

HO GENT

H2 Controlestructuren en operatoren

Table of Contents

1. Doelstellingen	1
2. Inleiding	1
3. Algoritmen	1
4. Pseudocode	2
5. Controlestructuren	2
6. De selectiestructuren	3
6.1. De "if" selectiestructuur	3
6.2. De "if-else" selectiestructuur	7
6.3. Geneste "if-else" selectiestructuren	9
6.4. De conditionele operator (?:)	12
6.5. De "switch" meervoudige selectiestructuur	14
7. De herhalingsstructuren	18
7.1. Inleiding	18
7.2. Tellergestuurde lussen	18
7.2.1. De while-lus met teller	18
7.2.2. De for-lus	23
7.3. De andere lussen	32
7.3.1. De while-lus met schildwacht	32
7.3.2. De do-while-lus	37
7.4. Samenvatting herhalingsstructuren	39
8. Samengestelde toekenningsoperatoren	39
9. Increment- en decrementoperatoren	41
10. Logische operatoren	44
10.1. Waarheidstabel conditionele en (&&)	44
10.2. Waarheidstabel conditionele of ()	45
10.3. Short-circuit operatoren	45
10.4. Waarheidstabel logische exclusieve of (^)	46
10.5. Waarheidstabel logische niet (!)	46
11. Precedentie en associativiteit van de operatoren	48
12. De statements break en continue	51
13. Primitieve datatypes	55
13.1. Het primitieve datatype: boolean	55
13.2. Het primitieve datatype: char	56
13.3. Het primitieve datatype: byte	57
13.4. Het primitieve datatype: short	57
13.5. Het primitieve datatype: int	58
13.6. Het primitieve datatype: long	58
13.7. De primitieve datatypes: float en double	58

1. Doelstellingen

Na het bestuderen van dit hoofdstuk ben je in staat

- Kan de juiste controlestructuur kiezen
- Kan een correcte controlestructuur schrijven in Java

2. Inleiding

Waarom controlestructuren?

- principe van gestructureerd programmeren toepassen
- controlestructuren helpen om methodes leesbaar en performant uit te werken

3. Algoritmen

Een algoritme is een procedure om een probleem op te lossen in termen van

- de opdrachten die uitgevoerd worden
- de volgorde waarin ze worden uitgevoerd

Voorbeeld

Sta op en ga werken algoritme

- 1) Kom uit bed
- 2) Doe je pyjama uit
- 3) Neem een douche
- 4) Kleed je aan
- 5) Neem een gezond ontbijt
- 6) Carpool naar het werk

Voorbeeld (zelfde stappen, verschillende volgorde) Sta op en ga werken algoritme

- 1) Kom uit bed
- 2) Doe je pyjama uit
- 3) Kleet je aan
- 4) Neem een douche
- 5) Neem een gezond ontbijt
- 6) Carpool naar het werk

VERSCHILLEND EINDRESULTAAT?

4. Pseudocode

Pseudocode is ...

- een informele taal om algoritmen te ontwikkelen
- GEEN taal die uitgevoerd wordt door computers
- een taal die softwareontwikkelaars helpt bij het "uitdenken" van algoritmen
- meestal beperkt tot de uitvoerbare statements, geen declaraties

5. Controlestructuren

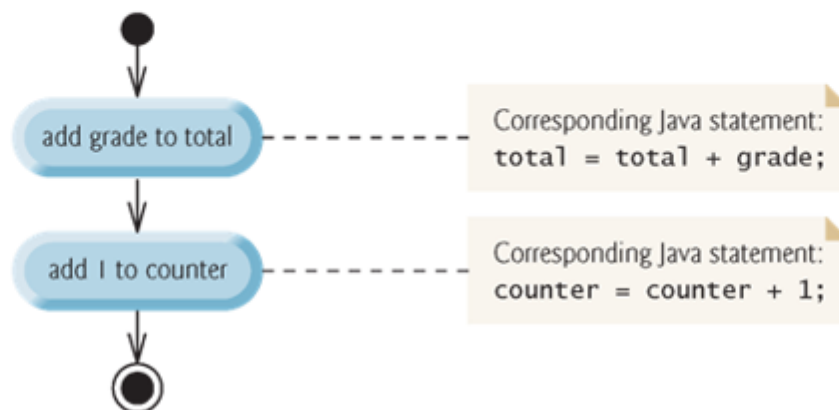
Programmacontrole is de volgorde waarin de acties uitgevoerd worden in een programma en 3 controlestructuren helpen hierbij:

- sequentiestructuur: **Java heeft een “ingebouwde” sequentiestructuur, dit is default behaviour. De statements in een programma worden één voor één uitgevoerd in de volgorde waarin ze geschreven zijn.**
- selectiestructuur: **Java voorziet drie selectiestructuren**
 - if
 - if/else
 - switch-case
- herhalingsstructuur: **Java voorziet drie herhalingsstructuren**
 - while
 - do/while
 - for

Elk van deze woorden is een Java-sleutelwoord.

- Activity diagram
 - Grafische representatie van een algoritme.
 - Lijnen geven de volgorde aan waarin de acties worden uitgevoerd.

Activity diagram van een sequentiestructuur



6. De selectiestructuren

6.1. De "if" selectiestructuur

if (voorwaarde) statement

- Het statement wordt alleen uitgevoerd als de voorwaarde WAAR is (true).
- Het statement kan ook op de volgende lijn staan.
- Als het statement enkelvoudig is, dan wordt de selectiestructuur afgesloten met het punt-komma in het statement. Als het statement samengesteld is, dan wordt de selectiestructuur afgesloten met de eind-accolade van het statement.
- Voorwaarden kunnen geformuleerd worden met relationele operatoren en/of gelijkheidsoperatoren
- De ronde haakjes rond de voorwaarde zijn verplicht!

Operator	In java	
=	==	exact gelijk
≠	!=	verschillend
>	>	groter dan
<	<	kleiner dan
≥	>=	groter dan of gelijk aan
≤	<=	kleiner dan of gelijk aan

- Prioriteitsregels:

haakjes: ()

*, / , %

+, -

<, <=, >, >=

==, !=

=

Bij gelijke prioriteit ⇒ regels van de associativiteit (van L naar R)

Uitzondering: = (assignment)

x = y = z; wordt geëvalueerd als x = (y = z);

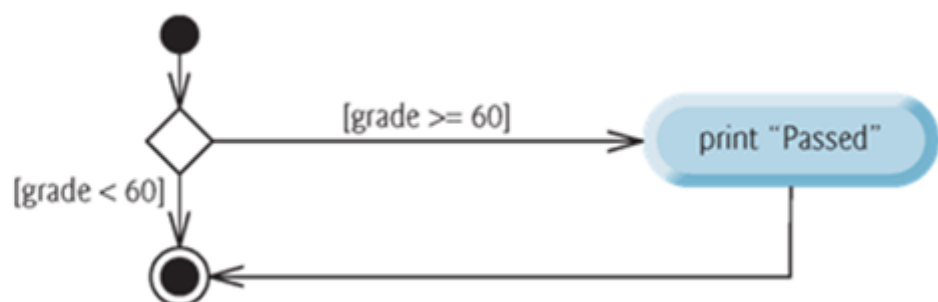
Voorbeeld 1

- in pseudocode

```

Als punten groter dan of gelijk aan 60
    dan print "Geslaagd!"
Eind-als
  
```

- activity diagram



Structuur met één ingang en één uitgang.

- in Java

```
if (punten >= 60) System.out.println("Geslaagd");

// OF het statement op de volgende lijn
if (punten >= 60)
    System.out.println("Geslaagd");

// OF met accolades (niet verplicht als er maar 1 statement is)
if (punten >= 60) { System.out.println("Geslaagd"); }

// OF
if (punten >= 60)
{
    System.out.println("Geslaagd");
}
```

Voorbeeld 2

- in pseudocode

```
Als punten gelijk is aan 20
  dan print "Je behaalt een credit!";
  aantalGeslaagden <- aantalGeslaagden + 1
Eind-als
```

- in Java

```
if (punten == 20)
{
    System.out.println("Je behaalt een credit!");
    aantalGeslaagden = aantalGeslaagden + 1;
}
```

De accolades zijn hier noodzakelijk, gezien de 2 statements een samengesteld statement vormen en enkel uitgevoerd worden als de voorwaarde waar is!

Voorbeeld 3

- gewenste uitvoer

Enter first integer: 12

Enter second integer: 36

12 != 36

12 < 36

12 <= 36

- in Java

```
1 package cui;
2
3 import java.util.Scanner;
4
5 public class Comparison
6 {
7     // main method start de uitvoering van Java applicatie
8
9     public static void main(String[] args)
10    {
11        // creëer een object van Scanner; voor invoer vanaf het toetsenbord
12        Scanner input = new Scanner(System.in);
13
14        int number1;           // eerste getal om te vergelijken
15        int number2;           // tweede getal om te vergelijken
16
17        System.out.print("Enter first integer: "); // prompt
18        number1 = input.nextInt(); // leest eerste getal van de gebruiker
19
20        System.out.print("Enter second integer: "); // prompt
21        number2 = input.nextInt(); // leest tweede getal van de gebruiker
22
23        if (number1 == number2) ①
24        {
25            System.out.printf("%d == %d\n", number1, number2); ②
26        }
27
28        if (number1 != number2)
29        {
30            System.out.printf("%d != %d\n", number1, number2);
31        }
32
33        if (number1 < number2)
34        {
35            System.out.printf("%d < %d\n", number1, number2);
36        }
37    }
```



```

38     if (number1 > number2)
39     {
40         System.out.printf("%d > %d\n", number1, number2);
41     }
42
43     if (number1 <= number2)
44     {
45         System.out.printf("%d <= %d\n", number1, number2);
46     }
47
48     if (number1 >= number2)
49     {
50         System.out.printf("%d >= %d\n", number1, number2);
51     }
52
53 } // methode main
54 }

```

- bespreking

① if structuur om te testen op gelijkheid (==)

- Als de conditie waar (true) is dan wordt het statement **System.out.printf("%d == %d\n", number1, number2);** uitgevoerd.
- Als de conditie onwaar (false) is, dan wordt het statement overgeslagen.

② **System.out.printf("%d == %d\n", number1, number2);** In de formatstring is == vaste tekst en geen operator!

6.2. De "if-else" selectiestructuur

if (voorwaarde) statement1 else statement2

- Het statement1 wordt enkel uitgevoerd wanneer de voorwaarde WAAR (true) is.
- Het alternatieve statement2 wordt enkel uitgevoerd wanneer de voorwaarde VALS (false) is.

