H2 Controlestructuren en operatoren

Table of Contents

1. Doelstellingen

Na het bestuderen van dit hoofdstuk ben je in staat

- om een eenvoudige methode te schrijven en aan te roepen in een applicatie
- om de juiste controlestructuur te kiezen
- om een correcte controlestructuur te schrijven in Java
- om een eenvoudige methode te gebruiken om dubbele code te vermijden

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

1) Kom uit bed
2) Doe je pyjama uit
3) Kleed 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

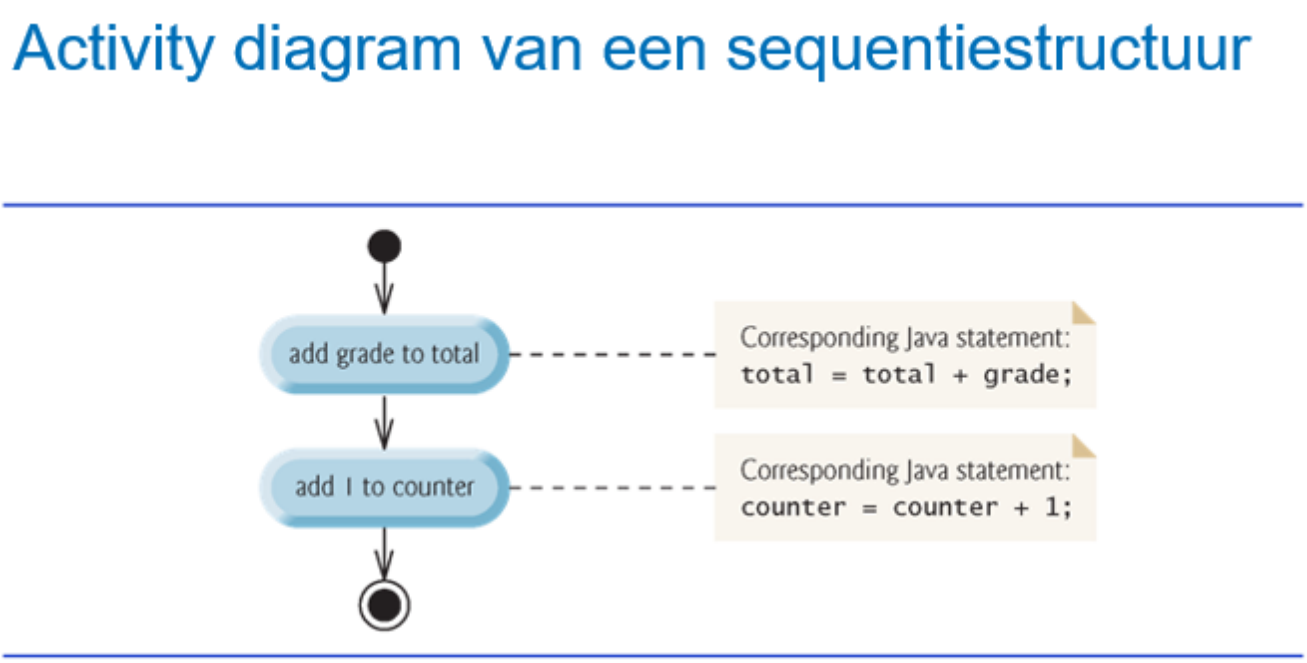
Programmacontrole is de volgorde waarin de acties uitgevoerd worden in een programma en 3 controlestructuren helpen hierbij:

- 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**
  - for
  - while
  - do/while

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.

Voor een sequentie ziet zo'n activity diagram er als volgt uit:



6. Een eenvoudige methode

6.1. Wat en hoe

In de hierna volgende programmavoorbeelden wordt telkens een methode gebruikt die vanuit de main-methode wordt aangeroepen in de applicatieklasse. Om deze methode te kunnen aanroepen moet je eerst een object maken van de klasse waarin deze main-methode staat. Zo'n object maak je met de constructor (zie H3) die wordt aangeroepen met het keyword new.

6.2. Voorbeeld

Als je in een klasse H2VoorbeeldMetMethode werkt, dan ziet die er zo uit:

```
public class H2VoorbeeldMetMethode
{
    public static void main (String[] args)
    {
    }
}
```

In deze klasse gaan we nu in de main-methode een andere methode aanroepen. Hiervoor moeten we eerst een object aanmaken van de klasse met behulp van de constructor. Je kan dit vergelijken

met de werkwijze voor het invoeren van gegevens vanaf het toetsenbord. Om de methodes hiervoor te kunnen aanroepen, hebben we een object van de klasse `Scanner` nodig, waarbij we de constructor gebruiken om dit object aan te maken.

```
public class H2VoorbeeldMetMethode
{
    public static void main (String[] args)
    {
        H2VoorbeeldMetMethode object = new H2VoorbeeldMetMethode();
    }
}
```

Via dit object gaan we nu een methode aanroepen. Zo'n methode ziet er uit zoals de main-methode, maar aangezien we een non-static methode gaan gebruiken, is er geen keyword `static`. Daarnaast is de methode ook private (dit wil zeggen dat ze niet van buiten deze klasse kan aangeroepen worden) en zijn er ook geen parameters nodig voor deze eenvoudige methode. Kies een gepaste naam voor wat de methode moet doen en zet de methodenaam in de gebiedende wijs, zoals in onderstaand voorbeeld.

Om de methode aan te roepen, gaan we opnieuw te werk zoals we dat eerder ook al bij de invoer hebben gedaan. We hebben intussen een object waar we gebruik van kunnen maken en nu kunnen we met de puntnotatie aan dit object iets vragen door er de naam van een methode, gevolgd door haakjes (waartussen eventueel parameters kunnen staan), achter te plakken.

