
JAVA FUNDAMENTALS – SAMENVATTING

HFST 1 : JAVA FUNDAMENTALS

NAAM VAN EEN KLASSE

Java is case-sensitive

bestandsnaam = naam van de klasse

Hallo.java

```
public class Hallo {  
    public static void main(String[] args) {  
        System.out.println("Dag 1 EO-ICT !!!");  
    }  
}
```

STIJLFOUTEN

- Stijlregel 1 : Naam van een **klasse** steeds met een **Hoofdletter** beginnen! Is ook het geval voor ALLE andere klassen in de java klassenbibliotheek!!! – (bvb. System, Math, String, ...)
- Stijlregel 2 : Laat elke Java **methodenaam** beginnen met een **kleine letter** – Is ook het geval voor ALLE andere methoden in de java klassenbibliotheek!!! – (bvb. println, ...)
- Stijlregel 3 : Laat elke Java **variabele** beginnen met een **kleine letter**
- Stijlregel 4 : Gebruik **CamelCasing** voor lange namen :
klasse KennisMaking, EnergieVerbruik, methodebeschrijfJezelf, ...

COMPILATIEFOUTEN (SYNTAXFOUTEN)

Compilatiefouten zijn fouten in de **code van het programma** = syntaxfouten (compileren doen we met javac xxx.java)

RUNTIME FOUTEN (SEMANTISCHE FOUTEN)

Een voorbeeld van een **semantische fout** is “wanneer je programma een error geeft omdat je **als gebruiker een foute input** hebt ingegeven”. Voorbeeld programma vraagt **getal (int)** maar user geeft **tekst** als input (**String**).

LABO 1

```
System.out.println("tekst hier");
```

HFST 2 DATA TYPES EN OPERATIES, DEEL 1

DECLARATIE VAN EEN VARIABLE

Een variabele is een naam voor een locatie in het geheugen. In Java moet elke variabele gedeclareerd worden vooral deze kan gebruikt worden. Een declaratie specificeert van welk type de data is die op die bepaalde geheugenlocatie kan bijgehouden worden. In dit voorbeeld wordt er plaats in het geheugen gezocht om 2 gehele getallen bij te houden. Java is een sterk getypeerde taal. (+ assignment = het toekennen van een waarde aan een variable.)

```
int var1; // declaratie van een
           variabele van type int
int var2; // declaratie van een tweede
           variabele van type int
```

INT VERSUS DOUBLE

Gehele Deling vs Reële deling: een gehele deling geeft het gehele resultaat na deling terug zonder de rest!

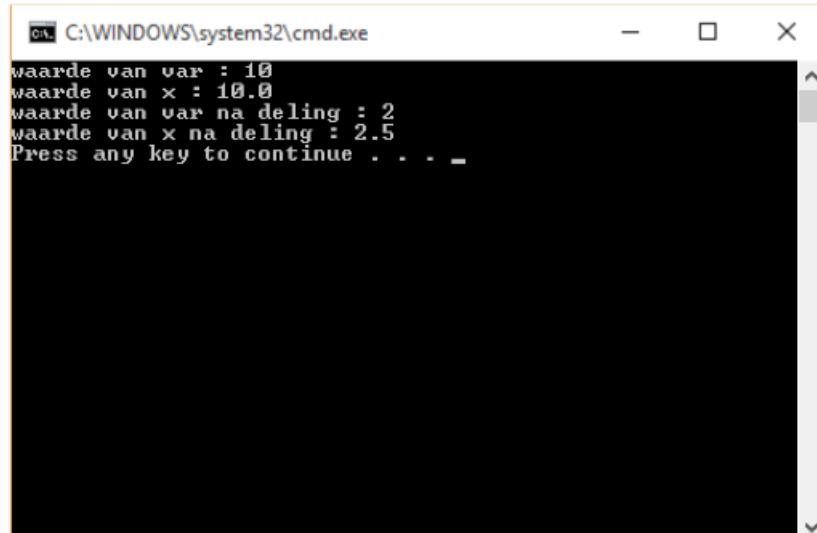
```
class IntVsDouble {
    public static void main (String args[]){
        int var; // gehele variabele
        double x; // reële variabele

        var = 10;
        x = 10.0;

        System.out.println("waarde van var : " + var);
        System.out.println("waarde van x : " + x);

        var = var / 4;
        x = x / 4;

        System.out.println("waarde van var na deling : " + var);
        System.out.println("waarde van x na deling : " + x);
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
waarde van var : 10
waarde van x : 10.0
waarde van var na deling : 2
waarde van x na deling : 2.5
Press any key to continue . . .
```

