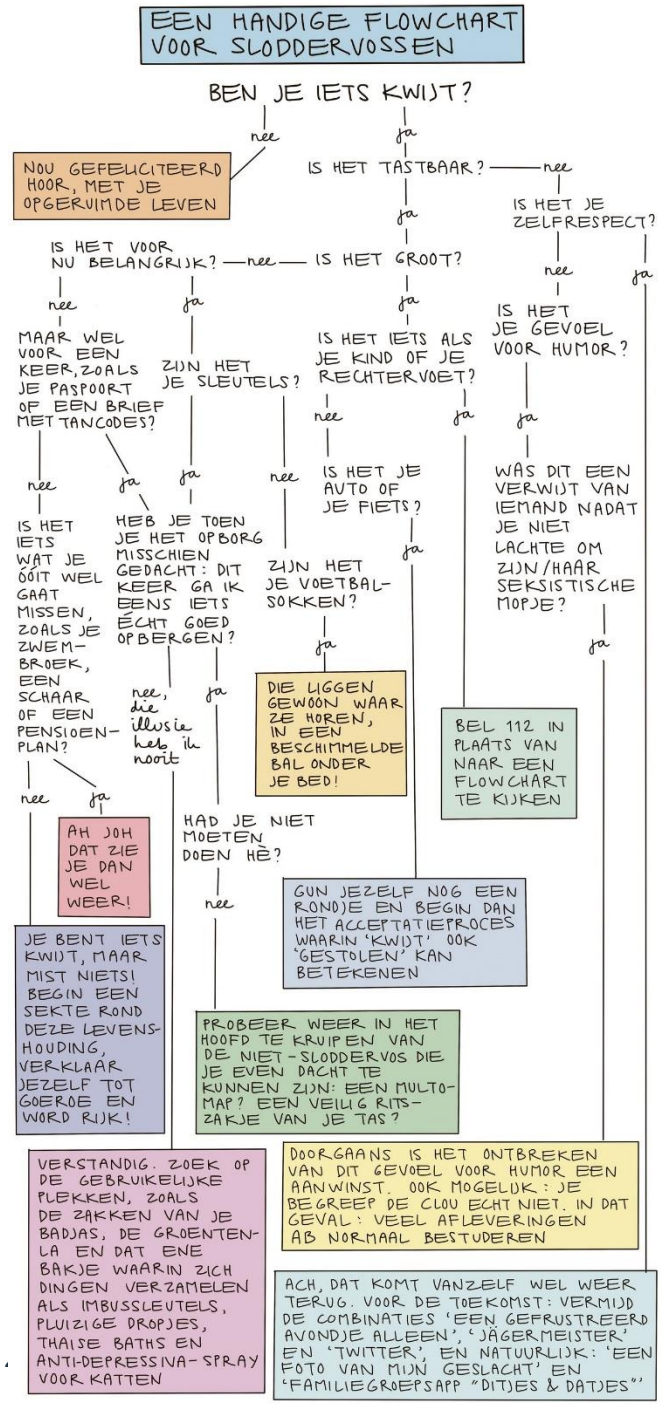
APIP #2

Condities & iteraties

Of: beslisbomen voor gevorderden

DE HAAGSE HOGESCHOOL



Vanavond

- Condities
- If/else
- Booleans
- While
- For

Doel: begrijpen *dat* en *op welke manier* een beslisboom vertaald kan worden naar Java

DE HAAGSE HOGESCHOOL

Selectie

- Een programma moet vaak keuzes maken, dit proces heet *selectie*; uit te voeren regels worden ‘geselecteerd’.
- Dat betekent dus dat vanaf nu niet elke regel meer wordt uitgevoerd.
- Selectie vindt altijd plaats op basis van een *conditie*.
  - een ‘afweging’ met als mogelijke uitkomst de waarde 'waar' (*true*) of 'onwaar' (*false*).

