APIP Les 4
Methodes & ArrayLists

DE HAAGSE HOGESCHOOL

Agenda

- Methodes
  - Introductie
  - Methode syntax en aanroep
  - Methodes samenvatting
  - Waarom methodes?
  - Methode naamgeving
- ArrayLists
  - Wat zijn ArrayLists?
  - ArrayList methodes
  - ArrayList kopiëren
  - ArrayList vs Array

Methodes

- Klein programma dat voor mij het maximale salaris kiest

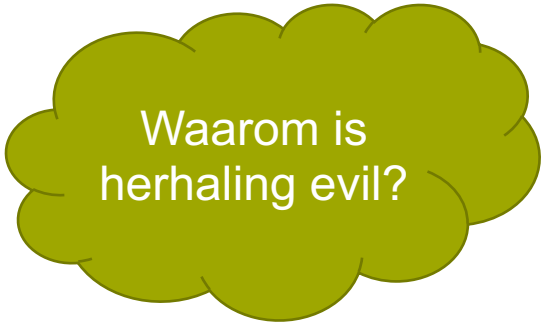
```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    int salaryOfferGoogle = scanner.nextInt();
    int salaryOfferFacebook = scanner.nextInt();

    int mySalary;
    if(salaryOfferGoogle > salaryOfferFacebook) {
        mySalary = salaryOfferGoogle;
    }
    else {
        mySalary = salaryOfferFacebook;
    }
}
```

Methodes

- Wat als ik het maximum op verschillende plekken in de code moet berekenen?
- Hoe leesbaar is dit nu?

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    int salaryOfferGoogle = scanner.nextInt();
    int salaryOfferFacebook = scanner.nextInt();

    int mySalary;
    if(salaryOfferGoogle > salaryOfferFacebook) {
        mySalary = salaryOfferGoogle;
    }
    else {
        mySalary = salaryOfferFacebook;
    }

    int travelTimeWorst;
    int travelTime1 = 50;
    int travelTime2 = 75;

    if(travelTime1 > travelTime2) {
        travelTimeWorst = travelTime1;
    }
    else {
        travelTimeWorst = travelTime2;
    }
}
```

Methodes



Als je het maximum vaker moet berekenen, wordt het een chaos
Herhaling is evil! -> DRY = Don’t repeat Yourself

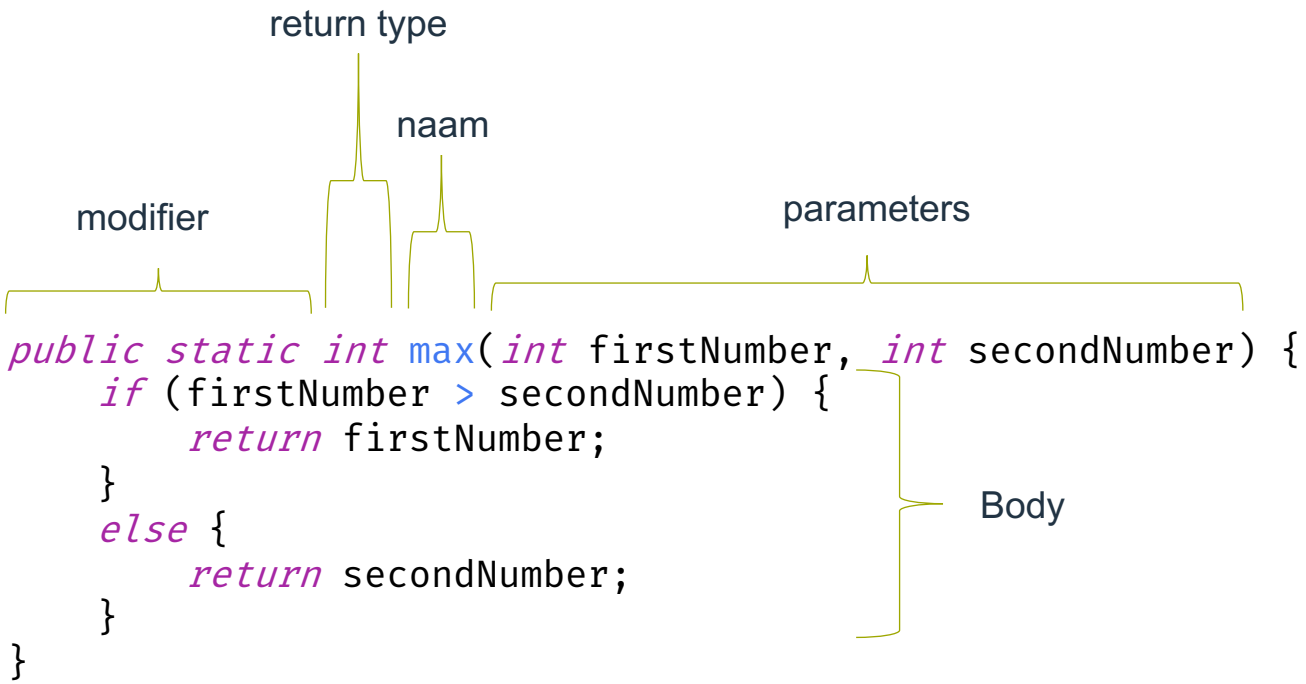
Hoe kunnen we stoppen met herhalen?
Wat als we code konden “groeperen”?

Dat kan! Zo een “groepje” code heet een **methode**

DE HAAGSE HOGESCHOOL

Methodes

- **Modifier** voor nu altijd *public static*
  - Denk er nu niet over na
- **Return type:** wat wordt er teruggegeven
  - *void*
  - *int*
  - *double*
  - *String*
  - Etc
- **Methode naam:** betekenisvol
- **Parameters:** wat geef je mee aan de methode



DE HAAGSE HOGESCHOOL

Welke methodes gebruiken en kennen we al?

Methodes

- Maak een methode die het maximum kan berekenen
- Roep deze methode aan op alle plekken waar je het maximum nodig hebt
- Stuk leesbaarder toch?