VARIABELEN EN HUN NAMEN

- Een **identifier** is de keuze van **een naam die je geeft aan een variabele** (of zie later een methode)
- een identifier kan starten met een letter, underscore of dollar teken (nooit starten met een cijfer)
- stijlfpraak is om alleen kleine letters te gebruiken
- java is case-sensitive : eenVar is niet hetzelfde dan eenvar. Volgens de stijlregels moet het echter eenVar zijn.
- gebruik geen keywords, of reeds bestaande namen uit de library bvb. Println

PRIMITIEVE VERSUS OBJECT TYPES

Primitieve types zijn toegevoegd aan de taal omwille van efficiëntie redenen. Zij zijn niet afgeleid van een klasse maar gewoon opgebouwd aan de hand van binaire waarden.

Object types zijn worden gemaakt aan de hand van Java klassen. Dit omvat:

- alle Java bibliotheek klassen (<https://docs.oracle.com/javase/8/docs/api/>) System, Math, String, ...
- waaronder ook de Java Wrapper klasse Byte, Short, Integer, Long, Float, Double, Boolean, Character, BigDecimal, BigInteger, ...
- elke klasse die je zelf definieert (zie verder ...)

ER ZIJN 8 PRIMITIEVE TYPES

Type	waarden van dit type
boolean	binaire waarden : true of false
byte	een geheel getal van max. 8 bits
short	een geheel getal van max. 16 bits
int	een geheel getal van max. 32 bits
long	een geheel getal van max 64 bits
char	een karakter
float	een reëel getal met enkele precisie (32 bit)
double	een reëel getal met dubbele precisie (64 bit)

DE GEHELE TYPES (**BSIL**)

type	#bits	bereik
byte	8 bits	$-128 \text{ tot } 127$ of $-2^7 \text{ tot } (2^7 - 1)$
short	16 bits	$-32.768 \text{ tot } 32.767$ of $-2^{15} \text{ tot } (2^{15} - 1)$
int	32 bits	$-2^{31} \text{ tot } (2^{31} - 1)$
long	64 bits	$-2^{63} \text{ tot } (2^{63} - 1)$

DE REËLE TYPES

type	#bits	bereik
float	32 bit	tekenbit (#1), mantisse (#23) en exponent (#8)
double	64 bit	tekenbit (#1), mantisse (#52) en exponent (#11)

KARAKTERS

type	#bits	bereik
char	16 bit	0 tot 65.536

decimaal	karakter	decimaal	karakter	decimaal	karakter
32	spatie	64	@	96	,
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(72	H	104	h
41)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	
61	=	93]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

ESCAPE SEQUENCES

Escape Sequentie	beschrijving
\ '	enkele quote
\ "	dubbele quotes
\\	backslash
\r	carriage return : begin van de lijn
\n	nieuwe lijn
\f	form feed : volgende pagina
\t	horizontale tab
\b	backspace
\uxxxx	hexadecimale waarde met xxxx de hex constante

SCANNING

De standaard manier om te lezen van de console is door gebruik te maken van den **Scanner klasse**. Met behulp van de Scanner klasse kan text input opgebroken worden in stukken volgens een delimiter zoals een spatie. De **text input** kan van **verschillende bronnen** komen zoals een gewone String maar ook **System.in**.

```
String input = "1 2 3 ";  
Scanner s = new Scanner(input);
```

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

IMPORT

Merk op : de Scanner klasse wordt niet automatisch ingeladen zoals bvb. System of Math dus moet je dit zelf doen via het import keyword, helemaal bovenaan je code

```
import java.util.Scanner;
```

```
import java.util.Scanner;

public class ScanString {
    public static void main(String args[]){
        String input = "1 2 3 ";
        Scanner s = new Scanner(input);
        System.out.println(s.nextInt());
        System.out.println(s.nextInt());
        System.out.println(s.nextInt());

        input = "one two three ";
        s = new Scanner(input);
        System.out.println(s.next());
        System.out.println(s.next());
        System.out.println(s.next());
    }
}
```

```
        input = "one_two_three";
        s = new Scanner(input).useDelimiter("_");
        System.out.println(s.next());
        System.out.println(s.next());
        System.out.println(s.next());

        s.close();
    }
}
```