Voorbeeld 1

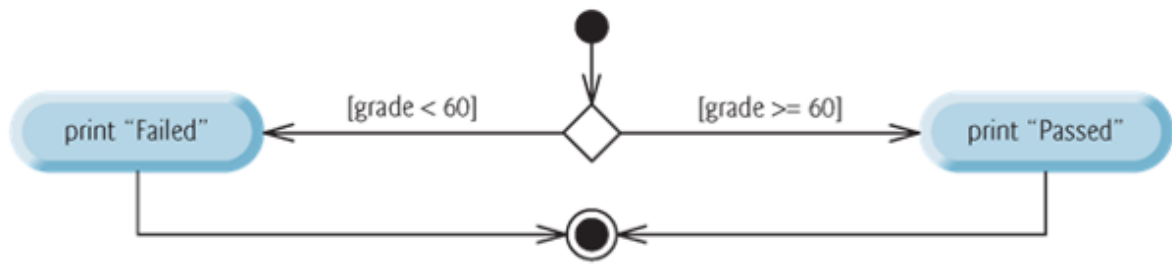
- in pseudocode

```

Als punten groter dan of gelijk aan 60
    dan print "Geslaagd!"
    anders print "Niet geslaagd!"
Eind-als

```

- activity diagram



Structuur met één ingang en één uitgang.

- in Java

```
if (punten >= 60)
    System.out.println("Geslaagd");
else
    System.out.println("Niet geslaagd");

// OF met accolades (niet verplicht als er maar 1 statement is)
if (punten >= 60)
{
    System.out.println("Geslaagd");
}
else
{
    System.out.println("Niet geslaagd");
}
```

Voorbeeld 2

- in pseudocode

```
Als punten gelijk is aan 20
dan
    print "Je behaalt een credit!"
    aantalGeslaagden <- aantalGeslaagden + 1
anders
    print "Je behaalt geen credit!"
    aantalNietGeslaagden <- aantalNietGeslaagden + 1
Eind-als
```

- in Java

```

if (punten == 20)
{
    System.out.println("Je behaalt een credit!");
    aantalGeslaagden = aantalGeslaagden + 1;
}
else
{
    System.out.println("Je behaalt geen credit!");
    aantalNietGeslaagden = aantalNietGeslaagden + 1;
}

```

Hier zijn de accolades noodzakelijk gezien er telkens 2 enkelvoudige statements zijn die samen horen.

6.3. Geneste "if-else" selectiestructuren

Voorbeeld 1: Drie getallen worden ingegeven. Het grootste getal wordt op het scherm weergegeven.

if (getal1 > getal2)

if (getal1 > getal3)

System.out.printf("%d",getal1);

else // getal1 <= getal3

System.out.printf("%d",getal3);

else // getal1 <= getal2

if (getal2 > getal3)

System.out.printf("%d",getal2);

else // getal2 <= getal3

System.out.printf("%d",getal3);



Noteer in commentaar naast de else de negatie van de voorwaarde; zo zie je beter in wat er eventueel nog moet getest worden!

Voorbeeld 2: De ingegeven code (1, 2, 3 of 4) wordt gecontroleerd. Je keuze komt op het scherm.

```

if (code == 1)
    System.out.println("Je kiest om op te tellen!");
else // code <> 1
if (code == 2)
    System.out.println("Je kiest om af te trekken!");
else // code <> 1 en code <> 2
if (code == 3)
    System.out.println("Je kiest om te delen!");
else // code <> 1 en code <> 2 en code <> 3
    System.out.println("Je kiest om te vermenigvuldigen!");

```

Voorbeeld 3: De grade wordt gecontroleerd. In de WAAR-tak hebben we 1 statement, in de VALS-tak 2 enkelvoudige statements die samen horen.

```

if (grade >= 60)
    System.out.println("Passed");
else
{
    System.out.println("Failed");
    System.out.println("You must take the course again.");
}

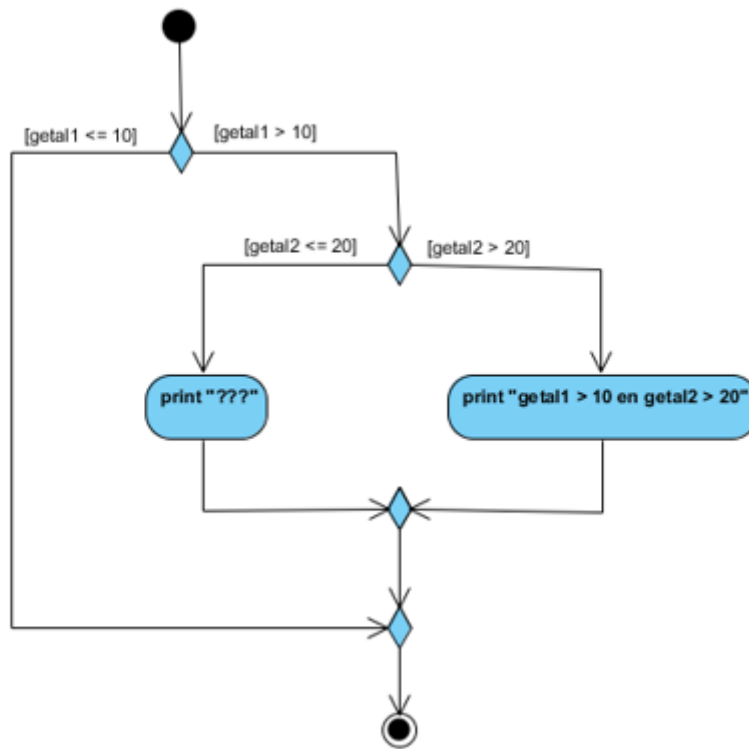
```

Voorbeeld 4: Hoort de else bij de eerste of bij de tweede if?

```

if (getal1 > 10)
    if (getal2 > 20)
        System.out.println("getal1 > 10 en getal2 > 20");
    else
        System.out.println(" ??? ");

```

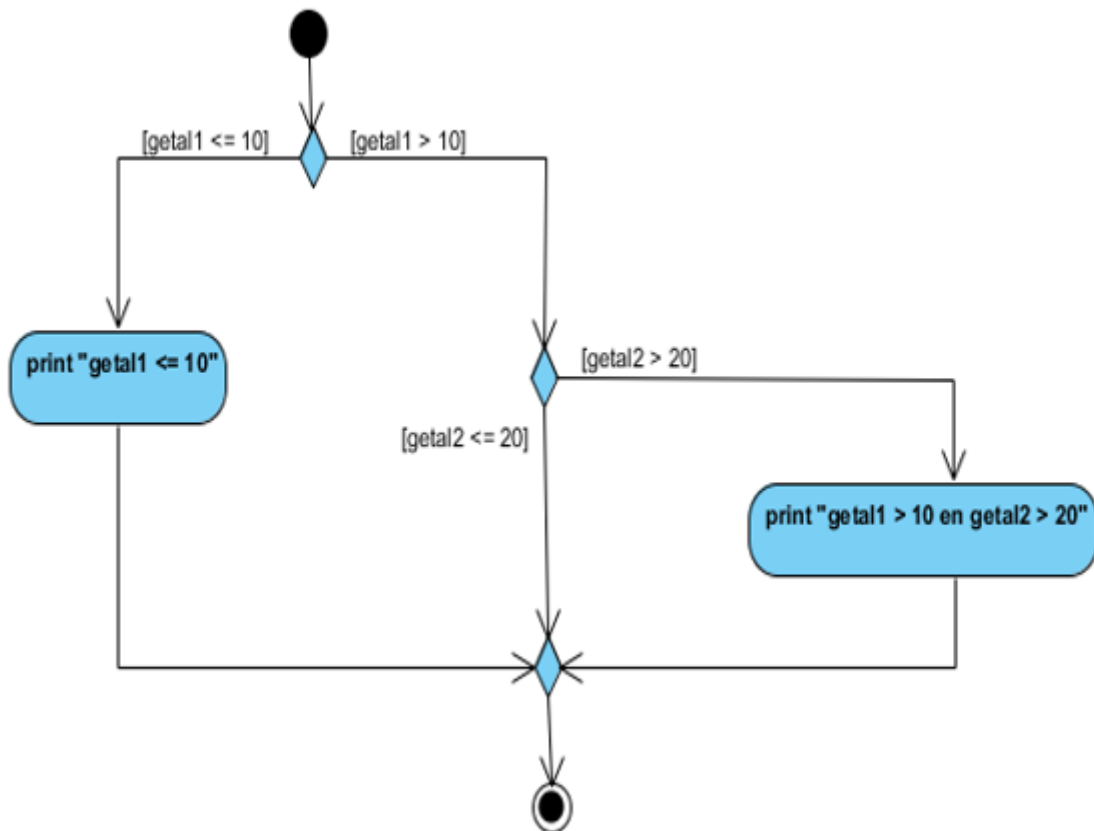


Zoals uit het diagram blijkt, hoort de else bij de tweede if of bij de dichtst bijzijnde if.

Voorbeeld 5: We wensen de else bij “if (getal1 > 10)” en niet bij “if (getal2 > 20)”!

```

if (getal1 > 10)
{
    if (getal2 > 20)
        System.out.println("getal1 > 10 en getal2 > 20");
    else
        System.out.println(" ??? ");
}
  
```



Gebruik van {} om aan te geven bij welke if de else hoort (zie voorbeeld 5) en om meerdere statements te groeperen in een blok.

6.4. De conditionele operator (?:)

De conditionele operator kan gebruikt worden om sommige if/else-structuren korter te schrijven. In de conditionele operator komt altijd een vraagteken (?) en een dubbel punt (:) voor.

voorwaarde ? expressie1 : expressie2

- Indien de voorwaarde waar is, wordt expressie1 uitgevoerd.
- Indien de voorwaarde niet waar is, wordt expressie2 uitgevoerd.

