scanner(System.in);
int b, dubbel;
for (int a = 1; a <= 5; a++) {
    System.out.println("Geef een getal in: ");
    b = keyboard.nextInt();
    dubbel = b * 2;
    System.out.println("Het dubbel is: " + dubbel);
}
```



- c) Maak een programma dat de getallen van 350 t.e.m. 400 afdruckt in omgekeerde volgorde.
- d) Maak een programma dat alle veelvouden van 7 afdruckt kleiner dan 200.
- e) Maak een programma dat de letters 'z' t.e.m. 'a' op het scherm afdruckt.
- f) Maak een programma dat alle getallen afdruckt tussen -10 en +10. Voeg bij de positieve getallen het +-teken toe bij het afdrukken van het getal. Het getal 0 heeft geen teken.
- g) Maak een programma dat alle getallen tussen 0 en 10000 afdruckt die zowel deelbaar zijn door 6 als door 28. Hint: maak gebruik van de %-operator.



Opdracht 15: *for, while en do ... while*

a)

```
int g = 1;
for (int i = 2; i < 8; i += 3) {
    System.out.println(g);
}
```

- Hoeveel keer wordt bovenstaande lus doorlopen?
- Herformuleer met een *while* structuur.
- Herformuleer met een *do ... while* structuur.

b)

```
Scanner keyboard = new Scanner(System.in);
int g;
int tot = 0;
int i = 12;
do {
    g = keyboard.nextInt();
    tot = tot + g;
    i = i - 2;
} while (i != 9) ;
```

- Hoeveel getallen worden er ingelezen?
- Welke voorwaarde had er moeten staan om 3 getallen in te lezen?
- Herformuleer met een *for* structuur (zodat 3 getallen ingelezen worden).
- Herformuleer met een *while* structuur (zodat 3 getallen ingelezen worden).



6. Methoden

- Om code herhaaldelijk uit te voeren: een telkens terugkerende reeks van programmeerregels ga je groeperen en deze groep geef je een naam = een methode
- Om code te structureren (methode moet niet noodzakelijk herhaaldelijk uitgevoerd worden)

```
public class Methods {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = 6;  
        int c;  
        if (a > b){  
            c = a;  
        } else {  
            c = b * 5;  
        }  
        System.out.println(c);  
  
        int d = 7;  
        int e = 8;  
        int f;  
        if (d > e){  
            f = d;  
        } else {  
            f = e * 5;  
        }  
        System.out.println(f);  
    }  
}
```

Dit stuk code
komt
herhaaldelijk
voor!

Methode
maken!

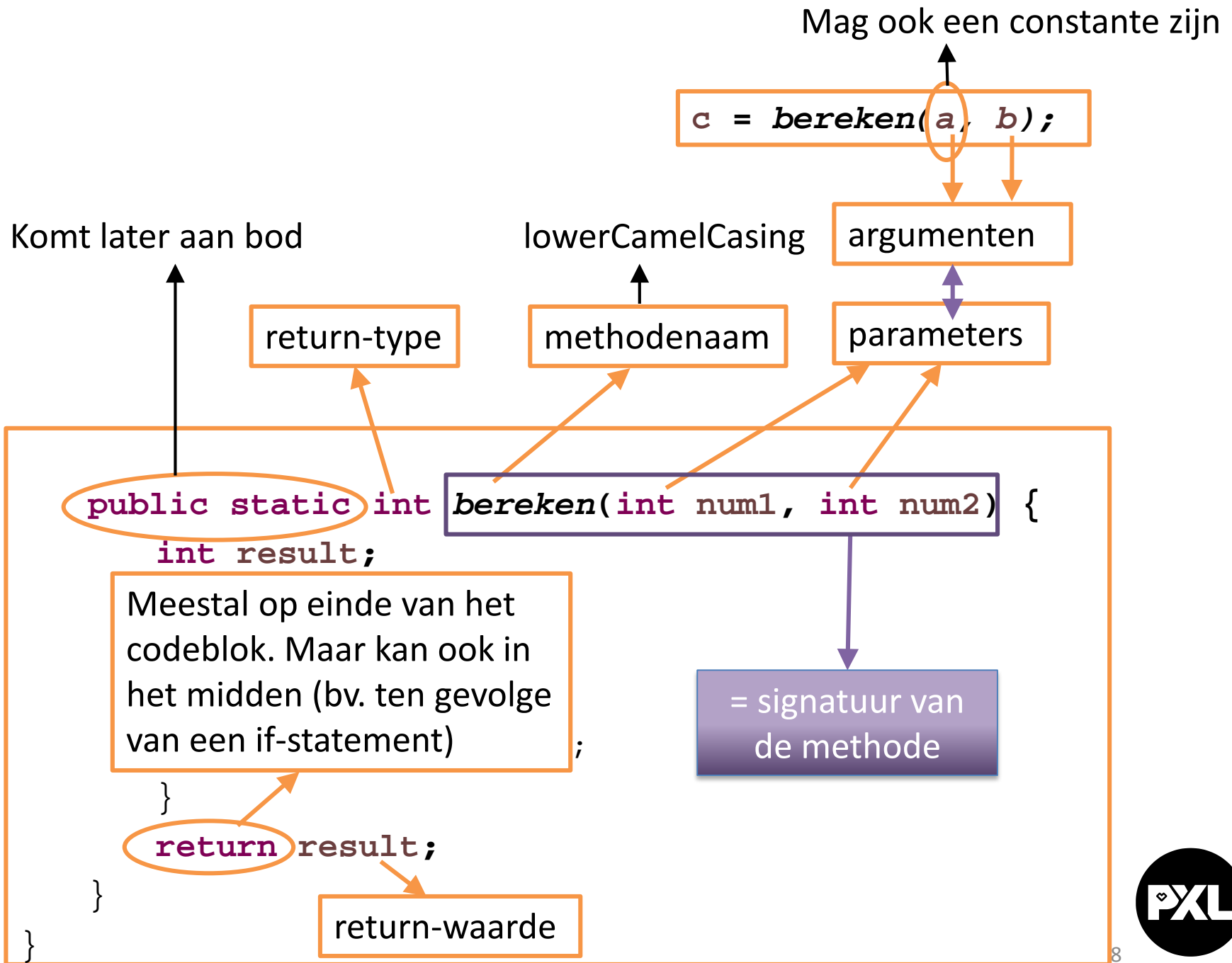
```
public class Methods {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = 6;  
        int c = bereken(a, b);  
        System.out.println(c);  
  