Condities

Condities: dicteren & interpreteren

```
x == y
```
- x gelijk aan y
- heeft x dezelfde waarde als y?

```
x < y
```
- x kleiner dan y
- is x kleiner dan y?

```
x <= y
```
- x kleiner dan of gelijk aan y
- is x kleiner dan of gelijk aan y?

```
x > y
```
- x groter dan y
- is x groter dan y?

```
x >= y
```
- x groter dan of gelijk aan y
- is x groter dan of gelijk aan y?

```
x != y
```
- x ongelijk aan y
- heeft x een andere waarde dan y?

```
a.equals(b)
```
- a 'punt 'equals ‘haakje openen’ b ‘haakje sluiten’
- bevat tekst (String) a dezelfde tekst als b?

NB:
- Geen puntkomma’s (;)
- Het zijn geen opdrachten.
- Condities staan ook nooit ‘alleen’ op een regel

- `x` en `y` zijn ints
- `a` en `b` zijn Strings

Dikke tip

- == kun je lezen als ‘is gelijk aan’
- = kun je lezen als ‘wordt’

Immers: int x = 10;

x = 20;

De inhoud van x *wordt* 10

Of 20

Of een ander getal

Is dit verstandig?

```
public class Main {
    public static void main(String[] args) {
        String dit = "verstandig";
        String dat = "verstandig";
        System.out.println(dit == dat);
    }
}
```

Condities: dicteren & interpreteren

```
x == y
x < y
x <= y
x > y
x >= y
x != y
a.equals(b)
```

- `x` en `y` zijn ints
- `a` en `b` zijn Strings

LET OP: STRINGS MAG JE NIET VERGELIJKEN MET == !!!

- a 'punt 'equals
  'haakje openen' b 'haakje sluiten'
- bevat tekst (String) a dezelfde tekst als b?

DE HAAGSE HOGESCHOOL

If / else

DE HAAGSE HOGESCHOOL

If-statement

- if ‘haakje openen’ x gelijk aan y ‘haakje sluiten’ ‘accolade openen’