```java
import java.util.Scanner;

public class Main {

    public static int max(int firstNumber, int secondNumber) {
        if (firstNumber > secondNumber) {
            return firstNumber;
        }
        else {
            return secondNumber;
        }
    }


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

        int salaryOfferGoogle = scanner.nextInt();
        int salaryOfferFacebook = scanner.nextInt();

        int mySalary = max(salaryOfferGoogle, salaryOfferFacebook);

        int travelTime1 = 50;
        int travelTime2 = 75;

        int travelTimeWorst = max(travelTime1, travelTime2);
    }
}
```

Methode aanroep, parameters, argumenten en return keyword

- **Aanroep methode in de laatste regel**

```
public class Main {

    public static int max(int firstNumber, int secondNumber) {
        if (firstNumber > secondNumber) {
            return firstNumber;
        }
        else {
            return secondNumber;
        }
    }


    public static void main(String[] args) {
        int salaryOfferGoogle = 20;
        int salaryOfferFacebook = 30;

        int mySalary = max(salaryOfferGoogle, salaryOfferFacebook);
    }
}
```

Methode aanroep, parameters, argumenten en return keyword

- Aanroep methode in de laatste regel

- **Bij de aanroep worden er 2 argumenten meegegeven:**
  `salaryOfferGoogle` **en** `salaryOfferFacebook`

```
public class Main {

    public static int max(int firstNumber, int secondNumber) {
        if (firstNumber > secondNumber) {
            return firstNumber;
        }
        else {
            return secondNumber;
        }
    }


    public static void main(String[] args) {
        int salaryOfferGoogle = 20;
        int salaryOfferFacebook = 30;