Voorbeeld 1:

```
if (a < b)
    z = a + 1;
else
    z = b - 1;
```

is equivalent met

```
z = a < b ? a + 1 : b - 1;
```

Voorbeeld 2:

```
if (a < b)
    System.out.printf("%d",a);
else
    System.out.printf("%d",b);
```

is equivalent met

```
System.out.printf("%d",a < b ? a : b);
```

Voorbeeld 3:

```
if (aantal == 1)
    System.out.printf("%d bal",aantal);
else
    System.out.printf("%d ballen",aantal);
```

is equivalent met

```
System.out.printf("%d %s",aantal, aantal == 1 ? "bal" : "ballen");
```

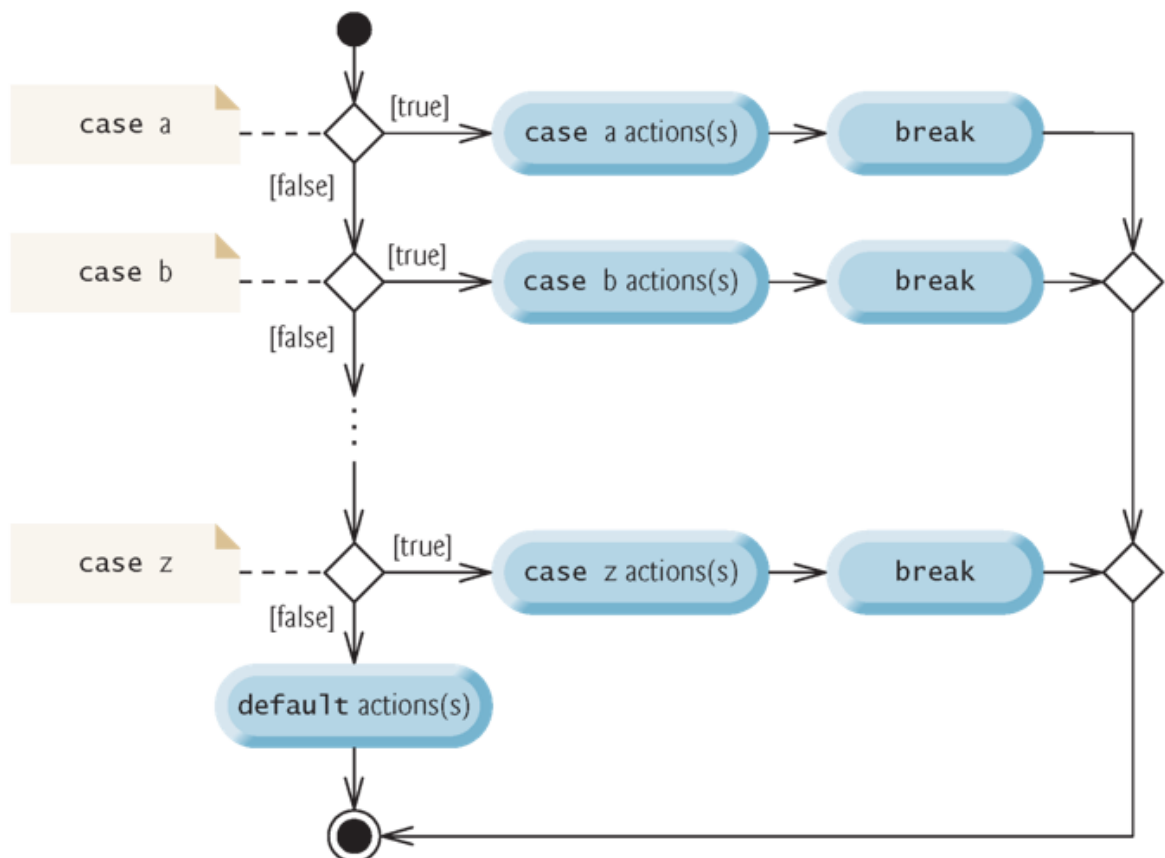


We hebben telkens in beide takken een gemeenschappelijk stukje code dat we vooraan kunnen plaatsen! Hebben we dat niet, dan kan deze operator niet gebruikt worden.

6.5. De "switch" meervoudige selectiestructuur

De switch-structuur wordt gebruikt voor meervoudige selecties:

```
switch (variabele)
{
case label1: statement1
    break;
case label2: statement2
    break;
default:    statement3
    [ break;]
}
```



Er zijn 5 mogelijke types voor de variabele en de labels:

- byte
- short
- int

- char
- String

Regels:

- Alle labels moeten verschillend zijn
- Labels van het type char staan tussen enkele quotes, labels van het type String tussen dubbele quotes.
- Het type van de labels is hetzelfde type als van de variabele
- De volgorde van de labels is willekeurig – default staat meestal als laatste
- De default-label is niet verplicht
- Elke case moet afgebroken worden met een break; indien geen break, dan worden alle statements na de corresponderende label uitgevoerd tot aan de eerste break of tot aan de sluit accolade van de switch

Voorbeeld 1 met int als type en break-statements:

```
System.out.print("Geef uw keuze in: ");
int keuze = input.nextInt();
switch (keuze)
{
case 1:
    System.out.println("Keuze 1!");
    break;
case 2:
    System.out.println("Keuze 2!");
    break;
case 3:
    System.out.println("Keuze 3!");
    break;
default:
    System.out.println("Foutieve keuze!");
    break;
}
```

Geef uw keuze in: 1
Keuze 1!

Geef uw keuze in: 4
Foutieve keuze!

Switch-structuur omgezet naar geneste if-else:

```

if (keuze == 1)
    System.out.println("Keuze 1!");
else if (keuze == 2)
    System.out.println("Keuze 2!");
else if (keuze == 3)
    System.out.println("Keuze 3!");
else
    System.out.println("Foutieve keuze!");

```



Als de meervoudige selectie controleert op gelijkheid, dan kunnen we een switch-structuur gebruiken.

Voorbeeld 2 met char als type en zonder break-statements:

```

String stad = "Gent";
switch (stad.charAt(0)) // eerste karakter van de String "stad"
{
    case 'A':
        System.out.println("Antwerpen");
    case 'G':
        System.out.println("Gent");
    case 'B':
        System.out.println("Brugge");
}

```

Gent
Brugge



De expressie bij switch levert het eerste karakter van de variabele stad op, zijnde 'G'. Er is m.a.w. een overeenkomst met de tweede label, maar gezien er geen break-statement staat, wordt de switch niet afgebroken.

Voorbeeld 3 met String als type en met break-statements:

```

package cui;

import java.util.Scanner;

public class SwitchVoorbeeld3
{
    public static void main(String[] args)
    {
        Scanner invoer = new Scanner(System.in);
        String maand ;

        System.out.print("Geef de naam van een maand: ");
        maand = invoer.next();

        int maandNummer;
        // we vormen de volledige naam om naar kleine letters!
        switch(maand.toLowerCase())
        {
            case "januari": maandNummer = 1; break;
            case "februari": maandNummer = 2; break;
            case "maart": maandNummer = 3; break;
            case "april": maandNummer = 4; break;
            case "mei": maandNummer = 5; break;
            case "juni": maandNummer = 6; break;
            case "juli": maandNummer = 7; break;
            case "augustus": maandNummer = 8; break;
            case "september": maandNummer = 9; break;
            case "oktober": maandNummer = 10; break;
            case "november": maandNummer = 11; break;
            case "december": maandNummer = 12; break;
            default: maandNummer = 0; break;
        }

        //afdruk verzorgen
        if (maandNummer != 0)
        {
            System.out.printf("Dit is de %d%s maand van het jaar%n", maandNummer,
                maandNummer==1 || maandNummer == 8? "-ste" : "-de");
        }
        else
            System.out.printf("De naam van de maand werd niet herkend.%n");
    }
}

```

Geef de naam van een maand: Januari
Dit is de 1-ste maand van het jaar

Geef de naam van een maand: **october**
De naam van de maand werd niet herkend.

- We vergelijken de String in de switch expressie met de case labels d.m.v. de methode equals uit de String-klasse.
- Het voorbeeld accepteert elke maand ongeacht hoofd- en/of kleine letters. Met de Stringmethode toLowerCase wordt de string “maand” omgezet naar kleine letters, gezien alle strings bij de case-labels in kleine letters staan.
- Komt de maand niet voor, dan wordt de code achter de default-label uitgevoerd.

7. De herhalingsstructuren

7.1. Inleiding

We kunnen 2 soorten lussen onderscheiden:

- een lus waarvan je op voorhand weet om hoeveel herhalingen het gaat, dit noemen we ook een tellergestuurde lus
- een lus waarvan je op voorhand NIET weet om hoeveel herhalingen het gaat

7.2. Tellergestuurde lussen

7.2.1. De while-lus met teller

while (voorwaarde) statement

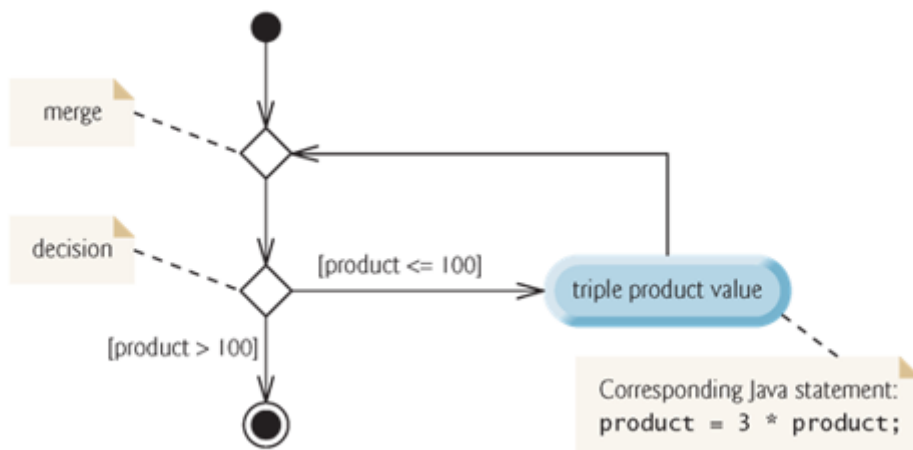
- Het statement wordt herhaald zolang de voorwaarde WAAR blijft!
- Zorg ervoor dat de voorwaarde beïnvloed wordt in het statement, anders oneindige lus!
- Als het statement enkelvoudig is, dan wordt de herhalingsstructuur afgesloten met het punt-komma in het statement. Als het statement samengesteld is, dan wordt de herhalingsstructuur afgesloten met de eind-accolade van het statement.
- Voorwaarden kunnen geformuleerd worden met relationele operatoren en/of gelijkheidsoperatoren
- De ronde haakjes rond de voorwaarde zijn verplicht!
- Voor een herhaling met een teller zijn nodig:
 - een controlevariabele (de teller), moet een gehele variabele zijn
 - initiële waarde van de controlevariabele
 - increment/decrement van de controlevariabele; wordt aangepast elke keer de lus doorlopen wordt
 - conditie, die test op de eindwaarde van de controlevariabele

Voorbeeld 1

- in pseudocode

```
product <- 3
Herhaal zolang product <= 100
  product <- product * 3
Eind-herhaal
```

- activity diagram



Structuur met één ingang en één uitgang.

- in Java

```
int product = 3;
while (product <= 100)
  product = product * 3;
```

Voorbeeld 2

- probleemstelling: Schrijf een programma dat de getallen 1 t.e.m. 5 na elkaar afdrukt.
- in pseudocode

Het aantal keer dat de lus wordt uitgevoerd is gekend, nl. 5. De variabele 'teller' houdt het aantal keer dat een reeks statements wordt uitgevoerd bij. Deze teller start op 1.

```
teller <- 1
Herhaal zolang teller <= 5
  Druk teller
  teller <- teller + 1
Eind-herhaal
```

- in Java

```
1 package cui;
2
3 public class WhileCounter
4 {
5     public static void main(String[] args)
6     {
7         int teller = 1; // declaratie en initialisatie ①
8
9         while (teller <= 5) // voorwaarde van de herhaling ②
10        {
11            System.out.printf("%d ", teller); ③
12            teller = teller + 1; // increment ④
13        } // einde while structuur
14    }
15 }
```

- uitvoer

<terminated> WhileCounter

1 2 3 4 5

- bespreking

- ① Naam van de controlevariabele is **teller**. Initiële waarde van de controlevariabele is 1.
- ② Conditie, die test op de eindwaarde van de controlevariabele **teller**. De herhaling stopt wanneer de controlevariabele groter is dan 5.
- ③ De waarde van **teller** wordt afgedrukt op de console gevolgd door een spatie.
- ④ Increment van de controlevariabele **teller**; wordt aangepast elke keer de lus doorlopen wordt.