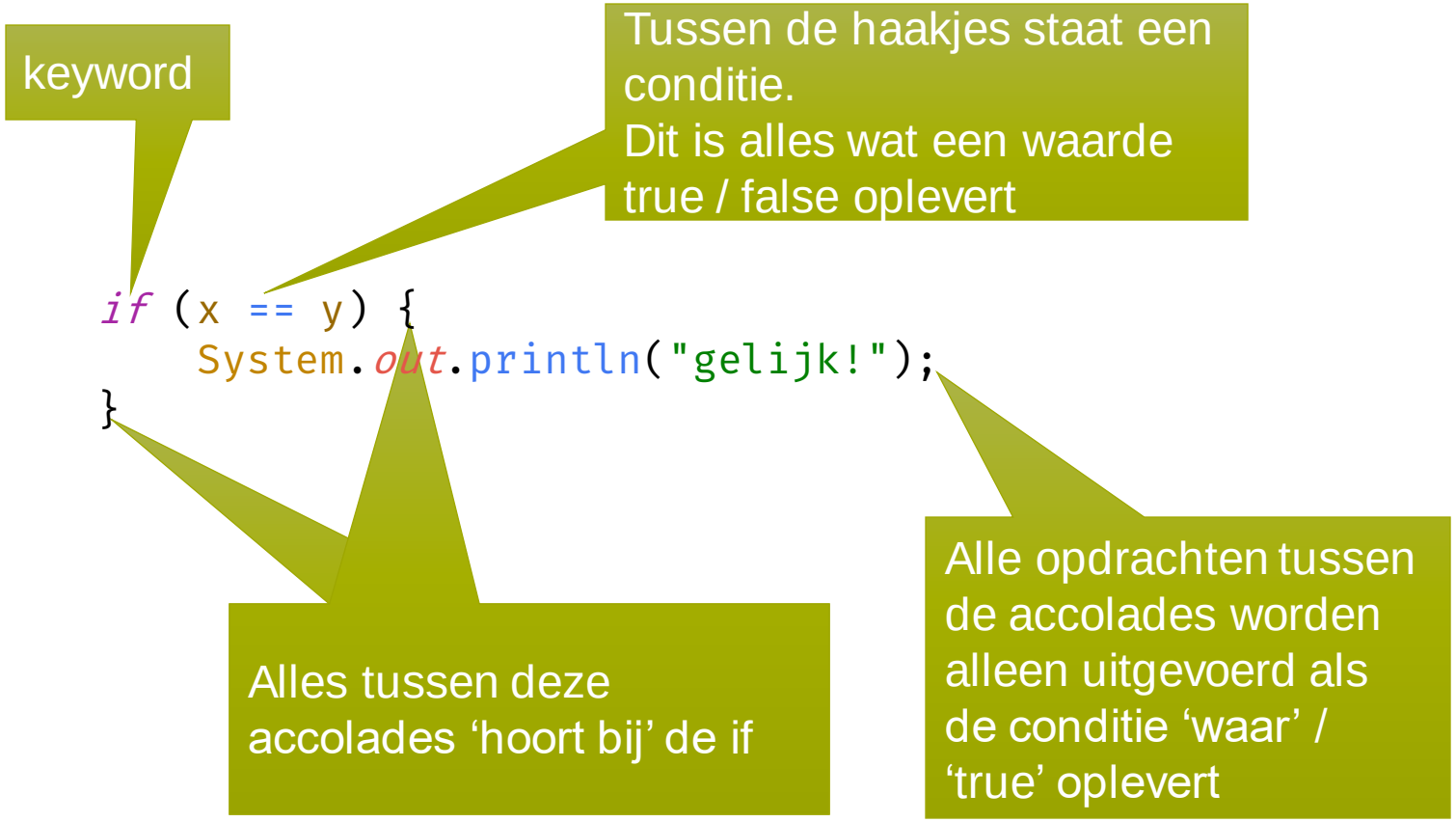
```
als (ietsWaar) {
  // dan dit
}
```

```
als (ietsWaar) {
  // dan dit
} anders {
  // dat
}
```

```
if (x == y) {
    System.out.println("gelijk!");
}
```

```
if (x == y) {
    System.out.println("gelijk!");
} else {
    System.out.println("Ongelijk");
}
```

Selectie: if statement interpreteren



DE HAAGSE HOGESCHOOL

Selectie: else statement interpreteren

```
if (x == y) {
    System.out.println("gelijk!");
} else {
    System.out.println("ongelijk!");
}
System.out.println("afgelopen!");
```

Als de conditie waar is en we klaar zijn met de `if` (hier dus) dan springen we naar de sluitaccolade van de `else` (daar dus)

keyword

Deze regel (alles na de sluitaccolade van de `else`) wordt sowieso uitgevoerd

Alle opdrachten tussen deze accolades worden alleen uitgevoerd als het statement ‘niet waar’ / ‘false’ oplevert

Pseudo-code

Om logica beter te begrijpen: een stappenplan

```
// als iemand een voldoende heeft
   // stuur ik een felicitatie
// anders
   // stuur ik een bericht met ‘jammer!’
```

```
if (cijfer >= 5.5) {
 // stuur ik een felicitatie
} else {
  // stuur ik een bericht met ‘jammer!’
}
```

```
if (voldoende) {
    // stuur ik een felicitatie
} else {
    // stuur ik een bericht met ‘jammer!’
}
```