De aanroep bestaat dus uit 3 delen: 1. de naam van het object 2. de punt 3. de methodenaam met haakjes en vergeet natuurlijk ook de puntkomma op het einde niet!

```
public class H2VoorbeeldMetMethode
{
    public static void main (String[] args)
    {
        H2VoorbeeldMetMethode object = new H2VoorbeeldMetMethode();
        object.doeIets();
    }

    private void doeIets()
    {
        // hier komt de code die beschrijft wat er in de applicatie moet gebeuren
    }
}
```

7. De selectiestructuren

7.1. 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 puntkomma in het statement. Als het statement samengesteld is, dan bestaat het statement uit meerdere enkelvoudige statements die gebundeld worden aan de hand van accolades; de selectiestructuur wordt dan 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)

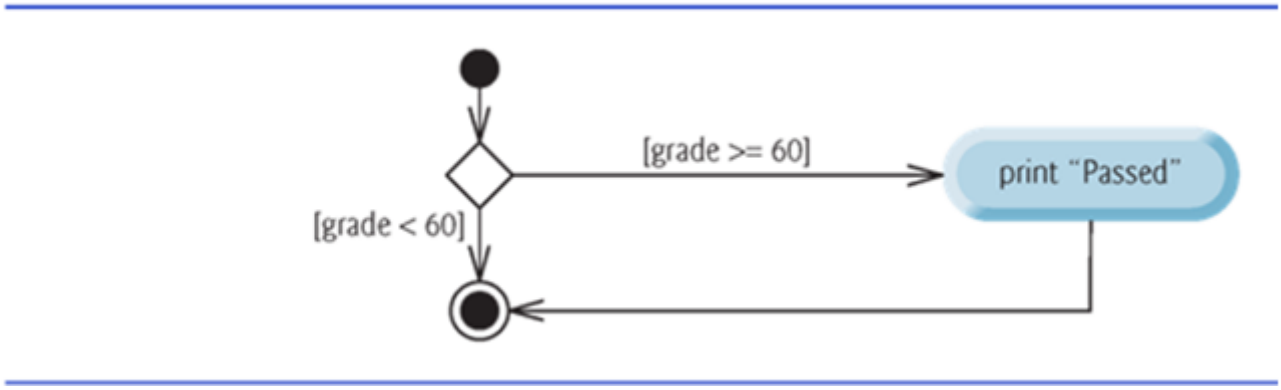
`x = y = z;` wordt geëvalueerd als `x = (y = z);`

Voorbeeld 1

- in pseudocode

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

- activity diagram



- 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 met accolades over verschillende regels gespreid
        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
```

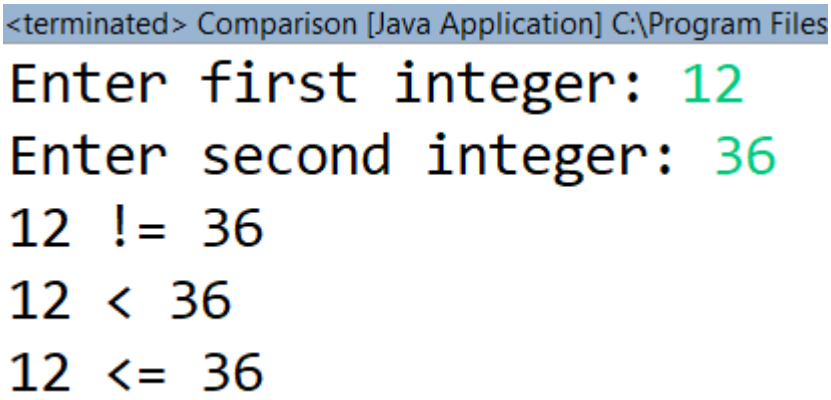
- in Java

```
        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



- 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         new Comparison().compare2Numbers();
12     }
13
14     private void compare2Numbers()
15     {
16         // creëer een object van Scanner; voor invoer vanaf het toetsenbord
17         Scanner input = new Scanner(System.in);
18
19         int number1;          // eerste getal om te vergelijken
20         int number2;          // tweede getal om te vergelijken
21
22         System.out.print("Enter first integer: "); // prompt
23         number1 = input.nextInt(); // leest eerste getal van de gebruiker
24
25         System.out.print("Enter second integer: "); // prompt
26         number2 = input.nextInt(); // leest tweede getal van de gebruiker
27
28         if (number1 == number2) ①
29         {
30             System.out.printf("%d == %d%n", number1, number2); ②
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          if (number1 >= number2)
54          {
55              System.out.printf("%d >= %d%n", number1, number2);
56          }
57      } // methode main
58  }
```

- 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!

7.2. De "if-else" selectiestructuur

if (voorwaarde) statement1 else statement2

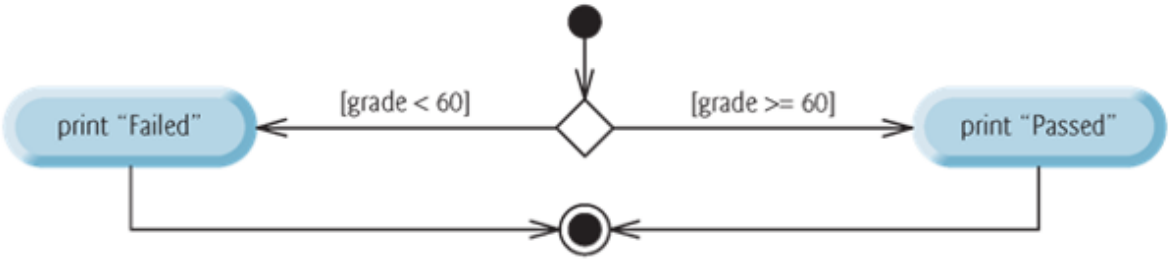
- Ofwel wordt statement1 uitgevoerd, ofwel statement2; NOOIT beide!
- Het statement1 wordt enkel uitgevoerd wanneer de voorwaarde WAAR (true) is.
- Het alternatieve statement2 wordt enkel uitgevoerd wanneer de voorwaarde VALS (false) is.

Voorbeeld 1

- in pseudocode

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

- activity diagram



- in Java

```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
```

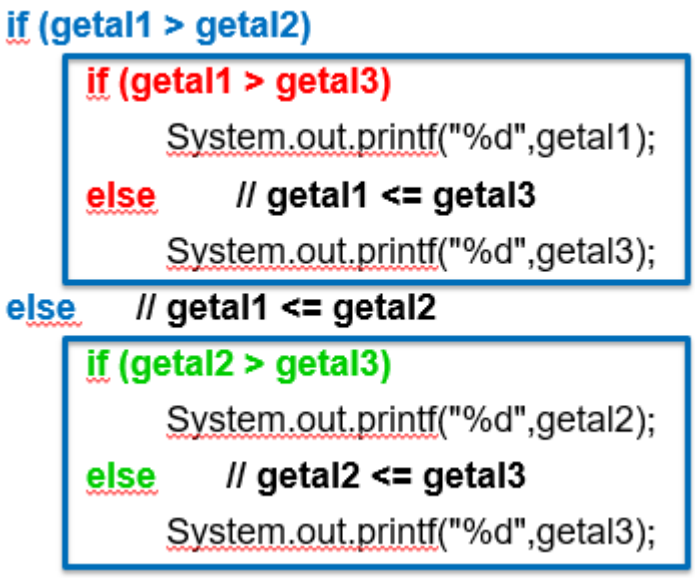
- in Java

```
        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.