Voorbeeld 3

- probleemstelling: Vraag aan 10 studenten hun punten. Schrijf de totale som en het gemiddelde uit.
- in pseudocode

Het aantal keer dat de lus wordt uitgevoerd is gekend, nl. 10. De variabele 'puntenTeller' houdt het aantal keer dat een reeks statements wordt uitgevoerd bij. Deze teller start op 1.

De variabele 'totaal' houdt de som van alle punten bij en moet initieel op 0 gezet worden!

```
totaal <- 0
puntenTeller <- 1
Herhaal zolang puntenTeller <= 10
  lees punt in
  totaal <- totaal + punt
  puntenteller <- puntenTeller + 1
Eind-herhaal
klasGemiddelde <- totaal / 10
Druk totaal
Druk klasGemiddelde
```

- in Java

```

1 package cui;
2
3 import java.util.Scanner;
4
5 /**
6  * Dit is een voorbeeld van een herhaling met teller uitgewerkt met een while!
7  */
8 public class KlasgemiddeldeTeller
9 {
10     public static void main(String args[])
11     {
12         //declaratie lokale variabelen
13         int totaal, puntenTeller, punt, klasGemiddelde;
14         Scanner input = new Scanner(System.in);
15
16         //initialisatie variabelen
17         totaal = 0;
18         puntenTeller = 1; ①
19
20         //blijf herhalen tot...
21         while (puntenTeller <= 10) ②
22         {
23             // invoer
24             System.out.print("Geef score " + puntenTeller + " (/20):");
25             punt = input.nextInt();
26
27             totaal = totaal + punt;
28             puntenTeller = puntenTeller + 1; ③
29         }
30
31         klasGemiddelde = totaal / 10; ④
32
33         // uitvoer
34         System.out.printf("Het totaal is %d\n", totaal);
35         System.out.printf("Het klasgemiddelde is %d\n", klasGemiddelde);
36     }
37 }

```

- uitvoer


```
Geef score 1 (/20):12
Geef score 2 (/20):6
Geef score 3 (/20):9
Geef score 4 (/20):16
Geef score 5 (/20):18
Geef score 6 (/20):5
Geef score 7 (/20):3
Geef score 8 (/20):13
Geef score 9 (/20):20
Geef score 10 (/20):10
Het totaal is 112
Het klasgemiddelde is 11
```

- bespreking

- ① Initiële waarde van de controlevariabele **puntenTeller**.
- ② Conditie, die test op de eindwaarde van de controlevariabele **puntenTeller**.
- ③ Increment van de controlevariabele **puntenTeller**; wordt aangepast elke keer de lus doorlopen wordt.
- ④ Gehele deling want teller en noemer zijn geheel.

7.2.2. De for-lus

for (expressie1; expressie2; expressie3) statement

Een for-lus hanteert alle details van de herhalingsstructuur met teller!

We hernemen de 3 voorbeelden uit de vorige paragraaf en werken ze nu uit met een for-lus:

Voorbeeld 1

- in pseudocode

```
product <- 3
Herhaal zolang product <= 100
  product <- product * 3
Eind-herhaal
```

- in Java

```
for (int product = 3; product <= 100; product = product * 3)
;
```

Initialisatie, herhalingsconditie en incrementeren zijn allemaal inbegrepen in de hoofding van de for-structuur. Bijgevolg krijgen we in dit voorbeeld een leeg statement in de lus. De variabele 'product' is ook gedeclareerd in het controlegedeelte van de for-lus. Bijgevolg kan deze variabele **ALLEEN** in de lus gebruikt worden!

Voorbeeld 2

- probleemstelling: Schrijf een programma dat de getallen 1 t.e.m. 5 na elkaar afdruckt.
- in pseudocode

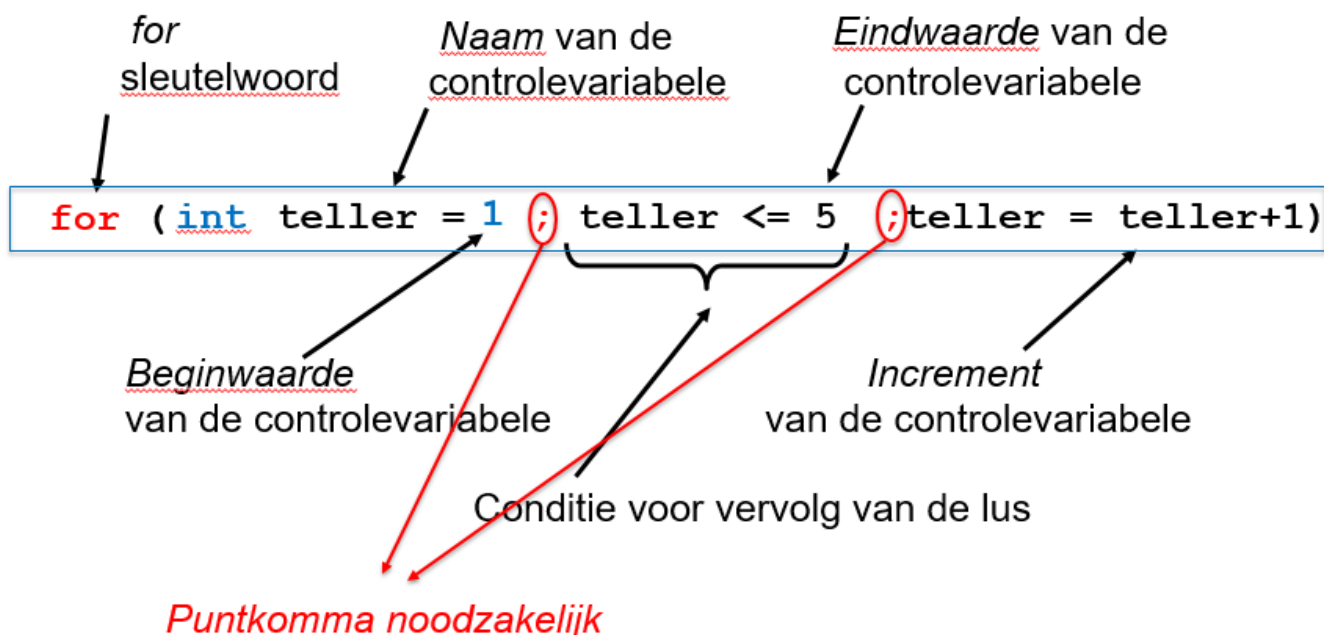
Het aantal keer dat de lus wordt uitgevoerd is gekend, nl. 5. De variabele 'teller' houdt het aantal keer dat een reeks statements wordt uitgevoerd bij. Deze teller start op 1.

```
teller <- 1
Herhaal zolang teller <= 5
  Druk teller
  teller <- teller + 1
Eind-herhaal
```

- in Java

```
1 for (int teller = 1; teller <= 5; teller = teller + 1)
2   System.out.printf("%d ", teller);
```

Componenten van een typische hoofding van een “for” structuur:



- uitvoer

<terminated> ForCounter

1 2 3 4 5

Voorbeeld 3

- probleemstelling: Vraag aan 10 studenten hun punten. Schrijf de totale som en het gemiddelde uit.
- in Java

```

1 package cui;
2
3 import java.util.Scanner;
4
5 /**
6  * Dit is een voorbeeld van een herhaling met teller uitgewerkt met een for!
7  *
8  */
9 public class KlasgemiddeldeTellerFOR
10 {
11     public static void main(String args[])
12     {
13         //declaratie lokale variabelen
14         int totaal, punt, klasGemiddelde;
15         Scanner input = new Scanner(System.in);
16
17         //initialisatie variabelen
18         totaal = 0;
19
20         //blijf herhalen tot...
21         for (int puntenTeller = 1;puntenTeller <= 10;puntenTeller = puntenTeller +
1)
22         {
23             System.out.print("Geef score " + puntenTeller + " (/20):");
24             punt = input.nextInt();
25             totaal = totaal + punt;
26         }
27
28         klasGemiddelde = totaal / 10;
29
30         // uitvoer
31         System.out.printf("Het totaal is %d\n", totaal);
32         System.out.printf("Het klasgemiddelde is %d\n", klasGemiddelde);
33     }
34 }

```

- uitvoer

```
Geef score 1 (/20):12
Geef score 2 (/20):6
Geef score 3 (/20):9
Geef score 4 (/20):16
Geef score 5 (/20):18
Geef score 6 (/20):5
Geef score 7 (/20):3
Geef score 8 (/20):13
Geef score 9 (/20):20
Geef score 10 (/20):10
Het totaal is 112
Het klasgemiddelde is 11
```

Voorbeeld 4

- probleemstelling: Vraag aan 10 studenten of ze geslaagd zijn. Maak een analyse van de resultaten behaald door de studenten (1 = geslaagd, 2 = niet-geslaagd).
- in Java

```

1 package cui;
2 // Analyse van resultaten gebruikmakend van geneste controlestructuren.
3
4 import java.util.Scanner;
5
6 public class Analysis1
7 {
8     public static void main(String[] args)
9     {
10         Scanner input = new Scanner(System.in);
11
12         // initialisatie van variabelen bij declaratie
13         int passes = 0;
14         int failures = 0;
15         int studentCounter = 1;
16         int result;
17
18         // verwerk 10 studenten gebruikmakend van een tellergestuurde while-lus
19         while (studentCounter <= 10)
20         {
21             System.out.printf("Enter result %d (1 = pass, 2 = fail): ",
studentCounter);
22             result = input.nextInt();
23
24             // if...else is genest in het while statement
25             if (result == 1)
26                 passes = passes + 1;
27             else
28                 failures = failures + 1;
29
30             // verhoog studentCounter
31             studentCounter = studentCounter + 1;
32         }
33
34         System.out.printf("Passed: %d\nFailed: %d\n", passes, failures);
35
36         // als er meer dan 8 studenten geslaagd zijn:
37         if (passes > 8)
38             System.out.println("Bonus to instructor!");
39     }
40 }