        int mySalary = max(salaryOfferGoogle, salaryOfferFacebook);
    }
}
```

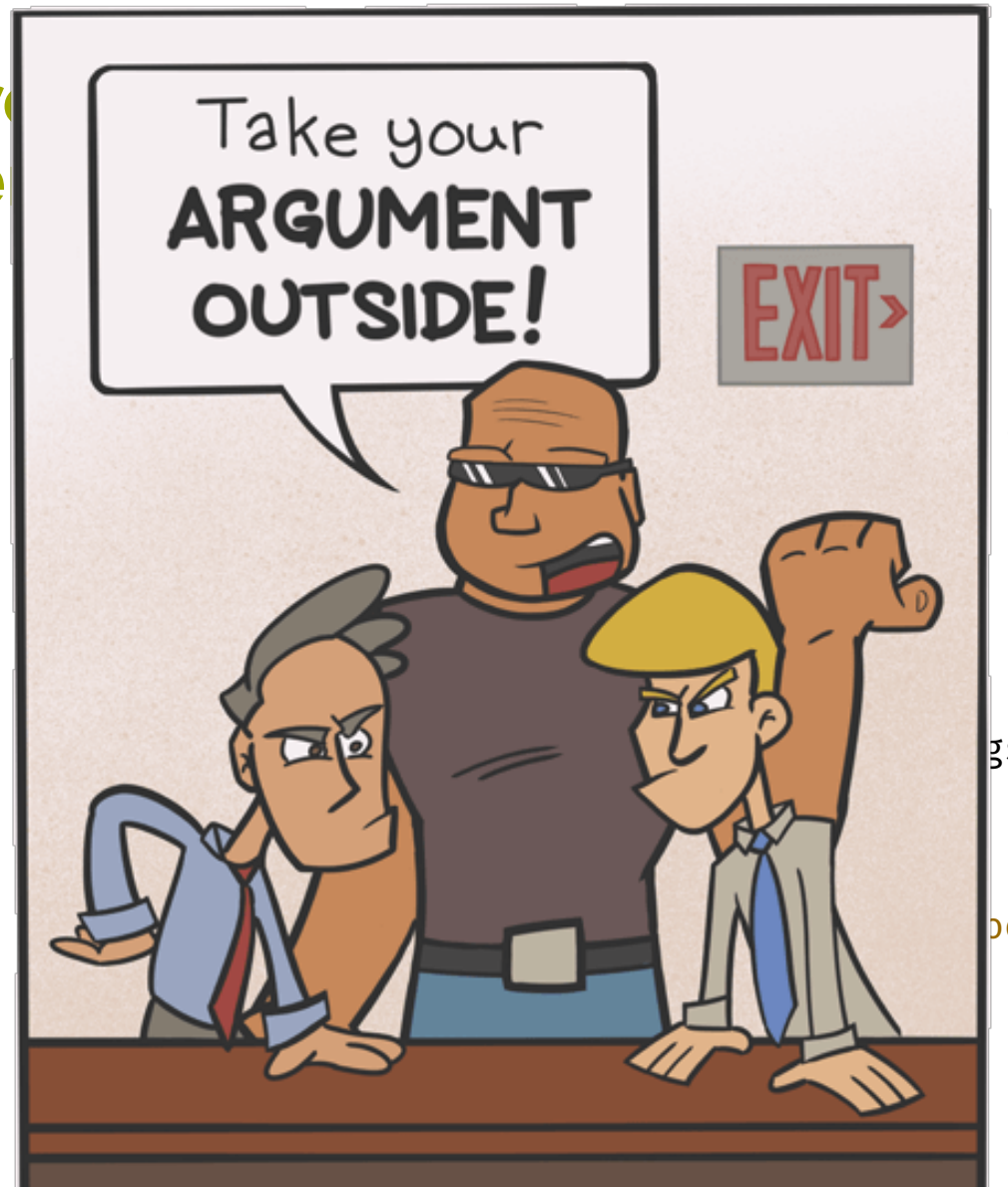
Methode aanroep, parameters, argumenten en return keyword

- Aanroep methode in de laatste regel
- Bij de aanroep worden er 2 argumenten meegegeven: `salaryOfferGoogle` en `salaryOfferFacebook`
- **Meegegeven argumenten worden overgekopieerd naar parameters**

```
public class Main {

    public static int max(int firstNumber, int secondNumber) {
        if (firstNumber > secondNumber) {
            return firstNumber;
        }
        else {
        return secondNumber;
    }
}


    public static void main(String[] args) {
        int salaryOfferGoogle = 20;
        int salaryOfferFacebook = 30;

        int mySalary = max(salaryOfferGoogle, salaryOfferFacebook);
    }
}
```

Methode aanr
argumenten e

- Aanroep methode in de laatste regel
- Bij de aanroep worden er 2 argumenten meegegeven: `salaryOfferGoogle` en `salaryOfferFacebook`
- Meegegeven argumenten worden overgekopieerd naar parameters




```
{

gs) {

ogle, salaryOfferFacebook);
```


DE HAAGSE HOGESCHOOL

Methodes met void return type

- Een methode mag ook “niks” teruggeven.
- Het return type is dan `void`

```
public class Main {
    public static void greet() {
        System.out.println("Hello :D");
    }

    public static void main(String[] args){
        greet();
    }
}
```

Opdracht: methode som array

- Schrijf een methode die het totaal berekent van alle elementen in een array

```
public static double sum(double[] array) {



}
```

Opdracht: methode som array

- Methode die het totaal berekent van alle elementen in een array

```
public static double sum(double[] a) {
    double sum = 0;
    for (int i = 0; i < array.length; i++) {
        sum += array[i];
    }
    return sum;
}
```

Opdracht: methode gemiddelde array

- Schrijf een methode die het gemiddelde van een array berekent
- De array bevat allemaal doubles
- Het gemiddelde kan een kommagetal kan zijn en geeft dus een double terug

Opdracht: methode gemiddelde array

- Methode voor het gemiddelde van een array
- Hoe kan dit nóg slimmer?