7.3. Geneste "if-else" selectiestructuren

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



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!");
```

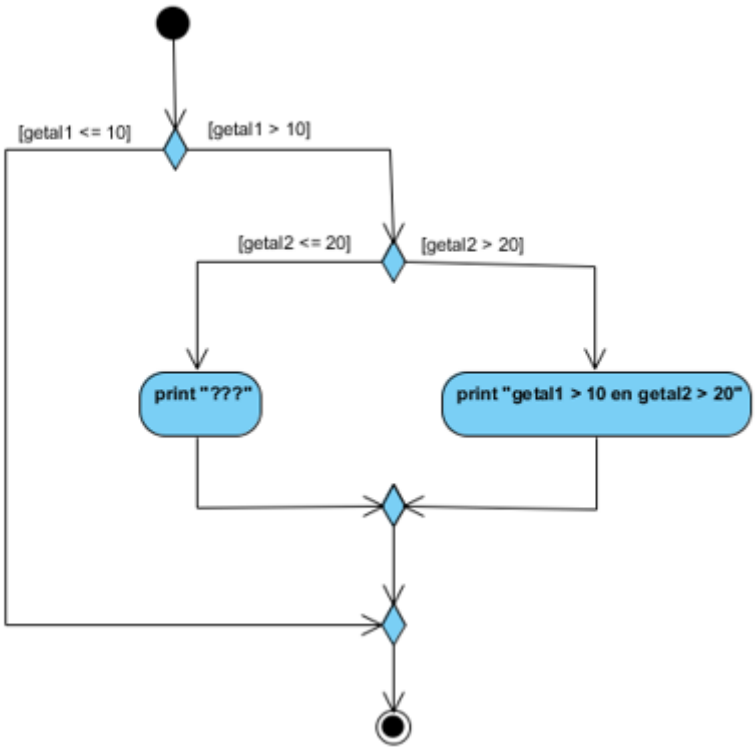
```
        else // code <> 1
            if (code == 2)
                System.out.println("Je kiest om af te trekken!");
            else // code <> 1 en code <> 2
                if (code == 3)
                    System.out.println("Je kiest om te delen!");
                else // code <> 1 en code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

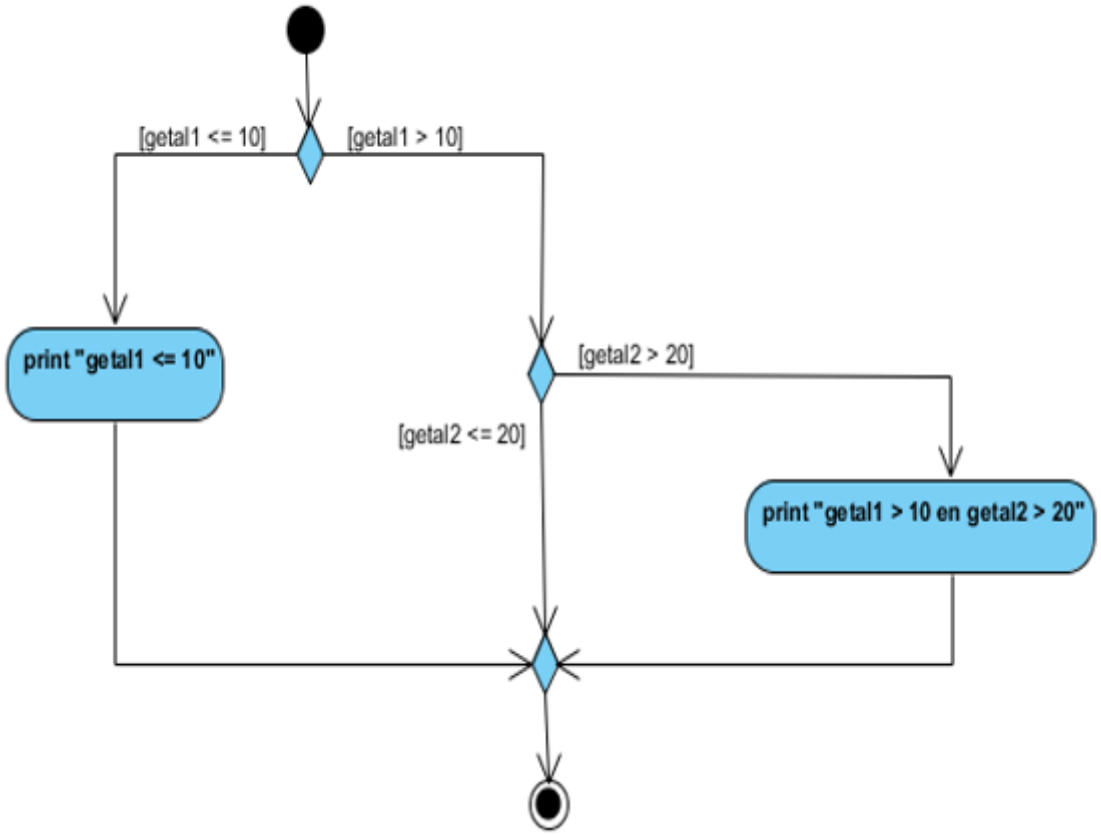
```
if (getal1 > 10)
    if (getal2 > 20)
        System.out.println("getal1 > 10 en getal2 > 20");
    else
        System.out.println(" ??? ");
```



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.