```

- uitvoer1

<terminated> Analysis [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (12

```
Enter result 1 (1 = pass, 2 = fail): 1
Enter result 2 (1 = pass, 2 = fail): 2
Enter result 3 (1 = pass, 2 = fail): 2
Enter result 4 (1 = pass, 2 = fail): 1
Enter result 5 (1 = pass, 2 = fail): 2
Enter result 6 (1 = pass, 2 = fail): 2
Enter result 7 (1 = pass, 2 = fail): 1
Enter result 8 (1 = pass, 2 = fail): 1
Enter result 9 (1 = pass, 2 = fail): 2
Enter result 10 (1 = pass, 2 = fail): 2
Passed: 4
Failed: 6
```

- uitvoer2

<terminated> Analysis [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (12

```
Enter result 1 (1 = pass, 2 = fail): 1
Enter result 2 (1 = pass, 2 = fail): 1
Enter result 3 (1 = pass, 2 = fail): 1
Enter result 4 (1 = pass, 2 = fail): 1
Enter result 5 (1 = pass, 2 = fail): 1
Enter result 6 (1 = pass, 2 = fail): 2
Enter result 7 (1 = pass, 2 = fail): 1
Enter result 8 (1 = pass, 2 = fail): 1
Enter result 9 (1 = pass, 2 = fail): 1
Enter result 10 (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

Voorbeeld 5

- probleemstelling: Schrijf een programma dat de samengestelde interest berekent:

$$a = p(1 + r)^n$$

p = kapitaal (in ons vb. 1000)

r = jaarlijkse interest (in ons vb. 5%)

n = aantal jaren

a = kapitaal na n jaar

Het kapitaal MET de samengestelde interest wordt voor de eerste 10 jaar weergegeven.

- in Java

```

1 package cui;
2
3 public class Interest
4 {
5     public static void main(String[] args)
6     {
7         double amount; // saldo op het einde van elk jaar ①
8         double principal = 1000.0; // beginkapitaal
9         double rate = 0.05; // rente
10
11         // hoofdingen: veldbreedte 4 en veldbreedte 20
12         System.out.printf("%s%20s\n", "Year", "Amount on deposit");
13
14         // bereken het saldo voor elk jaar
15         for (int year = 1; year <= 10; year++)
16         {
17             amount = principal * Math.pow(1.0 + rate, year); ②
18
19             // toon het jaar en het saldo terug op veldbreedte 4 en 20
20             System.out.printf("%4d%,20.2f\n", year, amount); ③
21         } // end for
22
23     } // end main
24 } // end class Interest

```

- uitvoer

Year	Amount on deposit
1	1.050,00
2	1.102,50
3	1.157,63
4	1.215,51
5	1.276,28
6	1.340,10
7	1.407,10
8	1.477,46
9	1.551,33
10	1.628,89

- bespreking

① Java voorziet 2 primitieve datatypes voor het stockeren van kommagetallen (getallen met decimalen):

- float: **enkelvoudige** precisie floating-point getallen (6 tot 7 decimalen nauwkeurig)
- double: **dubbele precisie** floating-point getallen (15 decimalen nauwkeurig)
- Een double neemt meer plaats in het geheugen dan een float. Zie ook 13.7!
- Andere notatie: **double amount, principal = 1000.0, rate = 0.05;**

② Machtsverheffing

java.lang

Class Math

java.lang.Object
java.lang.Math

Class Math

```
public final class Math  
extends Object
```

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

static double

pow(double a, double b)

Returns the value of the first argument raised to the power of the second argument.

static: klassemethode, geen object nodig, te gebruiken via de naam van de klasse

Voorbeeld:

2^6  Math.pow(2,6)

③ formatteren van floating-point getallen: %f gebruiken zowel voor het type float als double

- Tussen het procentteken (%) en het conversiekarakter **f** kan een punt en een getal staan.

- Het punt en getal geeft de precisie aan, nl. het aantal cijfers dat na het decimale teken wordt afgedrukt.
- Hier **%2f**, dus steeds 2 cijfers na de komma zichtbaar.
- Wordt hier gecombineerd met een veldbreedte: **%20.2f** betekent een totale veldbreedte van 20 posities waarvan er eentje gebruikt wordt voor het decimale teken en 2 voor de cijfers na de komma. Voor de komma hebben we bijgevolg nog 17 posities over.
- Wordt hier gecombineerd met een komma vlag: **%,20.2f** betekent dat er per 1000-tal een punt staat: **1.407,10**

Variërende controlevariabele in de for structuur

- De controlevariabele varieert van 1 tot 100, toenemend in stappen van 2

```
for (int i = 1; i <= 100; i+=2 ) {}
```

- De controlevariabele varieert van 100 tot 1, afnemend in stappen van -1

```
for (int i = 100; i >= 1; i-- ) {}
```

- De controlevariabele varieert van 7 tot 77 in stappen van 7

```
for (int i = 7; i <= 77; i += 7 ) {}
```

Voorbeeld

- probleemstelling: Schrijf een applicatie die de som van de even gehele getallen van 2 t.e.m. 20 weergeeft op het scherm.
- in Java:

```
1 package cui;
2
3 public class Sum
4 {
5     public static void main(String[] args)
6     {
7         int total = 0; // initialize total
8
9         // total even integers from 2 through 20
10        for (int number = 2; number <= 20; number += 2)
11            total += number;
12
13        System.out.printf("Sum is %d\n", total); // display results
14    } // end main
15 }
```

- uitvoer

<terminated> Sum [Java Application]

Sum is 110

7.3. De andere lussen

7.3.1. De while-lus met schildwacht

while (voorwaarde) statement

- Het statement wordt herhaald zolang de voorwaarde WAAR blijft!
- Zorg ervoor dat de voorwaarde beïnvloed wordt in het statement, anders oneindige lus!
- Als het statement enkelvoudig is, dan wordt de herhalingsstructuur afgesloten met het punt-komma in het statement. Als het statement samengesteld is, dan wordt de herhalingsstructuur afgesloten met de eind-accolade van het statement.
- Voorwaarden kunnen geformuleerd worden met relationele operatoren en/of gelijkheidsoperatoren.
- De ronde haakjes rond de voorwaarde zijn verplicht!
- Voor een herhaling met een schildwacht zijn nodig:
 - een waarde (schildwacht, sentinel) die het einde van de invoer van data aangeeft
 - de schildwacht wordt niet meer verwerkt!

Voorbeeld 1

- probleemstelling: Vraag aan een aantal studenten hun punten en stop met -1. Schrijf het gemiddelde uit.
- in pseudocode

Het aantal keer dat de lus wordt uitgevoerd is NIET gekend. De waarde -1 zal gebruikt worden om de lus te stoppen. De variabele 'puntenTeller' houdt het aantal punten bij. Deze teller start op 0. De variabele 'totaal' houdt de som van alle punten bij en moet ook initieel op 0 gezet worden!

```

totaal <- 0
puntenTeller <- 0
lees (het eerste) punt in (mogelijk de "schildwacht", dus invoeren juist VOOR de
controle)
Herhaal zolang punt <> -1
  totaal <- totaal + punt
  puntenTeller <- puntenTeller + 1
  lees (volgende) punt in (mogelijk de "schildwacht", dus invoeren juist VOOR we
teruggaan naar de controle)
Eind-herhaal
Als puntenTeller <> 0
  klasGemiddelde <- totaal / puntenTeller
  Druk klasGemiddelde
anders
  Druk "Er werden geen punten ingevoerd!"
Eind-als

```

- in Java

```

1 package cui;
2
3 import java.util.Scanner;
4
5 public class KlasgemiddeldeSchildwacht
6 {
7     public static void main(String args[])
8     {
9         // declaratie lokale variabelen
10        int totaal, puntenTeller, punt;
11        double gemiddelde;
12        Scanner input = new Scanner(System.in);
13
14        // initialisatie variabelen
15        totaal = 0;
16        puntenTeller = 0;
17
18        // verwerkingsfase
19        System.out.print("Geef score (/20) of -1 om te stoppen: ");
20        punt = input.nextInt();①
21
22        while (punt != -1) ②
23        {
24            totaal = totaal + punt;
25            puntenTeller = puntenTeller + 1;
26            System.out.print("Geef score (/20) of -1 om te stoppen: ");
27            punt = input.nextInt();③
28        }
29
30        if (puntenTeller != 0) ④
31        {
32            gemiddelde = (double) totaal / puntenTeller; ⑤
33            System.out.printf("Het klasgemiddelde is %.2f\n", gemiddelde); ⑥
34        } else
35            System.out.printf("Er werden geen punten ingegeven.\n");
36
37    }
38 }

```

- uitvoer 1 (invoer van 5 punten)

<terminated> KlasgemiddeldeSchildwacht [Java Application] C:\Program Files\Java\jdk-11.0.6\l

```
Geef score (/20) of -1 om te stoppen: 13
Geef score (/20) of -1 om te stoppen: 15
Geef score (/20) of -1 om te stoppen: 11
Geef score (/20) of -1 om te stoppen: 6
Geef score (/20) of -1 om te stoppen: 8
Geef score (/20) of -1 om te stoppen: -1
Het klasgemiddelde is 10,60
```

- uitvoer 2 (geen invoer van punten)

<terminated> KlasgemiddeldeSchildwacht [Java Application] C:\Program Files\Java\jdk-11.0.6\l

```
Geef score (/20) of -1 om te stoppen: -1
Er werden geen punten ingegeven.
```

- bespreking

- ① Invoer van de eerste punten **VOOR** de lus.
- ② **Voorwaarde:** zolang we de schildwacht (-1) niet invoeren wordt de lus herhaald
- ③ Invoer van de volgende punten op het **EINDE** in de lus, dus juist voor we teruggaan naar de controle.
- ④ We gebruiken een selectie om delen door 0 te vermijden.
- ⑤ teller en noemer zijn geheel; om een nauwkeurige deling te verkrijgen passen we een cast-operator toe:
 - (double) is een expliciete cast-operator om totaal als double te behandelen
 - we hebben hierdoor geen gehele deling maar een reële deling
 - puntenTeller wordt door Java gepromoot naar double, zodat teller en noemer terug hetzelfde type hebben
 - **Merk op:** (double)(totaal/puntenTeller) levert geen nauwkeurig resultaat op gezien eerst de gehele deling wordt uitgevoerd en pas daarna wordt gecast

Voorbeeld: `(double)(22/5) = 4.0`

- ⑥ formatteren van floating-point getallen met 2 cijfers na de komma: `%.2f` gebruiken
 - Tussen het procentteken (%) en het conversiekarakter **f** kan een punt en een getal staan.
 - Het punt en getal geeft de precisie aan, nl. het aantal cijfers dat na het decimale teken wordt afgedrukt.

- Hier %.2f, dus steeds 2 cijfers na de komma zichtbaar.

Voorbeeld 2

- probleemstelling: Vraag aan een aantal studenten of ze geslaagd zijn en stop met 0. Maak een analyse van de resultaten behaald door de studenten (1 = geslaagd, 2 = niet-geslaagd).
- in Java