LABO 2

Oefening 8:

RekeningNummer Gebruik de Scanner klasse om je bankrekeningnummer in te lezen. Opgelet, een geldig bankrekeningnummer wordt voorafgegaan door een IBAN code, bvb IBAN BE54 1234 5678 9012. Print daarna de IBAN code en het eigenlijke rekeningnummer afzonderlijk af.

```
Geef je rekeningnummer met IBAN code :  
IBAN BE54 1234 5678 9012
```

```
De IBAN code is : BE54  
Het rekeningnummer : 1234 5678 9012
```

```
1  /**  
2  * @author Ruben Van der Kelen  
3  * @version januari 2019  
4  */  
5  
6  import java.util.Scanner;  
7  
8  public class RekeningNummer{  
9      public static void main (String[] args){  
10         Scanner scan = new Scanner(System.in);  
11         System.out.println("Geef je rekeningnummer met IBAN code: ");  
12         String input = scan.nextLine();  
13         scan = new Scanner(input);  
14         scan.next();  
15         System.out.println("De IBAN code is: " + scan.next());  
16         System.out.println("Het rekeningnummer: " + scan.next() + " " + scan.next() + " " + scan.next());  
17     }  
18 }
```

HFST 2 DATA TYPES EN OPERATIES, DEEL 2

PRIMITIEVE OPERATOREN (NIET HETZELFDE ALS PRIMITIEVE TYPES: ZIE EERDER)

Java kent volgende types operatoren :

Arithmetic Rekenkundige operatoren : +, -, *, /, %, ++, --

Relational Vergelijkingsoperatoren : ==, !=, >, <, ≥, ≤

Logic Logische operaties : & (AND), | (OR), ^ (XOR),
&& (lazyAND), || (lazyOR), ! (NOT)

Bitwise Bit operatoren : zie later

Andere Assignement

=, +=, -=, *=, /=, %=, &=, |=, ^=
Ternaire operator

Geef de waarde van variabele *c* nadat volgende assignments uitgevoerd werden. Ga er voor elke expressie vanuit dat **int a = 10; int b= 3; int c = 2;**

expressie
c = a + b;
c = a - b;
c = a * b ;
c = a / b ;
c = a % b;
c ++ ; ++c;
c -- ; --c
System.out.println(c++);
System.out.println(++c);
--c+2 ;

waarde van c
13
7
30
3
1
3
1
2
3
3

Ga er voor elke expressie vanuit dat **int a = 10; int b= 3; int c = 2;**

expressie
c += b;
c -= b;
c *= b ;
c /= b ;
c %= b;
a > b
a < b
a ≥ b
a ≤ b
a == b
a != b

waarde van c
5
-1
6
0
2
true
false
true
false
false
true

DE TERNAIRE OPERATOR

```
public class TernaireOperator {  
    public static void main (String[] args){  
        int a = 10, b = 3, c = 2, grootste;  
        boolean res;  
  
        res = a > b;  
        System.out.println("res : " + res);  
  
        grootste = (a > b ? a : b);  
        System.out.println("grootste : " +  
            grootste);  
        System.out.println(a==b ? "gelijk" :  
            "verschillend");  
    }  
}
```

← ternaire operator geeft a weer indien true
en b indien false

PRIMITIEVE OPERATOREN: VERVOLG

Geef de waarde van variabele *c* nadat volgende assignments uitgevoerd werden. Ga er voor elke expressie vanuit dat **double a = 1.2, b = 0.4, c = 2.0**

expressie
c = a + b;
c = a - b;
c = a * b ;
c = a / b ;
c = a % b;
c ++ ; ++c;
c -- ; --c
--c+2 ;

waarde van c
1.6
0.8
0.48
3.0
-nvt-
3.0
1.0
3.0

Ga er voor elke expressie vanuit dat **double a = 1.2, int b= 0.4, c = 2.0;**

expressie
c += b;
c -= b;
c *= b ;
c /= b ;
c %= b;
a > b
a < b
a ≥ b
a ≤ b
a == b
a != b

waarde van c
2.4
1.6
0.8
5.0
-nvt-
true
false
true
false
false
true

expressie
x == y ;
x != y ;
x && y;
x y;
! x
x && !y x
!(x && y)
c2 = c1 + 2;
c2 = 'A' + 32;
c2 = '0' + 5;
boolean b = c1 > 'A' ;
System.out.println('5');
System.out.println((int)'5');
System.out.println((char)88);