        int d = 7;  
        int e = 8;  
        int f = bereken(d, e);  
        System.out.println(f);  
    }  
  
    public static int bereken(int num1, int num2) {  
        int result;  
        if (num1 > num2) {  
            result = num1;  
        } else {  
            result = num2 * 5;  
        }  
        return result;  
    }  
}
```

De waarde van de originele variabele (a, b, d of e) kan door de methode nooit gewijzigd worden. De inhoud (waarde) van de variabele wordt doorgegeven naar de **lokale** variabele van de methode.

= Call by value

De lokale variabele mag dezelfde naam hebben als de originele variabele. Waarom?






```
public static int bereken(int num1, int num) {  
    ...  
}  
  
public static int bereken (int num1, int num2, int num3) {  
    ...  
}  
  
public static float bereken (float num1, float num2) {  
    ...  
}
```

= **methodoverloading**

= Methoden met dezelfde naam
maar met een verschillende
parameterlijst.

Signatuur moet uniek zijn!



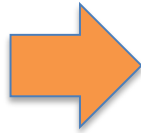
De compiler zal telkens o.b.v. de meegegeven argumenten de juiste methode oproepen:

```
bereken(1, 2);      ➔ bereken(int num1, int num2)
bereken(1, 2, 3); ➔ bereken(int num1, int num2, int num3)
bereken(1F, 2F);   ➔ bereken (float num1, float num2)
bereken(1.0, 2);   ➔ ???
```



Een methode moet niet noodzakelijk een parameter hebben *(komt aan bod in Hoofdstuk 6; vb. de methode toUpperCase())*

Een methode moet niet noodzakelijk een waarde teruggeven



void

```
printSum(1, 2);
```

```
public static void printSum(int a, int b){  
    System.out.println("Sum: " + (a + b));  
}
```

➔ Geen return-statement nodig

Opdracht 16: *methoden*

- a) Maak een methode (toonTafel) om de tafel van vermenigvuldiging te tonen. Vraag in het hoofdprogramma aan de gebruiker welke tafel hij wil zien.
- b) Schrijf een programma om het volgende te realiseren:
De gebruiker geeft een belastbaar bedrag in waarna de verschuldigde belasting op het beeldscherm verschijnt. De verschuldigde belasting wordt met een methode berekend. Voor de eerste 25000€ moet 38,4% belastingen betaald worden, voor de volgende 30000€ moet 50% belastingen betaald worden en voor elke € meer 60% belastingen.





Als je van een stuk bestaande code een methode wenst te maken → selecteer de code > RM > **Refactor** > **Extract Method ...**



Experimenteer ook eens met: selecteer de code > RM > **Surround With >**

- **do**
- **for**
- **if**
- **while**

!!! Je moet de syntax wél vanbuiten kennen (schriftelijk deel examen)