```
1 package cui;
2 // Analyse van resultaten gebruikmakend van geneste controlestructuren.
3
4 import java.util.Scanner;
5
6 public class Analysis2
7 {
8     public static void main(String[] args)
9     {
10         Scanner input = new Scanner(System.in);
11
12         // initialisatie van variabelen bij declaratie
13         int passes = 0;
14         int failures = 0;
15         int studentCounter = 1; // wordt hier gebruikt in de vraagstelling
16         int result;
17
18         // verwerk een aantal studenten gebruikmakend van een while-lus met
19         // schildwacht
20         System.out.printf("Enter result %d (0 = stop, 1 = pass, 2 = fail): ",
21         studentCounter);
22         result = input.nextInt();
23         while (result != 0)
24         {
25             // if...else is genest in het while statement
26             if (result == 1)
27                 passes = passes + 1;
28             else
29                 failures = failures + 1;
30
31             // verhoog studentCounter
32             studentCounter = studentCounter + 1;
33
34             System.out.printf("Enter result %d (0 = stop, 1 = pass, 2 = fail): ",
35             studentCounter);
36             result = input.nextInt();
37         }
38
39         System.out.printf("Passed: %d\nFailed: %d\n", passes, failures);
40     }
41 }
```

- uitvoer

```
<terminated> Analysis2 [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (12 jun. 2020 11:09:39)
Enter result 1 (0 = stop, 1 = pass, 2 = fail): 1
Enter result 2 (0 = stop, 1 = pass, 2 = fail): 2
Enter result 3 (0 = stop, 1 = pass, 2 = fail): 2
Enter result 4 (0 = stop, 1 = pass, 2 = fail): 1
Enter result 5 (0 = stop, 1 = pass, 2 = fail): 0
Passed: 2
Failed: 2
```

7.3.2. De do-while-lus

do statement while (voorwaarde);

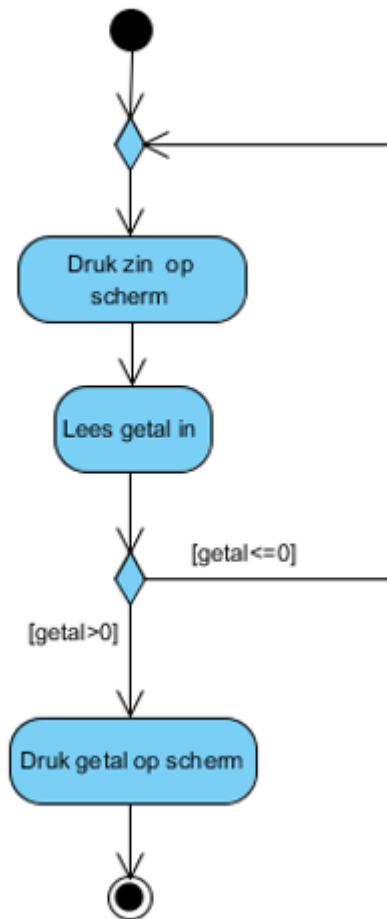
- Lijkt op de while structuur, MAAR test of de lus verder doorlopen moet worden **NADAT** de body van de loop uitgevoerd is.
- De lus wordt altijd **MINSTENS EENMAAL** doorlopen, gezien de test **NA** het statement staat.
- Het statement wordt herhaald zolang de voorwaarde **WAAR** blijft!
- Zorg ervoor dat de voorwaarde beïnvloed wordt in het statement, anders oneindige lus!
- Als het statement enkelvoudig is, dan zijn de accolades overbodig. Als het statement samengesteld is, dan zijn de accolades noodzakelijk.
- Deze herhalingsstructuur wordt **ALTIJD** afgesloten met een punt-komma.
- Voorwaarden kunnen geformuleerd worden met relationele operatoren en/of gelijkheidsoperatoren.
- De ronde haakjes rond de voorwaarde zijn verplicht!

Voorbeeld

- probleemstelling: Er wordt gevraagd om een strikt positief geheel getal in te voeren; de invoer wordt vervolgens gecontroleerd; indien deze niet voldoet, wordt een nieuwe waarde aan de gebruiker gevraagd. Is de invoer in orde, dan wordt de invoer getoond.
- in pseudocode

```
Herhaal
    druk zin op het scherm
    lees getal in
Zolang getal <= 0
    druk getal op scherm
```

- activity diagram



Structuur met één ingang en één uitgang.

- in Java

```

package cui;

import java.util.Scanner;

public class DoWhileControle1
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int getal;

        do // INVOERCONTROLE ①
        { ②
            System.out.print("Geef een strikt positief geheel getal in: ");
            getal = input.nextInt();
        } while (getal <= 0); // einde do/while structuur ③

        System.out.printf("Het ingevoerde getal = %d", getal);
    }
}
  
```


- uitvoer

<terminated> DoWhileControle1 [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (17 jun. 2020 17:49:56)

```
Geef een strikt positief geheel getal in: -6
Geef een strikt positief geheel getal in: 0
Geef een strikt positief geheel getal in: 234
Het ingevoerde getal = 234
```

- bespreking

- ① Startpunt van de lus.
- ② Het statement is samengesteld, dus de accolades zijn noodzakelijk.
- ③ Het getal moet strikt positief zijn, dus zolang het niet strikt positief is OF zolang het nul of negatief is, herhalen we de body van de lus. Is de invoer strikt positief dan is de voorwaarde vals en verlaten we de lus. Andere notatie: `do { ... } while (!(getal > 0));`



De lus wordt niet verlaten, zolang het ingevoerde getal niet voldoet.

7.4. Samenvatting herhalingsstructuren

- Je gebruikt een ‘for’ of een ‘while’ indien je van tevoren weet om hoeveel herhalingen het gaat.
- Weet je niet van tevoren om hoeveel herhalingen het gaat, dan dien je een ‘while’ of een ‘do-while’ te gebruiken.
 - Je kiest voor een ‘while’ indien de body van de while misschien nooit mag worden uitgevoerd. De ‘voorwaarde’ wordt eerst getest. De body van de while wordt uitgevoerd zolang de ‘voorwaarde’ waar is.
 - Je kiest voor een ‘do-while’ indien de body van de ‘do-while’ tenminste één keer moet worden uitgevoerd. Eerst wordt de body van de ‘do-while’ uitgevoerd, vervolgens wordt de voorwaarde gecontroleerd. Zolang de expressie waar is, wordt de body van de ‘do-while’ uitgevoerd.



De body van de while wordt 0, 1 of meerdere keren uitgevoerd.



De body van de do-while wordt minstens 1 keer uitgevoerd.

8. Samengestelde toekenningsoperatoren

In Java kunnen we in bepaalde situaties uitdrukkingen voor toekenningen afkorten:

Heel vaak willen we iets bij een variabele optellen:

```
int teller = 1;  
teller = teller + 1;  
teller = teller + 3;
```

In JAVA bestaat er een kortere en meer efficiënte vorm:

```
teller += 1;  
teller += 3;
```

Algemeen:

x = x operator a
kunnen we schrijven als
x operator= a;



x = x + a; kunnen we schrijven als **x += a;**

De volgende expressies spreken nu voor zichzelf:

```
x -= a;
```

```
x *= a;
```

```
x /= a;
```

```
x %= a;
```

Toekennings operator	Voorbeeld uitdrukking	Uitleg	Kent toe
<i>Stel:</i> <code>int c = 3, d = 5, e = 4, f = 6, g = 12;</code>			
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>	10 aan c
<code>-=</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1 aan d
<code>*=</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20 aan e
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2 aan f
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3 aan g

9. Increment- en decrementoperatoren

In het bijzonder komt het vaak voor, dat een telvariabele (bijvoorbeeld teller) met 1 moet verhoogd of verlaagd worden.

In plaats van

```
teller += 1;
teller -= 1;
```

kunnen we dit nog korter schrijven:

```
teller++;
teller--;
```

of

```
++teller;
--teller;
```

Unaire increment operator (++): **vermeerdert** de waarde van de variabele met 1

Unaire decrement operator (--): **vermindert** de waarde van de variabele met 1

Wat is nu het verschil tussen `teller++` en `++teller`?

```

int teller1, resul1, teller2, resul2;
teller1 = 0;
teller2 = 0;
resul1 = teller1++; ①
resul2 = ++teller2; ②
System.out.printf("%s : %d\n", "teller1", teller1);
System.out.printf("%s : %d\n", " resul1", resul1);
System.out.printf("%s : %d\n", "teller2", teller2);
System.out.printf("%s : %d\n", " resul2", resul2);

```

- uitvoer

<terminated> IncrementDecrement

```

teller1 : 1
  resul1 : 0
teller2 : 1
  resul2 : 1

```

- bespreking

- ① Het statement “resul1 = teller1++;” betekent, dat eerst de waarde van teller1 (=0) wordt toegekend aan de variabele resul1. Daarna wordt de waarde van variabele “teller1” met één verhoogd.

```
resul1 = teller1++; OF resul1 = teller1; teller1++; OF resul1 = teller1; ++teller1;
```

- ② Het statement “resul2 = ++teller2;” betekent, dat eerst de waarde van de variabele “teller2” met één wordt verhoogd (dus teller2 = 0 +1). Vervolgens wordt de waarde van teller2 (=1) toegekend aan de variabele resul2.

```
resul2 = ++teller2; OF ++teller2; resul2 = teller2; OF teller2++; resul2 = teller2;
```

Operator	Wordt genoemd	Voorbeeldexpressie	Uitleg
++	preincrement	++a	Vermeerder a met 1 en gebruik vervolgens de nieuwe waarde van a in de expressie waarin a voorkomt.
++	postincrement	a++	Gebruik de huidige waarde van a in de expressie waarin a voorkomt en vermeerder a vervolgens met 1.
--	predecrement	--b	Verminder b met 1 en gebruik vervolgens de nieuwe waarde van b in de expressie waarin b voorkomt.
--	postdecrement	b--	Gebruik de huidige waarde van b in de expressie waarin b voorkomt en verminder b vervolgens met 1.

Voorbeeld

```
package cui;

public class Increment
{
    public static void main(String[] args)
    {
        int c;

        c = 5;
        System.out.printf("%d\n", c); // print 5
        System.out.printf("%d\n", c++); // print 5 en vermeerder c dan met 1 ①
        System.out.printf("%d\n\n", c); // print 6

        c = 5;
        System.out.printf("%d\n", c); // print 5
        System.out.printf("%d\n", ++c); // vermeerder c eerst met 1, print 6 ②
        System.out.printf("%d\n", c); // print 6
    }
}
```

- uitvoer

<terminated> Increment

5
5
6

5
6
6

- bespreking

- ① Voert een postincrement uit op c, dus de waarde van c wordt pas aangepast **NA** deze uitvoer.
- ② Voert een preincrement uit op c, dus de waarde van c wordt aangepast **VOOR** deze uitvoer.

10. Logische operatoren

- Logische operatoren combineren eenvoudige condities tot complexere condities.
- Logische operatoren in Java