waarde
false
true
false
true
false
true
true
'c'
'a'
'5'
true
5
53
X

VOLGORDE VAN DE BEWERKINGEN

prioriteit	operator	beschrijving
1	(expr)	haakjes
	++ --	increment- en decrementoperator (unair)
	-	unaire minoperator
	!	logische negatie
	(type)	castoperator
2	* / %	rekenkundige maaloperatoren
3	+ -	rekenkundige plusoperatoren
	+	stringconcatenatie
4	< <= >= >	relationele operatoren
5	== !=	(on)gelijkheidsoperatoren
6	&&	logische AND
7		logische OR
8	=	toekenningoperator
	*= /= %= += -=	rekenkundige toekenningsoveratoren

CONCATENATIE

String > int > char

expressie
1 + 'a'
'a' + 1
"" + 'a'
1 + ""
"! " + 'a' + 1
"!! " + ('a' + 1)
"" + (char)('a' + 1)

type	waarde
int	98
int	98
String	"a"
String	"1"
String	"! a1"
String	"!! 98"
String	"b"

IMPLICIETE VERSUS EXPLICIETE CAST

```
byte b = 20;  
int kopie = b; // impliciet wordt b omgezet naar een type int
```

Dit kan natuurlijk alleen maar als :

- de twee types compatibel zijn : bvb. van **int** naar **boolean** kan niet, maar **int** naar **float** kan wel
- het type waaraan je toekent moet meer geheugencapaciteit hebben : wanneer je een **int** in een **byte** stockeert verlies je gegevens.

toch kan dit maar dan met een expliciete cast

```
kopie = kopie + 256 ;  
byte b2 = (byte) kopie; // expliciete omzetting
```

expressie
5 - 2 / 4 - 2
(double) (11 / 2)
(double) 11 / 2
11 / (double) 2
1 < 2 && !(2 < 1)
(int) 1.0 / 2 + 0.5
(int) (1.0 / 2) + 0.5
(int) (1.0 / 2 + 0.5)

type	waarde
int	3
double	5.0
double	5.5
double	5.5
boolean	true
double	0.5
double	0.5
int	1

LABO 3

Oefening 13:

Datum2 Schrijf een programma dat een datum kan afprinten in volgende vorm dd-mm-jjjj. Gebruik hierbij 3 variabelen van type int om respectievelijk de dag, maanden jaar bij te houden. Gebruik nu de ternaire operator om na te gaan of een 0 voor de dag of maand geplakt moet worden.

```
De opgegeven datum is : 01-09-2017
```

```

1  /**
2  * @author Ruben Van der Kelen
3  * @version januari 2019
4  */
5
6  import java.util.Scanner;
7
8  public class Datum2{
9      public static void main (String[] args){
10         int dag, maand, jaar;
11         String dd, mm, jjjj;
12         String nulToevoegen;
13
14         Scanner scan = new Scanner(System.in);
15         System.out.println("Geef de dag van vandaag in: ");
16         dag = scan.nextInt();
17         System.out.println("Geef de huidige maand in: ");
18         maand = scan.nextInt();
19         System.out.println("Geef het huidige jaar in: ");
20         jaar = scan.nextInt();
21
22         nulToevoegen = (dag <= 9 ? "0" : "");
23         dd = "" + nulToevoegen + dag;
24         nulToevoegen = (maand <= 9 ? "0" : "");
25         mm = "" + nulToevoegen + maand;
26         nulToevoegen = (jaar <= 999 ? "0" : "");
27         jjjj = "" + nulToevoegen + jaar;
28
29         System.out.println("De opgegeven datum is : " + dd + "-" + mm + "-" + jjjj);
30     }
31 }

```

HFST 2 DATA TYPES EN OPERATIES, DEEL 3

CONSTANTEN PI EN E

Voorbeeld Math.PI

```

int straal = 5;
double cirkelOmtrek = 2 * Math.PI * straal;

```

ZELF CONSTANTEN DEFINIËREN

Via het keyword **final** kan je in java een variabele constant maken, d.i. de waarde ervan kan niet veranderd worden tijdens het programma. Je kan een constante alleen bij initialisatie een waarde geven.

Volgens de stijlregels schrijf je constanten **steeds volledig in hoofdletters** !