```
public static double mean(double[] array) {
    int total = 0;

    for (int i = 0; i < array.length; i++) {
        total += array[i];
    }

    return total / array.length;
}
```

Opdracht: methode gemiddelde array

Methode voor het gemiddelde van een array

- Gebruik de reeds bestaande sum methode van 3 slides terug!

```
public static double mean(double[] array) {
    return sum(array) / array.length;
}
```

Korte pauze

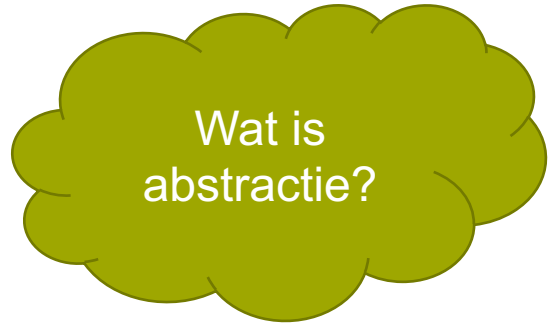
Terug over ? minuten 

DE HAAGSE HOGESCHOOL

Methodes samenvatting

- Een methode is “gegroepeerde code” dat herhaaldelijk aangeroepen kan worden
- Heeft een naam om aangeroepen te kunnen worden
- Methodes kunnen 0, 1 of meer parameters (inputs) krijgen
- Slechts 1 ding teruggeven van het return type
- Teruggeven betekent niet printen

Waarom methodes?



Programma's worden snel heel groot

- Methodes creëren "abstractie"
  - Implementatie methode boeit niet
  - Wat hij voor mij doet en wat hij ervoor nodig heeft is belangrijk
- Methodes bieden een manier om overzicht te bewaren
- Methodes bevorderen herbruikbaarheid
- Methodes bevorderen aanpasbaarheid

DE HAAGSE HOGESCHOOL

Methode naamgeving

- Afgeleid van werkwoorden, want een methode doet iets
- Gebruik betekenisvolle en logische namen
  - camelCase
- Ofwel helemaal Engels ofwel helemaal Nederlands
  - Één van de twee en niet allebei tegelijkertijd
- Mijn persoonlijke mening

Overloading

- Meerdere methodes met zelfde naam maar andere “signature”
  - Meer of minder input parameters
  - Ander type parameters
  - Ander return type

```
public class Main {

    public static int sum(int num1, int num2) {
        return num1 + num2;
    }

    public static int sum(int num1, int num2, int num3) {
        return num1 + num2 + num3;
    }

    public static double sum(double num1, double num2) {
        return num1 + num2;
    }

    public static String sum(String str1, String str2) {
        return str1 + str2;
    }

    public static void main(String[] args) {
        sum(13, 18);
        sum(21, 13, 4);
        sum(1.5, 4.7);
        sum("Haha", "hihi");
    }

}
```

ArrayList

ArrayList introductie

- Arrays waren al fantastisch
- ArrayLists zijn nog beter!

- Nadeel array: altijd vaste grootte
- Moeilijk om elementen toe te voegen
- Moeilijk om elementen te verwijderen en array te krimpen

- ArrayLists lossen dat op

ArrayList introductie

- ArrayLists groeien en krimpen als dat nodig is.
- ArrayLists bevatten methodes die al een hoop voor je doen.
  - `add();`
  - `get();`
  - `set();`
  - `remove();`
- ArrayLists hebben ook ander handige kant en klare methodes
  - (`addAll(), contains(), indexOf(), clear(),...`)

ArrayList syntax

- Lege ArrayList aanmaken:

```
ArrayList<type> lst  =  new ArrayList<type>();
ArrayList<type> lst  =  new ArrayList<>();
```

- Nadeel: primitieve types kunnen niet in een ArrayList

~~`ArrayList<int> grades  =  new ArrayList<>();`~~

```
ArrayList<Integer> grades =  new ArrayList<>();
```

Wrapper gebruiken

int	->	Integer,
double	->	Double,
boolean	->	Boolean

ArrayList elementen toevoegen

- Waardes plaatsen met de add methode:

```
ArrayList<String> names = new ArrayList<>();
names.add("Ab");
names.add("Bas");
names.add("Charlie");
names.add("Zoë");
```

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Dirk”
[4] = “Zoë”

- add methode met 2 parameters (overloading)

```
names.add(3, "Dirk");
```

ArrayList elementen uitlezen

```
ArrayList<String> names = new ArrayList<>();
names.add("Ab");
names.add("Bas");
names.add("Charlie");
names.add("Zoë");
```

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Zoë”

- We gebruiken de get methode om te lezen

```
System.out.println(names.get(1));
```

ArrayList elementen overschrijven

```
ArrayList<String> names = new ArrayList<>();
names.add("Ab");
names.add("Bas");
names.add("Charlie");
names.add("Zoë");
```

names
[0] = “Ab”
[1] = “Bassie”
[2] = “Charlie”
[3] = “Zoë”

- We gebruiken de ‘set’ methode om waardes te (over)schrijven

```
names.set(1, "Bassie");
System.out.println(names.get(1));
```

ArrayList elementen verwijderen

```
ArrayList<String> names = new ArrayList<>();
names.add("Ab");
names.add("Bas");
names.add("Charlie");
names.add("Zoë");
```

names
[0] = “Ab”
[1] = “Zoë”

- We gebruiken de remove methode om te verwijderen

```
names.remove("Charlie"); // We geven de waarde mee
names.remove(1);// We geven de index mee
```

Korte pauze

Terug over ? minuten 

DE HAAGSE HOGESCHOOL

ArrayLists aanmaken en initialiseren met data

- Aanmaken en initialiseren met data van array vs ArrayList

Arrays:

```
String[] arrayNames = {"Ab", "Bas", "Charlie", "Zoë"};
```

ArrayLists:

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
```

ArrayList lengte

length vs size()

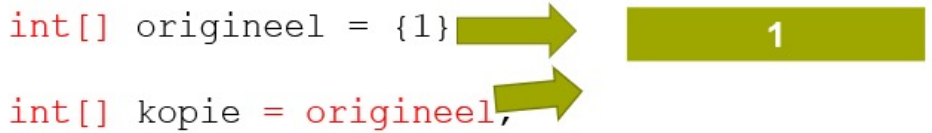
```
String[] arrayNames = {"Ab", "Bas", "Charlie", "Zoë"};

ArrayList<String> names = new ArrayList<>(Arrays.asList(arrayNames));

System.out.println(arrayNames.length);
System.out.println(names.size()); // method
```

ArrayList kopiëren

Weet je nog… Arrays kun je niet zomaar kopiëren, omdat je te maken hebt met references.

We moesten de hele array doorlopen en alles met de hand één voor één overkopiëren.

Ook ArrayLists kan je niet zomaar kopiëren. De oplossing is wel eenvoudiger!



DE HAAGSE HOGESCHOOL

ArrayList kopiëren

- Arrays

```
String[] names = {"Ab", "Bas", "Charlie", "Zoë"};
String[] copy = new String[names.length];

for(int i = 0; i < names.length; i++){
    copy[i]=names[i];
}
```

- ArrayLists

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab", "Bas", "Charlie", "Zoë"));
ArrayList<String> copy = new ArrayList<>(names); // geef names mee bij het aanmaken
```

ArrayList aan ArrayList toevoegen

teachers	names
[0] = “Fatih”	[0] = “Ab”
[1] = “Maltie”	[1] = “Bas”
[2] = “Jelle”	[2] = “Charlie”
[3] = “Jaap”	[3] = “Zoë”

- addAll():

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
ArrayList<String> teachers = new ArrayList<>(Arrays.asList("Fatih","Maltie","Jelle","Jaap"));
names.addAll(teachers);
```

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Zoë”
[4] = “Fatih”
[5] = “Maltie”
[6] = “Jelle”
[7] = “Jaap”

ArrayList aan ArrayList toevoegen

teachers	names
[0] = “Fatih”	[0] = “Ab”
[1] = “Maltie”	[1] = “Bas”
[2] = “Jelle”	[2] = “Charlie”
[3] = “Jaap”	[3] = “Zoë”

- addAll() overload:

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
ArrayList<String> teachers = new ArrayList<>(Arrays.asList("Fatih","Maltie","Jelle","Jaap"));
names.addAll(0, teachers); //gebruik index om aan te geven vanaf waar…
```

names
[0] = “Fatih”
[1] = “Maltie”
[2] = “Jelle”
[3] = “Jaap”
[4] = “Ab”
[5] = “Bas”
[6] = “Charlie”
[7] = “Zoë”

ArrayList element zoeken

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Zoë”

- contains():

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
System.out.println(names.contains("Bas")); // true
System.out.println(names.contains("Piet")); // false
```