```
&&    (conditionele EN)
&     (logische EN)
||    (conditionele OF)
|     (logische inclusieve OF)
^     (logische exclusieve OF)
!     (logische NIET)
```

10.1. Waarheidstabel conditionele en (&&)

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Enkel WAAR als beide delen WAAR zijn!

10.2. Waarheidstabel conditionele of (| |)

expression1	expression2	expression1 expression2
false	false	false
false	true	true
true	false	true
true	true	true

Enkel VALS als beide delen VALS zijn!

10.3. Short-circuit operatoren

De logische AND (&) en de logische inclusieve OR (|) werken op dezelfde manier als de conditionele AND (&&) en de conditionele OF (| |), mits één uitzondering: het zijn **GEEN** short-circuit-operatoren



De logische operatoren & en | evalueren steeds de beide operanden!

Voorbeelden

```
geslacht == 1 & leeftijd >= 65
```

```
verjaardag == true | ++leeftijd >= 65 (de leeftijd wordt eerst verhoogd en daarna  
vergeleken met 65)
```

Een expressie met een conditionele AND (&&) of een conditionele OF (| |) wordt geëvalueerd totdat geweten is of de volledige expressie true of false oplevert.

Gevolgen

- Plaats bij de operator && de conditie die de meeste kans heeft om vals te zijn links van de operator, ook bij de operator | | de conditie die de meeste kans heeft om waar te zijn links van de operator plaatsen.
- Plaats de conditie die eventueel niet mag geëvalueerd worden rechts.

Voorbeelden

```
n != 0 && q < 1.0 / n  
Als n = 0 dan is n != 0 vals => volledig vals en GEEN deling door nul
```

```
n == 0 || q > 1.0 / n
```

Als $n = 0$ dan is $n == 0$ true => volledig true en GEEN deling door nul

```
verjaardag == true || ++leeftijd >= 65
```

Als $verjaardag = true$ dan is $conditie1$ true => volledig true, MAAR geen aanpassing van de leeftijd!

10.4. Waarheidstabel logische exclusieve of (^)

expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

Beide operanden worden steeds geëvalueerd. Deze operator heeft geen short-circuit evaluatie.

Voorbeeld

Als x negatief is, moet y positief zijn of omgekeerd, als y negatief is, moet x positief zijn.

```
if (x < 0 ^ y < 0) // (x < 0 en y >= 0) of (y < 0 en x >= 0)
```

10.5. Waarheidstabel logische niet (!)

expression	! expression
false	true
true	false

Samenvattend voorbeeld in Java: alle tabellen worden op het scherm gezet


```

package cui;

public class LogicalOperators
{
    public static void main(String[] args)
    {
        // create truth table for && (conditional AND) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n", ①
            "Conditional AND (&&)", "false && false", (false && false),
            "false && true", (false && true),
            "true && false", (true && false),
            "true && true", (true && true));

        // create truth table for || (conditional OR) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
            "Conditional OR (||)", "false || false", (false || false),
            "false || true", (false || true),
            "true || false", (true || false),
            "true || true", (true || true));

        // create truth table for & (boolean logical AND) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
            "Boolean logical AND (&)", "false & false", (false & false),
            "false & true", (false & true),
            "true & false", (true & false),
            "true & true", (true & true));

        // create truth table for | (boolean logical inclusive OR) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
            "Boolean logical inclusive OR (|)",
            "false | false", (false | false),
            "false | true", (false | true),
            "true | false", (true | false),
            "true | true", (true | true));

        // create truth table for ^ (boolean logical exclusive OR) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
            "Boolean logical exclusive OR (^)",
            "false ^ false", (false ^ false),
            "false ^ true", (false ^ true),
            "true ^ false", (true ^ false),
            "true ^ true", (true ^ true));

        // create truth table for ! (logical negation) operator
        System.out.printf("%s\n%s: %b\n%s: %b\n", "Logical NOT (!)",
            "!false", (!false), "!true", (!true));
    }
}

```

```
Conditional AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true
```

```
Conditional OR (||)
false || false: false
false || true: true
true || false: true
true || true: true
```

```
Boolean logical AND (&)
false & false: false
false & true: false
true & false: false
true & true: true
```

```
Boolean logical inclusive OR (|)
false | false: false
false | true: true
true | false: true
true | true: true
```

```
Boolean logical exclusive OR (^)
false ^ false: false
false ^ true: true
true ^ false: true
true ^ true: false
```

```
Logical NOT (!)
!false: true
!true: false
```

- bespreking

① Formaat specifier **%b** toont het woord “true” of “false” afhankelijk van de waarde van de booleaanse expressie.

11. Precedentie en associativiteit van de operatoren

We kennen nu wiskundige-, relationele-, conditionele-, logische-, increment- en decrement-operatoren. We zijn in staat om complexe voorwaarden samen te stellen.

Veronderstel volgende condities:

```
5 + 25 / 12 > 6
```

```
33 % 11 + 1 < 2
```

Deze expressies bevatten zowel wiskundige als relationele operatoren.

Mogen we gewoon de expressies evalueren van links naar rechts? **Neen, bepaalde operatoren hebben voorrang op andere.**

Operators	Associativity	Type
()	van links naar rechts	haakjes
++ --	van rechts naar links	unair postfix
++ -- + - ! (type)	van rechts naar links	unair
* / %	van links naar rechts	multiplicerend
+ -	van links naar rechts	toevoegend
< <= > >=	van links naar rechts	relationeel
== !=	van links naar rechts	gelijkheid
&	van links naar rechts	booleaanse logische EN
^	van links naar rechts	booleaanse logische exclusieve OF
	van links naar rechts	booleaanse logische inclusieve OF
&&	van links naar rechts	logische EN
	van links naar rechts	logische OF
?:	van rechts naar links	conditioneel
= += -= *= /= %=	van rechts naar links	toekenning

Merk op: de ronde haakjes zijn geen operatoren, maar staan er bij gezien ze absolute voorrang hebben op de operatoren!

Voorbeeld 1

```
5 + 25 / 12 > 6
  5 + 2 > 6
    7 > 6
      true
```

Voorbeeld 2

```
33 % 11 + 1 < 2
  0 + 1 < 2
    1 < 2
      true
```

- Haakjes kunnen gebruikt worden om de prioriteiten te negeren. Alles wat tussen de haakjes staat wordt altijd het eerst uitgewerkt.
- Bij complexe voorwaarden is het soms aangewezen haakjes te gebruiken, om de lezer zonder meer duidelijk te maken hoe de expressie wordt uitgewerkt.
- Een belangrijke stelregel is: bij twijfel, gebruik altijd haakjes.

Voorbeeld 3: een volledige applicatie met do-while

- probleemstelling: Er wordt gevraagd om een strikt positief én even geheel getal in te voeren; de invoer wordt vervolgens gecontroleerd; indien deze niet voldoet, wordt een nieuwe waarde aan de gebruiker gevraagd. Is de invoer in orde, dan wordt de invoer getoond.
- in pseudocode

```

Herhaal
    druk zin op het scherm
    lees getal in
Zolang getal <= 0 OF getal oneven
    druk getal op scherm

```

- in Java

```

package cui;

import java.util.Scanner;

public class DoWhileControle2
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int getal;

        do // INVOERCONTROLE
        {
            System.out.print("Geef een strikt positief en even geheel getal in: ");
            getal = input.nextInt();
        } while (getal <= 0 || getal % 2 != 0); // einde do/while structuur ①

        System.out.printf("Het ingevoerde getal = %d", getal);
    }
}

```

- uitvoer

<terminated> DoWhileControle2 [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (17 jun. 2020 17:52:21)

```

Geef een strikt positief en even geheel getal in: 13
Geef een strikt positief en even geheel getal in: 0
Geef een strikt positief en even geheel getal in: 26
Het ingevoerde getal = 26

```

- bespreking

① Het getal moet strikt positief en even zijn, dus zolang het niet strikt positief is OF oneven is, herhalen we de body van de lus. Is de invoer strikt positief én even dan is de voorwaarde vals en verlaten we de lus. Andere notatie: `do { ... } while (!(getal > 0 && getal % 2 == 0));`

12. De statements break en continue

- break/continue: veranderen de stroom van de controle
- break statement
 - veroorzaakt een onmiddellijke exit uit de controlestructuur
 - wordt gebruikt in while, for, do-while en switch statements
- continue statement
 - slaat de resterende statements in de body van de controlestructuur over
 - gaat verder naar de volgende iteratie
 - wordt gebruikt in while, for en do-while statements

Voorbeeld met break-statement

- in Java

```
package cui;

public class BreakTest
{
    public static void main(String[] args)
    {
        int count; // control variable also used after loop terminates

        for (count = 1; count <= 10; count++) // loop 10 times ①
        {
            if (count == 5)
                break; // terminates loop if count is 5 ②

            System.out.printf("%d ", count); ③
        }

        System.out.printf("\nBroke out of loop at count = %d\n", count);
    }
}
```

- uitvoer

<terminated> BreakTest [Java Application] C:\Program Files\Java\jdk-1

1 2 3 4

Broke out of loop at count = 5

- bespreking

① Herhaal 10 keer.

- ② Ga uit de for structuur (break) wanneer count gelijk is aan 5.
- ③ Vanaf count = 5 wordt deze uitvoerlijn niet meer uitgevoerd!

Voorbeeld met continue-statement

- in Java

```
package cui;

public class ContinueTest
{
    public static void main(String[] args)
    {
        for (int count = 1; count <= 10; count++) // loop 10 times ①
        {
            if (count == 5)
                continue; // skip remaining code in loop body if count is 5 ②

            System.out.printf("%d ", count); ③
        }

        System.out.printf("\nUsed continue to skip printing 5\n");
    }
}
```

- uitvoer

<terminated> ContinueTest [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (18 jun. 2020 10:11:17)

1 2 3 4 6 7 8 9 10

Used continue to skip printing 5

- bespreking

- ① Herhaal 10 keer.
- ② Sla de uitvoerlijn over en ga verder naar de for-lus wanneer count gelijk is aan 5.
- ③ Enkel als count = 5 wordt deze uitvoerlijn niet uitgevoerd!

Toepassingen van een break-statement

- in een switch-structuur (zie 6.5)
- in een applicatie met 2 verschillende stopvoorwaarden
 - Voorbeeld
 - probleemstelling: Vraag aan 10 studenten hun punten of stop vroeger met -1. Schrijf het gemiddelde uit als er punten werden ingevoerd, anders geef je een melding.
 - in pseudocode

```

totaal <- 0
puntenTeller <- 0
Herhaal zolang puntenTeller < 10
  lees punt in
  Als punt = -1 DAN break
  totaal <- totaal + punt
  puntenTeller <- puntenTeller + 1
Eind-herhaal
Als puntenTeller <> 0
  klasGemiddelde <- totaal / puntenTeller
  Druk klasGemiddelde
anders
  Druk "Er werden geen punten ingevoerd!"

```

- in Java