NUTTIGE METHODEN IN MATH

methode definitie	oproep
double Math.abs (double d)	double res; int i; long l; res = Math.abs(-7.25); 7.25
long Math.round (double d)	l = Math.round(25.1); 25 l = Math.round(25.8); 26
int Math.round (float f)	i = Math.round(-7.25f); -7
double Math.ceil (double d)	res = Math.ceil(25.1); 26.0 res = Math.ceil(25.8); 26.0
double Math.floor (double d)	res = Math.floor(25.1); 25.0 res = Math.floor(25.8); 25.0

methode definitie	oproep
double Math.cos (double x)	double res ; res = Math.cos(1); 0.54
idem : sin, tan, acos, asin, atan	
double Math.pow (double x, double y)	res = Math.pow(5, 2); 25.0 res = Math.pow(2, 8); 256.0
double Math.sqrt (double x)	res = Math.sqrt(49); 7.0
double Math.min (double x, double y)	res = Math.min(3, 5); 3.0
double Math.max (double x, double y)	res = Math.max(3, 5); 5.0

MATH.RANDOM()

methode definitie	oproep
double Math.random()	double res; res = Math.random(); $res \in [0, 1[$

Genereer een random getal in [0 10 [

```
double dtoeval = Math.random() * 10;
```

Genereer een random geheel getal in [0 10 [

```
int itoeval = (int)(Math.random() *  
    10);  
int itoeval2 =  
    (int)(Math.round(Math.random() *  
    9));
```

Merk op : getal 0 en 9 krijgen minder kans om gegenereerd te worden via de *round()* methode.

Genereer een random geheel getal in [1 10]

```
itoeval = (int)(Math.random() * 10 +  
    1);  
itoeval2 =  
    (int)(Math.round(Math.random() * 9  
    + 1));
```

Merk op : getal 1 en 10 krijgen minder kans om gegenereerd te worden via de *round()* methode.

IN JAVA, STRINGS ZIJN OBJECTEN!

De manier om in Java objecten aan te maken is door gebruik te maken van de operator *new*. Strings kunnen dus als volgt aangemaakt worden :

```
String muppet1 = new String("Fozzy");
```

maar door gebruik te maken van string literals kan ook het volgende :

```
String muppet2 = "Gonzo";  
String muppet3 = "Kermit";
```

IN JAVA WORDT ER GETELD VANAF 0!

methode definitie	oproep
char charAt(int i)	char c ; int i c = muppet1.charAt(0); 'F' c = muppet2.charAt(2); 'n' c = muppet3.charAt(5); 't' c = muppet2.charAt(5); —runtime error— String index out of bounds
int length()	i = muppet1.length(); 5 i = muppet2.length(); 5 i = muppet3.length(); 6

methode definitie	oproep
int indexOf(char c)	int i i = muppet1.indexOf('y'); 4 i = muppet1.indexOf('z'); 2 i = muppet2.indexOf('*'); -1
int indexOf(String s)	i = muppet2.indexOf("zo"); 3
int lastIndexOf(char c)	i = muppet1.lastIndexOf('z'); 3

STRINGS VERGELIJKEN

methode definitie	oproep
boolean equals(String s)	boolean b b = muppet1.equals(muppet3); false b = muppet1.equals(muppet1); true b = muppet1.equals("Fozzy"); true
== operator	b = muppet1 == muppet1; true b = muppet1 == "Fozzy"; false b = muppet2 == "Gonzo"; true

STRINGS KAN JE NIET MUTEREN

Dit wil zeggen dat je de rij van karakters intern niet kan wijzigen :

muppet1.charAt(1) ^{NOK} \neq 'u';

Er is wel een **replace** methode voorzien, maar die maakt een nieuwe string aan.

methode definitie	oproep
String replace(char c1, char c2)	String mup; mup = muppet1.replace('o','u') ; → "Fuzzy"
String replace(String s1, String s2)	mup = muppet1.replace("Fozz","Pigg") ; → "Piggy"

ANDERE NUTTIGE METHODEN

methode definitie	oproep
	boolean b; String mup;
boolean startsWith(String s)	b = muppet3.startsWith("Ker"); → true
String toUpperCase()	mup = muppet3.toUpperCase(); → "KERMIT"
String toLowerCase()	mup = muppet3.toLowerCase(); → "kermit"
String substring(int i)	mup = muppet1.substring(1); → "ozzy"
String substring(int i, int j)	mup = muppet2.substring(1,3); → "on"