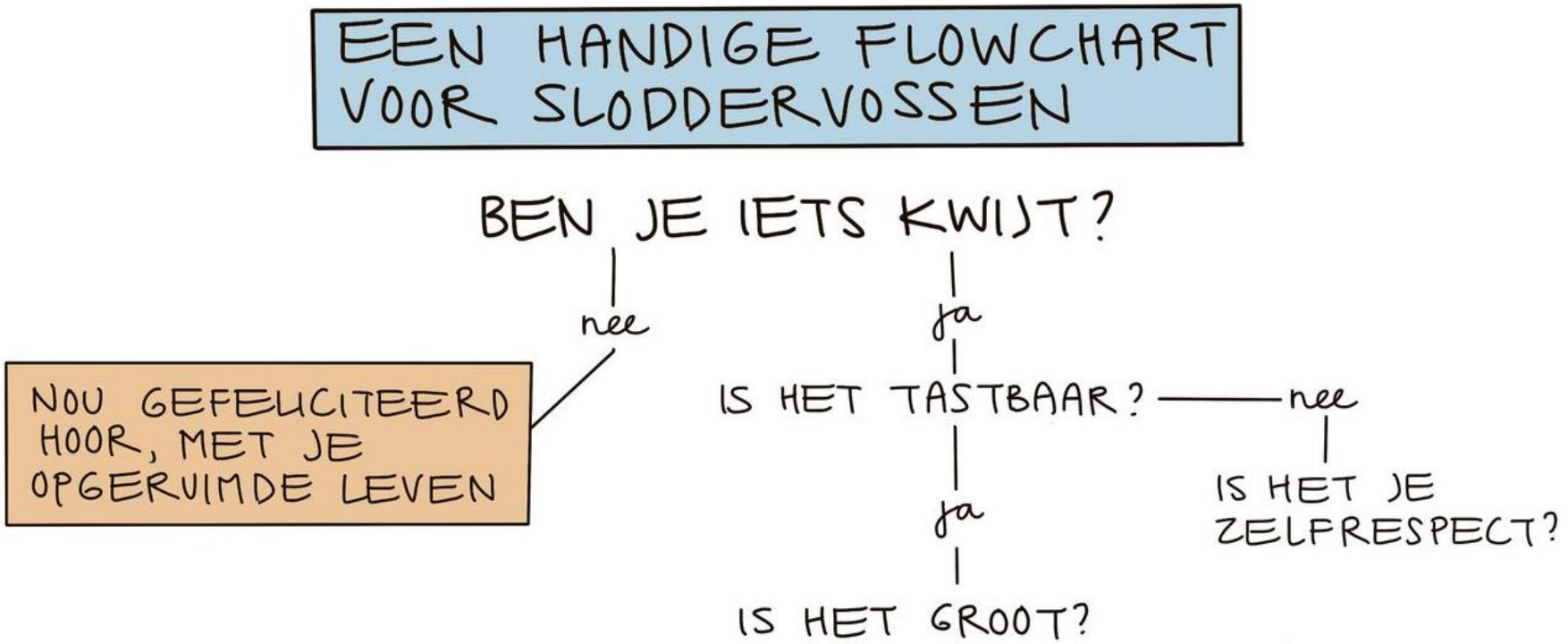
```
if (cijfer >= 5.5) {

System.out.println("Gefeliciteerd!");
} else {
    System.out.println("Jammer!");
}
```

Nesting

- Het is ook mogelijk een conditie op te nemen
  binnen de uitvoer van een andere conditie
- *'Voer een leeftijd in.*
  Als je oud genoeg bent wordt gevraagd of je een ID bij je hebt'
  - de tweede conditie (ID) wordt enkel uitgevoerd als de eerste conditie waar was.
- Dit kan (natuurlijk) ook binnen een `else`
- *'Als je niet oud genoeg bent wordt gevraagd*
  of je ouders mee zijn gekomen'

Nesting



DE HAAGSE HOGESCHOOL

Nesting: traceren

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
1.      int getal = scanner.nextInt();
2.      if (getal % 2 == 0) {
3.          if (getal % 3 == 0) {
4.              System.out.println("Dit getal heeft veel delers");
            }
5.          else {
6.              System.out.println("Dit getal is even");
            }
        }
7.      else {
8.          if (getal % 5 == 0) {
9.              System.out.println("dit getal is een 5-tal");
            }
10.         else {
11.             System.out.println("dit getal is slecht deelbaar");
            }
        }
12.     System.out.println("uitgedeeld (pft)");
        }
```

Welke regels worden uitgevoerd bij invoer:

- 8
- 25
- 30
- 111

Booleans

Boolean

- if ‘haakje openen’ geraden ‘haakje sluiten’ ‘accolade openen’

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int gok = scanner.nextInt();
    boolean geraden = (gok == 6);
    if (geraden) {
        System.out.println("gewonnen!");
    }
}
```