```

package cui;

import java.util.Scanner;

public class KlasgemiddeldeForEnBreak
{
    public static void main(String args[])
    {
        // declaratie en initialisatie lokale variabelen
        int totaal = 0, punt, puntenTeller = 0;
        double klasGemiddelde;
        Scanner input = new Scanner(System.in);

        // blijf herhalen tot...
        for (; puntenTeller < 10; puntenTeller = puntenTeller + 1) ①
        {
            System.out.printf("Geef score %d (/20) of stop met -1: ",
puntenTeller+1);
            punt = input.nextInt();
            if (punt == -1) ②
                break;
            totaal = totaal + punt; ③
        }
        if (puntenTeller != 0) ④
        {
            klasGemiddelde = (double) totaal / puntenTeller;
            System.out.printf("Het klasgemiddelde is %.2f%n",
klasGemiddelde);
        } else
            System.out.printf("Er werden geen punten ingegeven.%n");
        }
    }
}

```

- uitvoer 1 (geen punten ingevoerd)

```
<terminated> KlasgemiddeldeForEnBreak [Java Application] C:\Program Files\Java\jdk-  
Geef score 1 (/20) of stop met -1: -1  
Er werden geen punten ingegeven.
```

- uitvoer 2 (5 punten ingevoerd)

```
<terminated> KlasgemiddeldeForEnBreak [Java Application] C:\Program Files\Java\jdk-1  
Geef score 1 (/20) of stop met -1: 12  
Geef score 2 (/20) of stop met -1: 9  
Geef score 3 (/20) of stop met -1: 8  
Geef score 4 (/20) of stop met -1: 10  
Geef score 5 (/20) of stop met -1: 19  
Geef score 6 (/20) of stop met -1: -1  
Het klasgemiddelde is 11,60
```

- uitvoer 3 (10 punten ingevoerd)

```
<terminated> KlasgemiddeldeForEnBreak [Java Application] C:\Program Files\Java\jdk-11  
Geef score 1 (/20) of stop met -1: 9  
Geef score 2 (/20) of stop met -1: 12  
Geef score 3 (/20) of stop met -1: 10  
Geef score 4 (/20) of stop met -1: 18  
Geef score 5 (/20) of stop met -1: 14  
Geef score 6 (/20) of stop met -1: 16  
Geef score 7 (/20) of stop met -1: 2  
Geef score 8 (/20) of stop met -1: 10  
Geef score 9 (/20) of stop met -1: 14  
Geef score 10 (/20) of stop met -1: 18  
Het klasgemiddelde is 12,30
```

- bespreking

① In de for-lus bouwen de eerste stopvoorwaarde in, nl. 10 keer herhalen. De puntenTeller wordt ook NA de lus gebruikt, dus declareren we deze niet in het controlegedeelte van de for-lus maar **VOOR** de lus.

- ② Als de invoer gelijk is aan de schildwacht (hier -1) dan breken we de lus af met een break-statement.
- ③ Enkel invoer verschillend van de schildwacht wordt verwerkt.
- ④ Controleren of er wel degelijk punten werden ingevoerd.

13. Primitieve datatypes

- Primitieve types: “bouwblokken” voor ingewikkeldere types
- Java is **sterk getypeerd**: alle variabelen in een Java programma **MOETEN** een type hebben
- Primitieve types in Java zijn overdraagbaar over computerplatformen die Java ondersteunen
- De primitieve types hebben elk een zogenaamde **wrapperklasse**, nl. **Boolean**, **Character**, **Byte**, **Short**, **Integer**, **Long**, **Float** en **Double**. In de klassen **Byte**, **Short**, **Integer**, **Long**, **Float** en **Double** vinden we de constanten **MAX_VALUE** en **MIN_VALUE**. Deze bevatten respectievelijk de grootst mogelijke waarde en de kleinst mogelijke waarde voor dit type.

Type	Grootte in bits	Waarden	Standaard
boolean	8	true of false	
char	16	'\u0000' tot '\uFFFF' (0 tot 65535)	(ISO Unicode character set)
byte	8	-128 tot +127 (-2 ⁷ tot 2 ⁷ - 1)	
short	16	-32,768 tot +32,767 (-2 ¹⁵ tot 2 ¹⁵ - 1)	
int	32	-2,147,483,648 tot +2,147,483,647 (-2 ³¹ tot 2 ³¹ - 1)	
long	64	-9,223,372,036,854,775,808 tot +9,223,372,036,854,775,807 (-2 ⁶³ tot 2 ⁶³ - 1)	
float	32	<i>Negatief bereik:</i> -3.4028234663852886E+38 tot -1.40129846432481707e-45 <i>Positief bereik:</i> 1.40129846432481707e-45 tot 3.4028234663852886E+38	(IEEE 754 floating point)
double	64	<i>Negatief bereik:</i> -1.7976931348623157E+308 tot -4.94065645841246544e-324 <i>Positief bereik:</i> 4.94065645841246544e-324 tot 1.7976931348623157E+308	(IEEE 754 floating point)

13.1. Het primitieve datatype: boolean

Variabelen van het type boolean kunnen slechts **twee** waarden bevatten: **true** of **false**

Voorbeeld: in de invoer van de gebruiker zoeken we het getal 20

- in Java

```

boolean gevonden; ①
int teZoekenGetal, getal;
Scanner input = new Scanner(System.in);
gevonden = false; ②
teZoekenGetal = 20;

while (!gevonden) ③
{
    System.out.print("Geef een getal: ");
    getal = input.nextInt();
    if (getal == teZoekenGetal)
        gevonden = true; ④
}

```

- bespreking

- ① Declaratie van de variabele 'gevonden' van het type **boolean**.
- ② Initialisatie van de variabele 'gevonden' op **false**. Voorlopig is het te zoeken getal niet gevonden.
- ③ We herhalen het zoeken tot we het getal 20 gevonden hebben. Zolang de variabele 'gevonden' false is, is de voorwaarde **waar**.
- ④ Als we het te zoeken getal gevonden hebben, zetten we de variabele 'gevonden' op **true**.

13.2. Het primitieve datatype: char

In variabelen van het type char kan je **één letter of één ander teken** opslaan. Nooit meer dan één. Zo'n teken moet je in het programma tussen apostrofs zetten, bijvoorbeeld de letter a als 'a', een punt als '.', etc.

Voorbeeld: declaratie van 2 char-variabelen

- in Java

```

char letter1, letter2;
letter1 = 'a'; ①
letter2 = '*';

```

- bespreking

- ① De reden dat je apostrofs moet gebruiken rond a is dat we de compiler duidelijk willen maken dat het om een letter gaat, en niet om bijvoorbeeld een variabele die de naam a heeft.



String s = "a"; // hier gebruik je aanhalingstekens!

13.3. Het primitieve datatype: byte

In een variabele van het type byte kun je **gehele getallen van -128 (= Byte.MIN_VALUE) t.e.m. 127 (= Byte.MAX_VALUE)** bewaren.

Voorbeeld: buiten het bereik van het type gaan

- in Java

```
byte getal1, getal2, getal3;
getal1 = 50;
getal2 = -128;
getal3 = 127;
System.out.printf("%d\n", --getal2); ①
System.out.printf("%d\n", ++getal3); ②
```

- bespreking

- ① De uitvoer is 127. Dit is te verklaren door de 2-complements voorstelling van de getallen. Wanneer we -128 verminderen met 1 gaan we buiten het bereik van de negatieve getallen en komen we bij het grootste positief getal terecht.
- ② De uitvoer is -128. Wanneer we 127 verhogen met 1 gaan we buiten het bereik van de positieve getallen en komen we bij het kleinste negatief getal terecht.

13.4. Het primitieve datatype: short

In een variabele van het type short kun je **gehele getallen van -32768 (= Short.MIN_VALUE) t.e.m. 32767 (= Short.MAX_VALUE)** bewaren.

Voorbeeld: buiten het bereik van het type gaan

- in Java

```
short getal4, getal5;
getal4 = -32768;
getal5 = 32767;
System.out.printf("%d\n", --getal4); ①
System.out.printf("%d\n", ++getal5); ②
```

- bespreking

- ① De uitvoer is 32767. Dit is te verklaren door de 2-complements voorstelling van de getallen. Wanneer we -32768 verminderen met 1 gaan we buiten het bereik van de negatieve getallen en komen we bij het grootste positief getal terecht.
- ② De uitvoer is -32768. Wanneer we 32767 verhogen met 1 gaan we buiten het bereik van de positieve getallen en komen we bij het kleinste negatief getal terecht.

13.5. Het primitieve datatype: int

In een variabele van het type `int` kun je **gehele getallen van $-2\ 147\ 483\ 648$ (= `Integer.MIN_VALUE`) tot $2\ 147\ 483\ 647$ (= `Integer.MAX_VALUE`)** bewaren.

13.6. Het primitieve datatype: long

In een variabele van het type `long` kun je **gehele getallen van $-9\ 223\ 372\ 036\ 854\ 775\ 808$ (= `Long.MIN_VALUE`) tot $9\ 223\ 372\ 036\ 854\ 775\ 807$ (= `Long.MAX_VALUE`)** bewaren.

Voorbeeld:

- in Java

```
long getal6, getal7;  
getal6 = 2147483647; // bovengrens van int  
getal7 = getal6 * 10;  
System.out.printf("%d%n", getal7); ①
```

- bespreking

① De uitvoer is 21474836470.



%d wordt ook gebruikt voor byte, short, long!

13.7. De primitieve datatypen: float en double

- In veel gevallen waarin gerekend wordt, heb je reële getallen nodig, zoals bijvoorbeeld 24.75. Voor reële getallen kent het Engels de term **floating-point getallen**.
- Java heeft voor reële getallen twee verschillende primitieve typen:
 - **float**: storing single-precision floating-point
 - **double**: storing double-precision floating-point
- De **nauwkeurigheid** van getallen van het type **float** is **6 tot 7 cijfers**. Dat lijkt misschien veel, maar leidt in de praktijk al gauw tot grote onnauwkeurigheden door afrondingsfouten.
- Getallen van het type **double** nemen meer geheugen in beslag dan float, maar hebben een nauwkeurigheid van **15 cijfers**.

Voorbeeld:

- in Java

```
float float_getal;  
double double_getal;  
float_getal = 10;  
double_getal = float_getal; ①  
float_getal *= 12.123456789;  
double_getal *= 12.123456789;  
System.out.printf("%s%.14f%n", "float: ", float_getal); ②  
System.out.printf("%s%.14f%n", "double: ", double_getal); ③
```

- bespreking

- ① Automatische casting van een float naar een double.
- ② De uitvoer is **float: 121,23456573486328**
- ③ De uitvoer is **double: 121,234567890000001**