DE WRAPPERKLASSEN

String → byte	byte b = Byte.parseByte(s);
String → short	short sh = Short.parseShort(s);
String → int	int i = Integer.parseInt(s);
String → long	long l = Long.parseLong(s);
String → float	float f = Float.parseFloat(s);
String → double	double d = Double.parseDouble(s);

byte → String	String s = Byte.toString(b);
short → String	String s = Short.toString(sh);
int → String	String s = Integer.toString(i);
long → String	String s = Long.toString(l);
float → String	String s = Float.toString(f);
double → String	String s = Double.toString(d);

LABO 4

Int i, Char ch, String st;

Char naar int : `i = (int) ch;` expliciete cast

Oefening 10:

KlantKey: Schrijf een programma dat op basis van een ingelezen achternaam en voornaam van een klant een klantcode genereert op volgende manier :

- De key begint steeds met de prefix "ODISEE"
- De key bestaat alleen uit hoofdletters
- Het volgende karakter bepaal je als volgt : neem de decimale waarde van de derde letter van de voornaam (in hoofdletters) tel hier 5 bij op en zet terug om naar een letter (let op: zorg ervoor dat je nog steeds een hoofdletter bekomt, dus als je karakter > 'Z' moet je er 26 aftrekken)
- Voor het volgende karakter neem je de laatste letter van de familienaam (als hoofdletter)
- Tenslotte eindig je de code door het volgende : neem de eerste letters van de voor en achternaam (als hoofdletter). Vermenigvuldig hun numerieke waarden, dit getal zet je om naar een string en plak je aan decode.

Dit geeft het volgende voor de naam : Jan Janssens (Merk op de voornaam bestaat minstens uit 3 karakters, de achternaam kan samengesteld zijn bvb Van de winkel beschouw dit als 1 woord, met spaties.)

- Het 7de karakter van de code is gebaseerd op de derde letter van de voornaam hier dus n. Hiervan nemen we $N + 5$ ofwel $78 + 5 = 83$ wat volgens de ASCII tabel de letter 'S' geeft.
- Het volgende karakter is eveneens 'S'
- Het produkt van 'J' met zichzelf hier geeft : $74 \times 74 = 5476$

- De code voor Jan Janssens is dan als volgt : ODISEESS5476

```

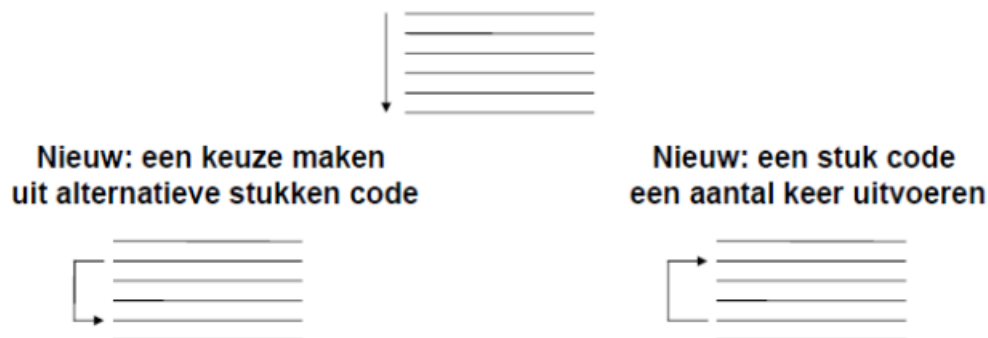
24 public class KlantKey {
25     public static void main(String[] args) {
26         String naam, naamHoofdLetters, voorNaam, achterNaam, klantKey;
27         String c1, c2, c3;
28         char clx, cly, clz, c5, c6;
29         int i1, i2, i3;
30
31         klantKey = "ODISEE";
32         Scanner scan = new Scanner(System.in);
33         System.out.println("Geef je voor- en achternaam: ");
34         naam = scan.nextLine();
35         naamHoofdLetters = naam.toUpperCase();
36         voorNaam = naamHoofdLetters.substring(0, naam.indexOf(" "));
37         achterNaam = naamHoofdLetters.substring(naam.indexOf(" "), naam.length());
38
39         c1 = naamHoofdLetters.substring(2, 3);
40         clx = c1.charAt(0);
41         cly = (char) (clx + 5);
42         clz = (cly > 'Z' ? (char) (cly - 26) : cly);
43         c2 = achterNaam.substring(achterNaam.length() - 1, achterNaam.length());
44         c5 = voorNaam.charAt(0);
45         c6 = achterNaam.charAt(0);
46         i1 = (int) c5;
47         i2 = (int) c6;
48         i3 = i1 * i2;
49         c3 = "" + i3;
50         klantKey = klantKey + clz + c2 + c3;
51
52         System.out.println(klantKey);
53     }
54 }
55
56