7.4. 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");
```

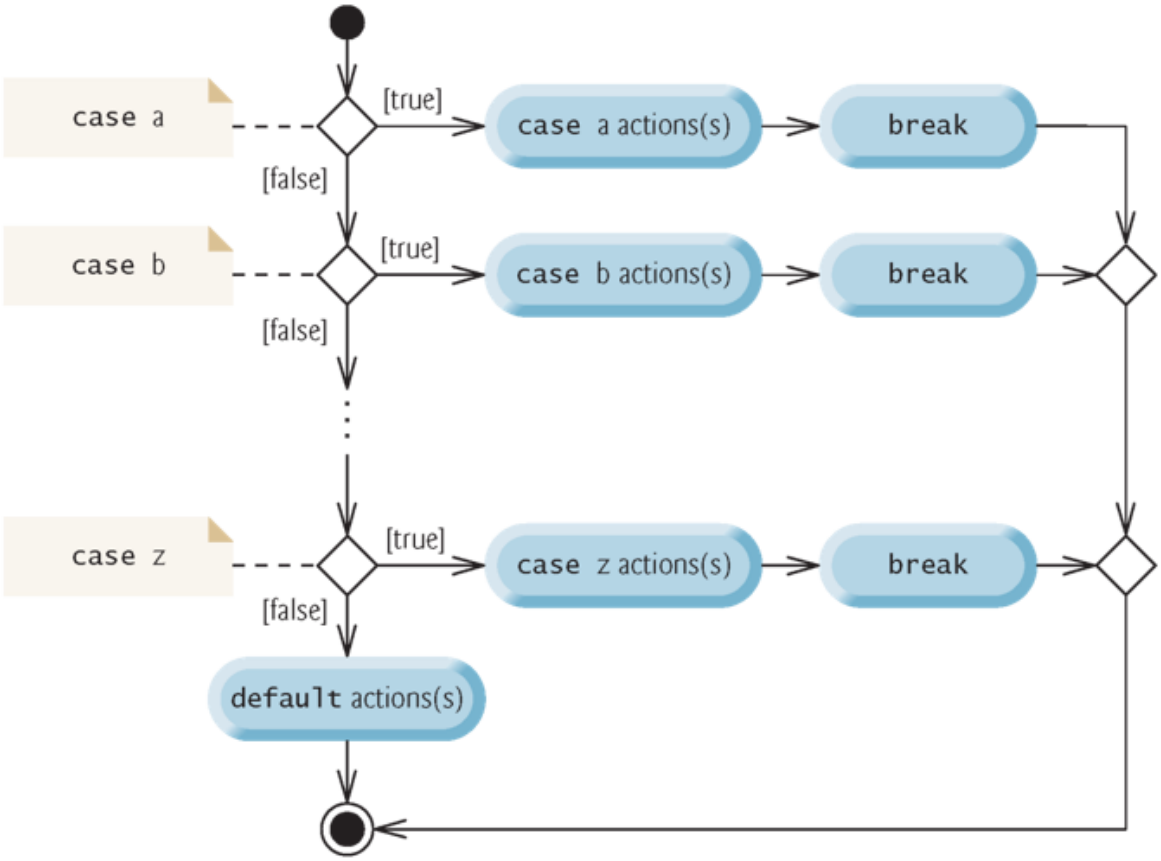
We hebben telkens in beide takken een gemeenschappelijk stukje code dat we

> vooraan kunnen plaatsen! Hebben we dat niet, dan kan deze operator niet gebruikt worden.

7.5. De "switch" meervoudige selectiestructuur

De switch-structuur wordt gebruikt voor meervoudige selecties:

switch (variabele) { case label1 → statement1; case label2 → statement2; default → statement3; }



Er zijn 5 mogelijke types voor de variabelen 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 de variabele

- De volgorde van de labels is willekeurig; default staat meestal als laatste
- Het default-label is niet verplicht

Voorbeeld 1: switch statement met int als type

```
        System.out.print("Geef uw keuze in: ");
        int keuze = input.nextInt();
        switch (keuze)
        {
            case 1 -> System.out.println("Keuze 1! Je hebt gekozen om op te tellen.");
            case 2 -> System.out.println("Keuze 2! Je hebt gekozen om te delen.");
            case 3 -> System.out.println("Keuze 3! Je hebt gekozen om af te
trekken.");
            default -> System.out.println("Foutieve keuze!");
        }
```

```
Geef uw keuze in: 1
Keuze 1! Je hebt gekozen om op te tellen.
```

```
Geef uw keuze in: 4
Foutieve keuze!
```

Switch-structuur omgezet naar geneste if-else:

```
        if (keuze == 1)
            System.out.println("Keuze 1! Je hebt gekozen om op te tellen.");
            else
                if (keuze == 2)
                    System.out.println("Keuze 2! Je hebt gekozen om te delen.");
                else
                    if (keuze == 3)
                        System.out.println("Keuze 3! Je hebt gekozen om af te
trekken.");
                    else
                        System.out.println("Foutieve keuze!");
```

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

Voorbeeld 2: switch statement met char als type

```
        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

De expressie bij switch levert het eerste karakter van de variabele stad op, zijnde 'G'. Er is met andere woorden een overeenkomst met het tweede label.

Voorbeeld 3: switch expressie met String als type

```
package cui;

import java.util.Scanner;

public class SwitchVoorbeeld3
{
    public static void main(String[] args)
    {
        new SwitchVoorbeeld3().bepaalMaandNummer();
    }

    private void bepaalMaandNummer()
    {
        Scanner invoer = new Scanner(System.in);
        String maand;

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

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

```
        // 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.
```

- Dit is een switch-expressie, met andere woorden de switch levert een resultaat op dat in een variabele kan worden opgeslagen (hier maandnummer) of als resultaat van een methode (zie H6) kan worden geretourneerd.
- Het default-label is wel vereist bij een switch-expressie, tenzij de case-labels alle mogelijkheden bevatten (bijv. bij een enumeratie - zie H6).
- We vergelijken de String in de switch expressie met de case labels door middel van 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 het default-label uitgevoerd.

Voorbeeld 4: voorbeeld 1 als switch-expressie

Aangezien in voorbeeld 1 hierboven ook een resultaat wordt bekomen aan de hand van het switch-statement (namelijk de te printen zin) en er ook een default-mogelijkheid aanwezig is, kan dit ook in de vorm van een switch-expressie gegoten worden.