ArrayList element zoeken

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Zoë”

- indexOf():

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
System.out.println(names.indexOf("Bas")); // 1
System.out.println(names.indexOf("Piet")); // -1
```

Handige methodes

names
[0] = “Ab”
[1] = “Bas”
[2] = “Charlie”
[3] = “Zoë”

- clear():

```
ArrayList<String> names = new ArrayList<>(Arrays.asList("Ab","Bas","Charlie","Zoë"));
names.clear(); //ArrayList is nu leeg
```

ArrayList veelgemaakte fout

- Vergeten ArrayList te initialiseren

```
public static void main(String[] args){
    ArrayList<String> names;
    names.add("Fatih");
}
```

ArrayList veelgemaakte fout oplossing

- Vergeten ArrayList te initialiseren

```
public static void main(String[] args){
    ArrayList<String> names = new ArrayList<>();
    names.add("Fatih");
}
```

Wanneer array en wanneer ArrayList?

Array

- Als de grootte niet meer gaat veranderen
- Als je primitieve waardes gaat opslaan

ArrayLists

- Alle andere gevallen

Vragen?

Huiswerk

- Schrijf alles op wat je nu nog weet als check voor jezelf
- Zoek zelf nog meer handige methodes op van ArrayLists
- Zoek ook methodes op voor arrays


"That's all Folks!"

Pass by value

- Bij een methode aanroep worden “value types” gekopieerd van argument naar de parameter

```
public static void increment(int num) {
    num++;
}

public static void main(String[] args) {
    int myAge = 30;
    increment(myAge);
    System.out.println(myAge);
}
```

Pass by value

- Bij een methode aanroep worden “value types” gekopieerd van argument naar de parameter
- Variabele myAge zal nog steeds 30 zijn terwijl x daadwerkelijk verhoogd wordt

```
public static void increment(int num) {
    num++;
}

public static void main(String[] args) {
    int myAge = 30;
    increment(myAge);
    System.out.println(myAge);
}
```

Pass by reference

- Sommige datatypes zijn zo groot dat Java ervoor heeft gekozen om ze niet over te kopiëren
- In plaats daarvan geeft het een referentie mee en daarmee heeft de methode directe toegang tot de data
- Dit worden “reference types” genoemd

Pass by reference

```
public static void increment(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i]++;
    }
}

public static void main(String[] args) {
    int myAge = 30;
    int yourAge = 40;

    int[] ages = {myAge, yourAge};
    increment(ages);

    for (int i = 0; i < ages.length; i++) {
        System.out.println(ages[i]);
    }
}
```

Variabele scope

Regel: Elke variabel heeft een scope en je kan slechts binnen de scope van een variabel ermee werken

Voorbeeld:

```
for(int i=0; i < 5; i++){
        System.out.println(i)
}
System.out.println(i); //dit werkt niet!
```

De scope van een variabel wordt bepaald door brackets

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

Binnen de brackets kan je de variabel gebruiken

Variabele scope - Methodes

Hoe werkt dat binnen methodes?

```
public class Main {
    public static void main(String[] args) {
        int getal = 5;
        System.out.println(getal); //5
        verhoog(getal);
        System.out.println(getal); //5!?
    }

    public static void verhoog(int getal) {
        getal++;
        System.out.println(getal);// 6
        }
}
```

Variabele scope – Methodes II

Uitzondering: referencetypes (o.a. arrays)

```
public class Main {
    public static void main(String[] args) {
        int[] getallen = {5};
        System.out.println(getallen[0]); //5
        verhoog(getallen);
        System.out.println(getallen[0]); //6
    }

    public static void verhoog(int[] getallen) {
        getallen[0]++;
        System.out.println(getallen[0]);// 6
    }
}
```

Variabele scope: Globale variabelen

Stel: Ik wil door mijn gehele programma bij een variabel komen. Dat kan!

```
public class Main {
    public static int temperatuur = 0;

    public static void main(String[] args) {
        temperatuur = 30;
        verhoog();
        verlaag();
        printTemperatuur();
    }

    public static void verhoog() {
        temperatuur++;
    }
    public static void verlaag() {
        temperatuur--;
    }
    public static void printTemperatuur() {
        System.out.println(temperatuur);
    }
}
```