naam: geraden
type: boolean

De uitkomst van gok==6 wordt opgeslagen in geraden

De waarde van geraden wordt bij de if gebruikt als conditie

boolean geraden 'wordt' ‘haakje openen’
gok 'gelijk aan' 6 ‘haakje sluiten’ puntkomma

Booleaanse operatoren

```
boolean een = true;
boolean twee = false;
boolean en = een && twee;
boolean of = een || twee;
boolean niet = !een;
boolean ingewikkeld = !(een || !twee) && !(een && twee);
```

- Boolean niet 'wordt' 'niet' een 'puntkomma'
- `true` als `een` `false` is en `false` als `een` `true` is

- Boolean *en* 'wordt' een *en* twee 'puntkomma
- `true` als `een` en `twee` `true` zijn

- Boolean *of* 'wordt' een *of* twee 'puntkomma'
- `true` als `een` of `twee` `true` is of beide

Net als bij rekenen gaan haakjes voor && en ||. Ook de (unaire) operator ! Gaat voor

Wat is de waarde van ingewikkeld?

In code

EN: true als beide true zijn

```
boolean arrogant;
boolean rijk;
boolean lastig;
if (arrogant && rijk) {
    System.out.println("U bent beurshandelaar");
}
if (rijk && lastig) {
    System.out.println(" U bent van adel");
}
if (arrogant && lastig) {
    System.out.println(" U bent narcist");
}
if (arrogant && !rijk) {
    System.out.println("U heeft een verkeerd wereldbeeld");
}
if (arrogant || lastig) {
    System.out.println("U heeft geen fijne persoonlijkheid");
}
```

NIET: true als variabele false is en andersom. Niet gaat voor EN en OF

OF: true als één van beide of allebei true zijn

Waarheidstabellen

- Om de complexe samenhang van booleans in een conditie of uitkomst weer te geven wordt vaak een *waarheidstabel* gebruikt.
- Hierin staat voor *elke(!) combinatie van waarden* wat de gewenste uitkomst is.
- De eerste kolommen zijn de gegeven booleans.
- De laatste kolom is de gewenste uitkomst.

Waarheidstabel voorbeelden

Deze tabel hoort bij (een simpele) && (en)

```
uitkomst = een && twee;
```

een	twee	uitkomst
true	true	true
true	false	false
false	true	false
false	false	false

Als *een* `true` is en *twee* `false` dan moet `uitkomst` `false` zijn

2 inputs, 1 output

```
uitkomst = drie && (een || twee);
```

een	twee	drie	uitkomst
true	true	true	true
true	true	false	false
true	false	true	true
true	false	false	false
false	true	true	true
false	true	false	false
false	false	true	false
false	false	false	false

3 inputs, 1 output

Korte pauze

Terug over ? minuten

While

Wat doet het programma rechts (vergelijk met links)

```java
Scanner scanner = new Scanner(System.in);
int getal1 = scanner.nextInt();
int getal2 = scanner.nextInt();
if (getal1 != getal2) {
    System.out.println("Niet gelijk! Opnieuw");
    getal1 = scanner.nextInt();
    getal2 = scanner.nextInt();
}
System.out.println("klaar");
```

```java
Scanner scanner = new Scanner(System.in);
int getal1 = scanner.nextInt();
int getal2 = scanner.nextInt();
while (getal1 != getal2) {
    System.out.println("Niet gelijk! Opnieuw");
    getal1 = scanner.nextInt();
    getal2 = scanner.nextInt();
}
System.out.println("klaar");
```