```
        System.out.print("Geef uw keuze in: ");
        int bewerking = input.nextInt();
        System.out.println(
            switch (bewerking)
            {
                case 1 -> "Keuze 1! Je hebt gekozen om op te tellen.";
                case 2 -> "Keuze 2! Je hebt gekozen om te delen.";
                case 3 -> "Keuze 3! Je hebt gekozen om af te trekken.";
```

```
                default -> "Foutieve keuze!";
            }
        );
```

Voorbeeld 5: nog een voorbeeld van een switch-expressie

```
package cui;

public class Voorbeeld_switch_expressie
{
    public static void main(String[] args)
    {
        new Voorbeeld_switch_expressie().gebruik_switch();
    }

    private void gebruik_switch()
    {
        int x = 5;
        int k = switch (x)
                {
                    case 1, 2 -> 7;
                    case 3, 5, 7 -> 9;
                    default -> 0;
                };
        System.out.println("k = " + k);
    }
}
```

we kunnen meerdere labels scheiden door een komma en de switch-expressie MOET afgesloten zijn door een puntkomma

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;
```

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

De volgende expressies spreken nu voor zichzelf:

```
x -= a;
x *= a;
x /= a;
x %= a;
```

Algemeen:

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

Toekennings operator	Voorbeeld uitdrukking	Uitleg	Kent toe
Stel: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 aan c
`-=`	`d -= 4`	`d = d - 4`	1 aan d
`*=`	`e *= 5`	`e = e * 5`	20 aan e
`/=`	`f /= 3`	`f = f / 3`	2 aan f
`%=`	`g %= 9`	`g = g % 9`	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, result1, teller2, result2;
        teller1 = 0;
        teller2 = 0;
        result1 = teller1++; ①
        result2 = ++teller2; ②
        System.out.printf("%s : %d%n", "teller1", teller1);
        System.out.printf("%s : %d%n", "result1", result1);
        System.out.printf("%s : %d%n", "teller2", teller2);
        System.out.printf("%s : %d%n", "result2", result2);
```

- uitvoer

```
teller1 : 1
result1 : 0
teller2 : 1
result2 : 1
```

- bespreking

① Het statement **result1 = teller1++;** betekent, dat de waarde van variabele **teller1** met één verhoogd wordt en wordt toegekend aan result1. Maar de waarde van teller1 die wordt toegekend aan result1, is de waarde van VOOR de verhoging (dus 0 ipv 1).

result1 = teller1++; heeft hetzelfde resultaat als result1 = teller1; teller1++; OF result1 = teller1; ++teller1;

② Het statement **result2 = ++teller2;** betekent, dat de waarde van variabele **teller2** met één verhoogd wordt en wordt toegekend aan result2. In dit geval is echter de waarde van teller2 die wordt toegekend aan result2, de waarde van NA de verhoging (dus 1 ipv 0).
result2 = ++teller2; heeft hetzelfde resultaat als ++teller2; result2 = teller2; OF teller2++; result2 = teller2;

Operator	Wordt genoemd	Voorbeeld expressie	Uitleg
++	preincrement	++a	Vermeerder a met 1. De waarde die gebruikt wordt in de expressie waarin a voorkomt, is de nieuwe waarde.
++	postincrement	a++	Vermeerder a met 1. De waarde die gebruikt wordt in de expressie waarin a voorkomt, is de waarde van voor de verhoging.
--	predecrement	--b	Verminder a met 1. De waarde die gebruikt wordt in de expressie waarin a voorkomt, is de nieuwe waarde.
--	postdecrement	b--	Verminder a met 1. De waarde die gebruikt wordt in de expressie waarin a voorkomt, is de waarde van voor de verlaging.

Voorbeeld

```
        int c;

        c = 5;
        System.out.printf("%d%n", c); // print 5
        System.out.printf("%d%n", c++); // vermeerder c met 1, maar print de vorige
waarde (5) 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 met 1 en print de nieuwe
waarde (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 geprinte waarde van c is degene van **VOOR** deze aanpassing.

② Voert een preincrement uit op c, dus de geprinte waarde van c is degene van **NA** deze aanpassing.

10. De herhalingsstructuren

10.1. 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

10.2. Tellergestuurde lus

10.2.1. De for-lus

for (expressie1; expressie2; expressie3) statement

Een for-lus wordt gebruikt om een reeks instructies herhaaldelijk uit te voeren. Het wordt gebruikt wanneer je een bepaald stuk code een vast aantal keren wilt herhalen.

Een for-lus is als het ware "gemaakt" om een herhaling met een teller te implementeren.