```

HFST 3 CONTROLE STRUCTUREN IN JAVA, DEEL 1

DE CONTROL FLOW

Binnen een methode worden instructies normaal sequentieel uitgevoerd:



De selecties:

- if-structuur
- if/else-structuur
- meervoudige if/else-structuur
- switch-structuur
- nesten van selectiestructuren

De iteraties:

- while-structuur
- do/while-structuur
- for-lus
- nesten van iteratiestructuren

NESTED-IFS

De java expressie binnen de **if** of **else** tak mag opnieuw een if-structuur zijn, en de java expressie binnen deze laatste if of else tak mag opnieuw een if-structuur zijn, en de java expressie binnen deze laatste if of else tak mag opnieuw een if-structuur zijn, en de

ALGEMEEN: MEERVOUDIGE KEUZE

```
if (booleaanse uitdrukking) {  
    ...  
}  
else if (booleaanse uitdrukking) {  
    ...  
}  
...  
else {  
    ...  
}
```

ALGEMEEN: SWITCH

```
Vorige blok code;  
  
switch (var) {  
    case waarde1:  
        //instructies1  
        break;  
    case waarde2:  
        //instructies2  
        break;  
    case waarde3:  
        //instructies3  
        break;  
    ...  
    default:  
        //instructies  
        break;  
}  
  
Volgende blok code;
```

Als men **de break tussen de cases zou weglaten**: meerdere cases met dezelfde output: tot aan de eerste break.

LABO 5

Automatisch aanvullen in netbeans: **ctrl + spatie**

Oefening 9:

AMPM: Schrijf een programma dat Europese tijd (24 uurs notatie) kan omzetten naar Amerikaanse tijd (AM/PM) en omgekeerd.

$8AM \rightarrow 8$
 $4PM \rightarrow 16$
 $21 \mapsto 9PM$
 $12 \rightarrow 12PM$

Merk op: AM = (0u : : 11u) , PM = (12u : : 23u) De enige input die je programma krijgt is bvb. 11 of 11AM , het programma gaat zelf bepalen in welke richting de conversie moet gebeuren.

```
14 public class AMPM {  
15     public static void main(String[] args) {  
16         Scanner scan = new Scanner(System.in);  
17         System.out.println("Geef een tijd in om te converteren: ");  
18         String tijd = scan.next();  
19         if (tijd.length() < 3){  
20             int tijdInt = Integer.parseInt(tijd);  
21             switch (tijdInt){  
22                 case 0:  
23                 case 1:  
24                 case 2:  
25                 case 3:  
26                 case 4:  
27                 case 5:  
28                 case 6:  
29                 case 7:  
30                 case 8:  
31                 case 9:  
32                 case 10:  
33                 case 11: tijd += "AM";  
34                 break;  
35                 case 12:  
36                 case 13:  
37                 case 14:  
38                 case 15:  
39                 case 16:  
40                 case 17:  
41                 case 18:  
42                 case 19:  
43                 case 20:  
44                 case 21:  
45                 case 22:  
46                 case 23: tijdInt = tijdInt - 12;  
47                     tijd = tijdInt + "PM";  
48                 break;  
49             }  
50             System.out.println("De tijd na conversie (Europees > Amerikaans) is: " + tijd);  
51     }
```

```

52     else {
53         String anteOfPost = tijd.substring(2, 4).toUpperCase();
54         String tijdUitString = tijd.substring(0, 2);
55         int tijdInt = Integer.parseInt(tijdUitString);
56         switch (anteOfPost){
57             case "AM": tijd = "" + tijdInt;
58             break;
59             case "PM": tijdInt = tijdInt + 12;
60             tijd = "" + tijdInt;
61             break;
62         }
63         System.out.println("De tijd na conversie (Amerikaans > Europees) is: " + tijd);
64     }
65 }
66 }
67

```

HFST 3 CONTROLE STRUCTUREN IN JAVA, DEEL 2

Deze bevat enkel herhaling.