Nieuw keyword

Iteratie

- Het komt in een programma vaak voor dat een bepaalde taak herhaald moet worden (dat heet *iteratie*)

- Er geldt dan dat een bepaald stuk code (tussen accolades) herhaald wordt **zolang** aan een bepaalde conditie (niet) voldaan wordt

DE HAAGSE HOGESCHOOL

Iteratie (while): dicteren & interpreteren

```
Scanner scanner = new Scanner(System.in);
int getal1 = scanner.nextInt();
int getal2 = scanner.nextInt();
while (getal1 != getal2) {
    System.out.println("Niet gelijk! Opnieuw");
    getal1 = scanner.nextInt();
    getal2 = scanner.nextInt();
}
System.out.println("klaar");
```

- while ‘(’ getal1 niet gelijk aan getal2 ‘)
- ’{‘ Zolang als…

Als het programma hier aankomt gaan we terug naar de while opdracht en bekijken we de conditie opnieuw

- Dit is de conditie. Elke keer als deze waar is wordt de code tussen de accolades uitgevoerd

DE HAAGSE HOGESCHOOL

Opdracht

- Stel: je hebt gespiekt bij de toets en je moet honderd strafregels schrijven (‘Ik mag niet spieken’). Kun je dan een while-loop gebruiken om dit efficiënter op te lossen?

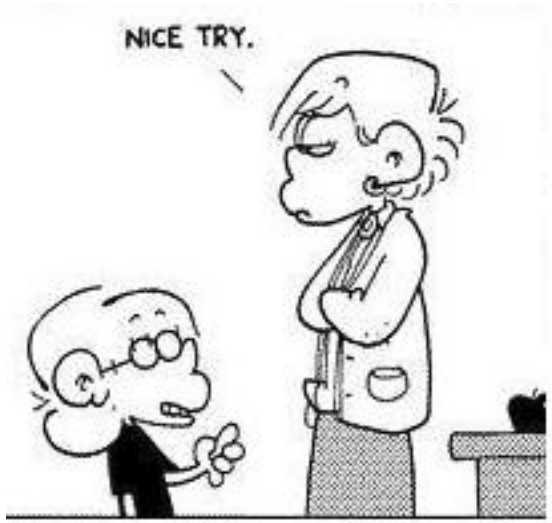
- Schrijf voor jezelf op welke stappen je zet. Als je dit direct in Java kunt: helemaal prima! Pseudo-code is ook goed, want het gaat om het principe

Iteratie: pseudo-code

```
// ik wil kunnen bijhouden hoe vaak ik de regel al heb geschreven
    // zolang het aantal kleiner is dan 100
    // schrijf ik dat ik niet mag spieken
    // het aantal geschreven regels wordt met één opgehoogd
```

Iteratie: traceren & interpreteren

```
int aantal = 0;
while (aantal < 100) {
    System.out.println("Ik mag niet spieken");
    aantal++;
}
```



DE HAAGSE HOGESCHOOL

Iteratie met teller

- Het komt vaak voor dat een loop een aantal keer moet worden uitgevoerd.
  - We zagen daar net een voorbeeld van
- In dat geval hou je een teller bij die telt hoe vaak we de code al gedaan hebben
- Een teller:
  - Heeft een beginwaarde
  - Staat in de conditie
  - Wordt verhoogd binnen de loop

Iteratie: Wat is hier aan de hand?

```
Scanner scanner = new Scanner(System.in);

String wachtwoord = scanner.nextLine();

while (!wachtwoord.equals("PaSsWoRd")) {
    System.out.println("Klopt niet. Geen toegang!");
}

System.out.println("Toegang!");
```

1. De inhoud van de variabele `wachtwoord` verandert niet bij het uitvoeren van de loop.
2. De conditie verandert dus nooit van uitkomst
3. De loop gaat dus oneindig door. (programma ‘reageert niet’)