Voorbeeld 1

- in pseudocode

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

- in Java

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

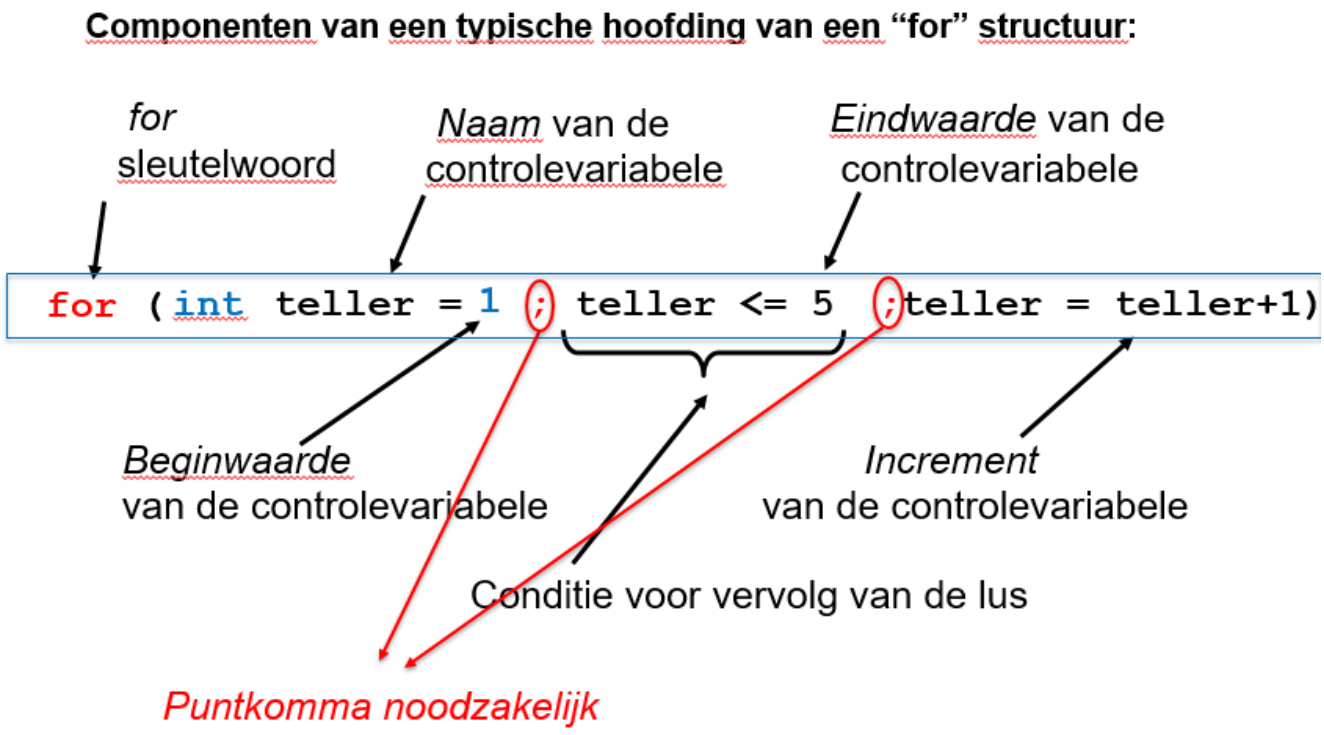
- probleemstelling: Schrijf een programma dat de getallen 1 t.e.m. 5 na elkaar afdrukt.
- in pseudocode

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



- uitvoer

```
<terminated> ForCounter
1 2 3 4 5
```

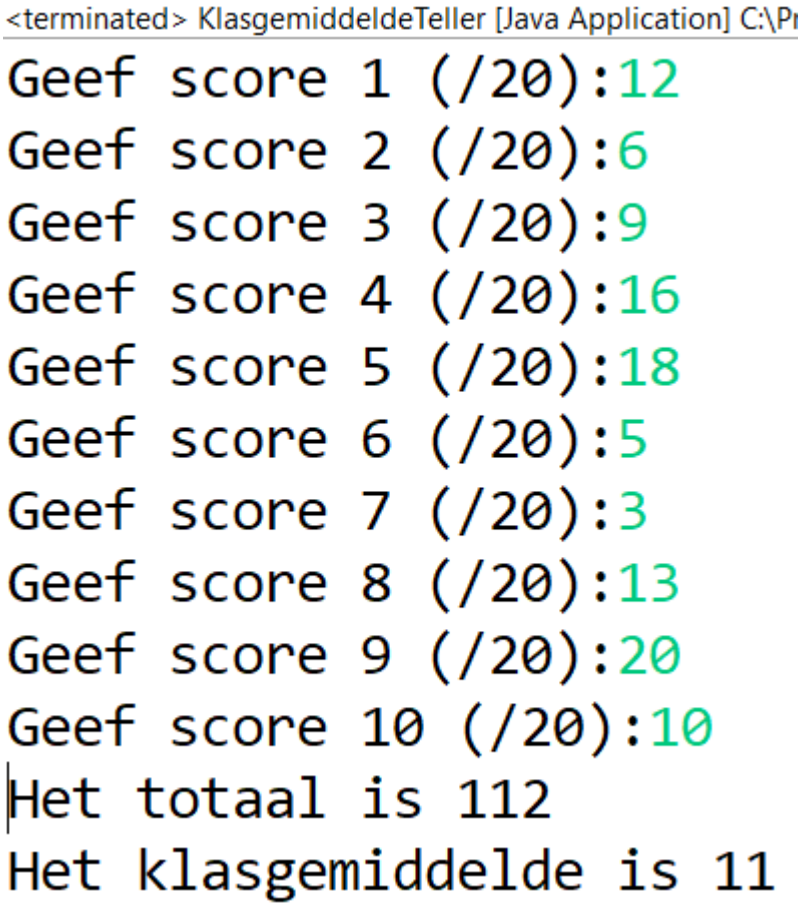
Voorbeeld 3

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

- in Java

```java
package cui;

import java.util.Scanner;

/**
 * Dit is een voorbeeld van een herhaling met teller uitgewerkt met een for!
 *
 */
public class KlasgemiddeldeTellerFOR
{
    public static void main(String args[])
    {
        new KlasgemiddeldeTellerFOR().berekenKlasgemiddelde();
    }

    private void berekenKlasgemiddelde()
    {
        //declaratie lokale variabelen
        int totaal, punt, klasGemiddelde;
        Scanner input = new Scanner(System.in);

        //initialisatie variabelen
        totaal = 0;

        //blijf herhalen tot...
        for (int puntenTeller = 1; puntenTeller <= 10; puntenTeller++)
        {
            System.out.print("Geef score " + puntenTeller + " (/20):");
            punt = input.nextInt();
            totaal += punt;
        }

        klasGemiddelde = totaal / 10;

        // uitvoer
        System.out.printf("Het totaal is %d%n", totaal);
        System.out.printf("Het klasgemiddelde is %d%n", klasGemiddelde);
    }
}
```

- uitvoer



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

```java
package cui;
// Analyse van resultaten gebruikmakend van geneste controlestructuren.

import java.util.Scanner;

public class Analysis1
{
    public static void main(String[] args)
    {
        new Analysis1().countPassesAndFailures();
    }

    private void countPassesAndFailures()
    {
        Scanner input = new Scanner(System.in);

        // initialisatie van variabelen bij declaratie
        int passes = 0;
        int failures = 0;
        int result;
```

```
21
22          // verwerk 10 studenten gebruikmakend van een tellergestuurde while-lus
23          for (int studentCounter = 1; studentCounter <= 10; studentCounter++)
24          {
25              System.out.printf("Enter result %d (1 = pass, 2 = fail): ",
    studentCounter);
26              result = input.nextInt();
27
28              // if...else is genest in het while statement
29              if (result == 1)
30                  passes++;
31              else
32                  failures++;
33          }
34
35          System.out.printf("Passed: %d%nFailed: %d%n", passes, failures);
36
37          // als er meer dan 8 studenten geslaagd zijn:
38          if (passes > 8)
39              System.out.println("Bonus to instructor!");
40      }
41  }
```

- uitvoer1

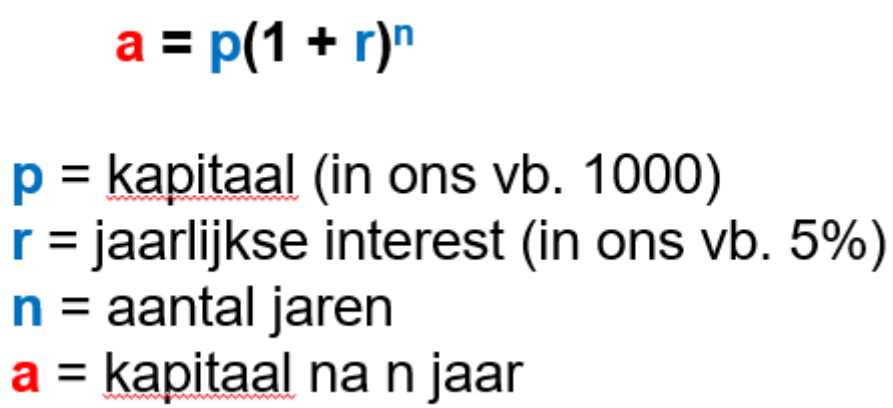
```
<terminated> Analysis [Java Application] C:\Program Files\Java\jdk-11.0.6\bin\javaw.exe (12
Enter result 1 (1 = pass, 2 = fail): 1
Enter result 2 (1 = pass, 2 = fail): 2
Enter result 3 (1 = pass, 2 = fail): 2
Enter result 4 (1 = pass, 2 = fail): 1
Enter result 5 (1 = pass, 2 = fail): 2
Enter result 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:



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

- in Java

```java
package cui;

public class Interest
{
    public static void main(String[] args)
    {
```

```
 7          new Interest().calculateAmountOnDeposit();
 8      }
 9
10      private void calculateAmountOnDeposit()
11      {
12          double amount; // saldo op het einde van elk jaar ①
13          double principal = 1000.0; // beginkapitaal
14          double rate = 0.05; // rente
15
16          // hoofdingen: veldbreedte 4 en veldbreedte 20
17          System.out.printf("%s%20s%n", "Year", "Amount on deposit");
18
19          // bereken het saldo voor elk jaar
20          for (int year = 1; year <= 10; year++)
21          {
22              amount = principal * Math.pow(1.0 + rate, year); ②
23
24              // toon het jaar en het saldo terug op veldbreedte 4 en 20
25              System.out.printf("%4d%,20.2f%n", year, amount); ③
26          } // end for
27
28      } // end main
29  } // end class Interest
```

- uitvoer

```
<terminated> Interest [Java Application] C:\Program Files\J
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         new Sum().calculate();
8     }
9
10    private void calculate()
11    {
12        int total = 0; // initialize total
13
14        // total even integers from 2 through 20
15        for (int number = 2; number <= 20; number += 2)
16            total += number;
17
18        System.out.printf("Sum is %d%n", total); // display results
19    }
20 }
```

- uitvoer

```
<terminated> Sum [Java Application]
Sum is 110
```

10.3. De andere lussen

10.3.1. 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 krijg je een oneindige lus!
- Als het statement enkelvoudig is, dan wordt de herhalingsstructuur afgesloten met het puntkomma 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

```java
package cui;

import java.util.Scanner;

public class KlasgemiddeldeSchildwacht
{
    public static void main(String args[])
    {
        new KlasgemiddeldeSchildwacht().berekenKlasgemiddelde();
```

```
10      }
11
12      private void berekenKlasgemiddelde()
13      {
14          // declaratie lokale variabelen
15          int totaal, puntenTeller, punt;
16          double gemiddelde;
17
18          // initialisatie variabelen
19          totaal = 0;
20          puntenTeller = 0;
21
22          // verwerkingsfase
23          punt = geefScore(); ①
24
25          while (punt != -1) ②
26          {
27              totaal += punt;
28              puntenTeller++;
29              punt = geefScore(); ③
30          }
31
32          if (puntenTeller != 0) ④
33          {
34              gemiddelde = (double) totaal / puntenTeller;  ⑤
35              System.out.printf("Het klasgemiddelde is %.2f%n", gemiddelde); ⑥
36          }
37          else
38              System.out.printf("Er werden geen punten ingegeven.%n");
39
40      }
41
42      private int geefScore() ⑦
43      {
44          Scanner input = new Scanner(System.in);
45          System.out.print("Geef score (/20) of -1 om te stoppen: ");
46          return input.nextInt();
47      }
48  }
```

- uitvoer 1 (invoer van 5 punten)