Oneindige loops

- Zorg ervoor dat een loop altijd kan stoppen
  1. In de conditie staat een variabele (of meer)
  2. De waarde van minimaal 1 van deze variabelen *kan* veranderen in de loop (er wordt ergens een waarde aan toegekend)
  3. De verandering van die variabele kan ertoe leiden dat de conditie van waarde verandert (`false` wordt)

```
int teller = 1;
while (teller < 10) {
    teller--;
}
```

1. Programma voldoet aan eisen 1 en 2, maar niet 3

For

For loop

- De for loop kan je gebruiken als alternatief voor een while loop
- De for loop gebruikt je alleen als:
  - De loop heeft een teller (integer)
  - De teller wordt met een vaste waarde verhoogd of verlaagd
  - De teller heeft een vaste startwaarde en vaste eindwaarde

Bijvoorbeeld: Het programma moet de volgende output printen

10
20
30
40
50

While loop versus for loop

```
int i = 10; //initialiseer
while (i <= 50) { //test
    totaal += 10;
    i += 10; // update
}
```

```
for (int i = 10; i <= 50; i += 10) {
    totaal += i;
}
```

Andere syntax

for loop "kost" minder regels

de variable i wordt vaak gebruikt als naam voor de teller

While loop versus for loop

```
int i = 10; //initialiseer
while (i <= 50) { //test
    totaal += 10;
    i += 10; // update
}
```

```
for (int i = 10; i <= 50; i += 10) {
    totaal += i;
}
```

Andere syntax

for loop "kost" minder regels

de variable i wordt vaak gebruikt als naam voor de teller

While loop versus for loop

```
int i = 10; //initialiseer
while (i <= 50) { //test
    totaal += 10;
    i += 10; // update
}
```

```
for (int i = 10; i <= 50; i += 10) {
    totaal += i;
}
```

Andere syntax

for loop "kost" minder regels

de variable i wordt vaak gebruikt als naam voor de teller

While loop versus for loop

```
int i = 10; //initialiseer
while (i <= 50) { //test
    totaal += 10;
    i += 10; // update
}
```

```
for (int i = 10; i <= 50; i += 10) {
    totaal += i;
}
```

Andere syntax

for loop "kost" minder regels

de variable i wordt vaak gebruikt als naam voor de teller

Dus?

- Een for loop wordt meestal gebruikt voor een loop met een vaste beginwaarde en vaste eindwaarde van de teller
  - die je weet voordat de loop start
  - ook weet je hoe vaak je de loop doorloopt

- Als dit niet zo is gebruik je een while loop

Opdracht

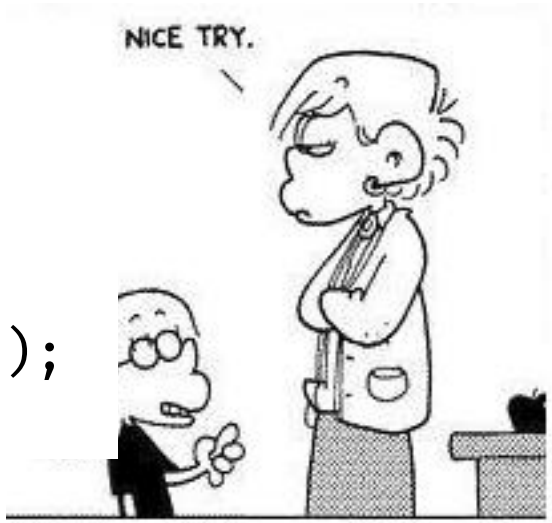
- Je hebt net met een while-loop je strafwerk geschreven. Kun je dit nóg efficiënter met for?

- Schrijf voor jezelf op welke stappen je zet. Als je dit direct in Java kunt: helemaal prima! Pseudo-code is ook goed, want het gaat om het principe

Iteratie

```
int i = 0;
while (i < 100) {
    System.out.println("Ik mag niet spieken");
    i++;
}
```

```
for (int i = 0; i < 100; i++) {
    System.out.println("Ik mag niet spieken");
}
```



DE HAAGSE HOGESCHOOL


"That's all Folks!"