```
<terminated> KlasgemiddeldeSchildwacht [Java Application] C:\Program Files\Java\jdk-11.0.6\
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\
Geef score (/20) of -1 om te stoppen: -1
Er werden geen punten ingegeven.
```

- bespreking

① Invoer van de eerste punten **VOOR** de lus. We maken hier ook gebruik van een methode, aangezien we straks nog een keer een puntenaantal willen invoeren. Als we dezelfde (of gelijkaardige) code meerdere keren nodig hebben, is het beter dit te vervangen door een methode-aanroep. Anders bestaat de mogelijkheid dat we, als we de code later moeten veranderen, vergeten om dit overal te doen. Als de code maar op 1 plaats staat, verdwijnt deze mogelijkheid tot het maken van fouten.

de methode die we gebruiken moet het ingevoerde puntenaantal ook kunnen teruggeven, dus we gebruiken een methode met een terugkeertype dat niet void is!

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. Dit gebeurt via een nieuwe aanroep van de methode.

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.

⑦ De methode geefScore vraagt een puntenaantal aan de gebruiker en geeft deze terug aan degene die deze methode aanroept. Bij de aanroep moet je dus die waarde toekennen aan een variabele van het juiste type (hier int). De methode is private, want ze hoeft buiten deze klasse niet aangesproken te kunnen worden.

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         new Analysis2().countPassesAndFailures();
11     }
12
13     private void countPassesAndFailures()
14     {
15         // initialisatie van variabelen bij declaratie
16         int passes = 0;
17         int failures = 0;
18         int studentCounter = 1;// wordt hier gebruikt in de vraagstelling
19         int result;
20
21         // verwerk een aantal studenten gebruikmakend van een while-lus met
   schildwacht
22         result = askResult(studentCounter); ①
23         while (result != 0)
24         {
25             // if...else is genest in het while statement
26             if (result == 1)
27                 passes++;
28             else
```

```
29                failures++;
30
31            // verhoog studentCounter
32            studentCounter++;
33            result = askResult(studentCounter); ①
34        }
35
36        System.out.printf("Passed: %d%nFailed: %d%n", passes, failures);
37    }
38
39    private int askResult(int studentCounter) ②
40    {
41        Scanner input = new Scanner(System.in);
42        System.out.printf("Enter result %d (0 = stop, 1 = pass, 2 = fail): ",
  studentCounter);
43        return input.nextInt();
44    }
45 }
```

- 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
```

① Hier hebben we een geheel getal (0, 1 of 2) nodig voor het result. We gebruiken we een methode met een parameter om dit in te lezen. Deze parameter dient omdat we in de vraagstelling willen aangeven van de hoeveelste student we het result willen opvragen. Bij het aanroepen van de methode geven we dus de studentCounter mee.

② De methode askResult werkt net zoals de methode geefScore uit het vorig voorbeeld. Het enige verschil is dat we in de printopdracht ook tonen over de hoeveelste student het hier gaat. Deze informatie krijgen we mee via de parameter studentCounter. Deze parameter is van het type int (een geheel getal) en wordt tussen de haakjes van de methodedefinitie gezet. Deze wordt dus: `private int askResult (int studentCounter)`

10.3.2. 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.

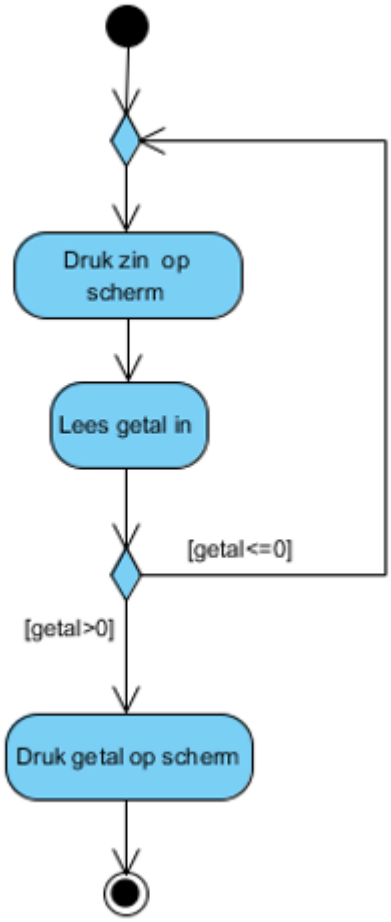
- De lus wordt altijd **MINSTENS EENMAAL** doorlopen, gezien de test **NA** het statement staat.
- Het statement wordt herhaald zolang de voorwaarde WAAR blijft!
- Zorg ervoor dat de voorwaarde beïnvloed wordt in het statement, anders krijg je een oneindige lus!
- Als het statement enkelvoudig is, dan 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)
    {
        new DoWhileControle1().controleerInvoer();
    }

    private void controleerInvoer()
    {
        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.

10.4. Samenvatting herhalingsstructuren

- Je gebruikt een **for** 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.

11. Logische operatoren

- Logische operatoren combineren eenvoudige condities tot complexere condities.

- Logische operatoren in Java

```
&&      (conditionele EN)
||      (conditionele OF)
!       (logische NIET)
```

11.1. 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! Na het controleren van de eerste voorwaarde, kan je dus mogelijk al besluiten dat het eindresultaat VALS zal zijn!

11.2. 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! Na het controleren van de eerste voorwaarde, kan je dus mogelijk al besluiten dat het eindresultaat WAAR zal zijn!

Gebruik conditionele EN (&&) en conditionele OF (||)

- 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!
```

11.3. 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)
    {
        new LogicalOperators().showResultLogicalOperators();
    }

    private void showResultLogicalOperators()
    {
          // 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 ! (logical negation) operator
        System.out.printf("%s%n%s: %b%n%s: %b%n", "Logical NOT (!)",
            "!false", (!false), "!true", (!true));
    }
}
```

- uitvoer

```
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

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.

12. 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	logische EN
||	van links naar rechts	logische OF
?:	van rechts naar links	conditioneel
= += -= *= /= %=	van rechts naar links	toekenning

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 EN 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)
    {
        new DoWhileControle2().controleerInvoer();
    }

    private void controleerInvoer()
    {
        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 EN even dan is de voorwaarde vals en verlaten we de lus. Andere notatie: **do { ... } while (!(getal > 0 && getal % 2 == 0));**

13. Primitieve datatypes

- Primitieve types zijn "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**. In het geval van Byte, Short, Integer en Long bevatten deze respectievelijk de grootst mogelijke waarde en de kleinst mogelijke waarde voor dit type. Bij Float en Double zijn het de grootst mogelijke positieve waarde voor MAX_VALUE en de kleinst mogelijke positieve waarde (waarde dicht bij 0) voor MIN_VALUE.

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	*Negatief bereik:* –3.4028234663852886E+38 tot –1.4012984632481707e–45 *Positief bereik:* 1.40129846432481707e–45 tot 3.4028234663852886E+38	(IEEE 754 floating point)
`double`	64	*Negatief bereik:* –1.7976931348623157E+308 tot –4.94065645841246544e–324 *Positief bereik:* 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 enkele quotes 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 single quotes 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 double quotes!

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: 121,23456789000001**