LABO 6

Oefening 7:

MinMaxKlinkers: Schrijf een programma waarbij de gebruiker N woorden intypt en het woord met het minst aantal klinkers en het woord met het meest aantal klinkers van deze N woorden opnieuw uitprint naar het scherm. (Bij ex aequo print je het eerste woord af) Het getal N definieer je als constante.

```

14 public class MinMaxKlinkers {
15     public static void main(String[] args) {
16         Scanner scan = new Scanner(System.in);
17         System.out.println("Hoeveel woorden wil je ingegeven: ");
18         final int aantalWoorden = scan.nextInt();
19         String kleinsteWoord = "";
20         String grootsteWoord = "";
21         int kleinsteAantalKlinkers = 0;
22         int grootsteAantalKlinkers = 0;
23
24         for (int i = 0; i < aantalWoorden; i++){
25             System.out.println("Geef een woord in: ");
26             String woord = scan.next();
27             int aantalKlinkers = 0;
28             int woordLengte = woord.length();
29
30             if (i == 0){
31                 kleinsteWoord = woord;
32                 grootsteWoord = woord;
33                 for (int j = 0; j <= woordLengte - 1; j++){
34                     char Letter = woord.charAt(j);
35                     if (Letter == 'a' || Letter == 'e' || Letter == 'i' || Letter == 'o' || Letter == 'u' || Letter == 'y'){
36                         aantalKlinkers++;
37                     }
38                 }
39             }
40             else{
41                 aantalKlinkers = kleinsteAantalKlinkers;
42                 aantalKlinkers = grootsteAantalKlinkers;
43             }
44             else{
45                 for (int j = 0; j <= woordLengte - 1; j++){
46                     char Letter = woord.charAt(j);
47                     if (Letter == 'a' || Letter == 'e' || Letter == 'i' || Letter == 'o' || Letter == 'u' || Letter == 'y'){
48                         aantalKlinkers++;
49                     }
50                 }
51             }
52             else{
53                 if (aantalKlinkers < kleinsteAantalKlinkers){
54                     kleinsteWoord = woord;
55                 }
56                 else if (aantalKlinkers > grootsteAantalKlinkers){
57                     grootsteWoord = woord;
58                 }
59             }
60         }
61     }
62 }
63
64 System.out.println("Het woord met het minst aantal klinkers is: " + kleinsteWoord);
65 System.out.println("Het woord met het grootste aantal klinkers is: " + grootsteWoord);
66 }
67 }
68

```

Oefening 8:

AfEnOp: Schrijf een programma dat een ingelezen woord eerst afbouwt en vervolgens weer opbouwt als volgt :

```
Welk woord wil je af-en opbouwen?  
MARSEPEIN
```

```
MARSEPEIN  
MARSEPEI  
MARSEPE  
MARSEP  
MARSE  
MARS  
MAR  
MA  
M  
MA  
MAR  
MARS  
MARSE  
MARSEP  
MARSEPE  
MARSEPEI  
MARSEPEIN
```

```
14 public class AfEnOp {  
15     public static void main(String[] args) {  
16         Scanner scan = new Scanner(System.in);  
17         System.out.println("Geef een woord in: ");  
18         String woord = scan.next();  
19         char leeg = '\0';  
20         String woordAfbouwen = "";  
21         for (int i = woord.length() - 1; i > 0; i--){  
22             woordAfbouwen = woord.substring(0, i);  
23             System.out.println(woordAfbouwen);  
24         }  
25         String woordOpbouwen = "";  
26         for (int i = 1; i <= woord.length(); i++){  
27             woordOpbouwen = woord.substring(0, i);  
28             if (woordOpbouwen.length() > 1 ){  
29                 System.out.println(woordOpbouwen);  
30             }  
31             else{  
32                 leeg = woord.charAt(i-1);  
33             }  
34         }  
35     }  
36 }  
37  
38
```

Opletten!!!

Vb. String woord = "abc";

Int lengte = woord.length(); Geeft 3 terug

Dus als je wilt werken met deze int lengte, moet je deze altijd met 1 aftrekken! (Java begint te tellen vanaf 0)

Vb. laatste karakter van abc: char laatste = woord.charAt(lengte - 1); !!!

HFST 3 CONTROLE STRUCTUREN IN JAVA, DEEL 3

HERHALEN ZOLANG DE CONDITIE WAAR IS ... DO/WHILE OF WHILE LUS

Algemene vorm van een while-lus:

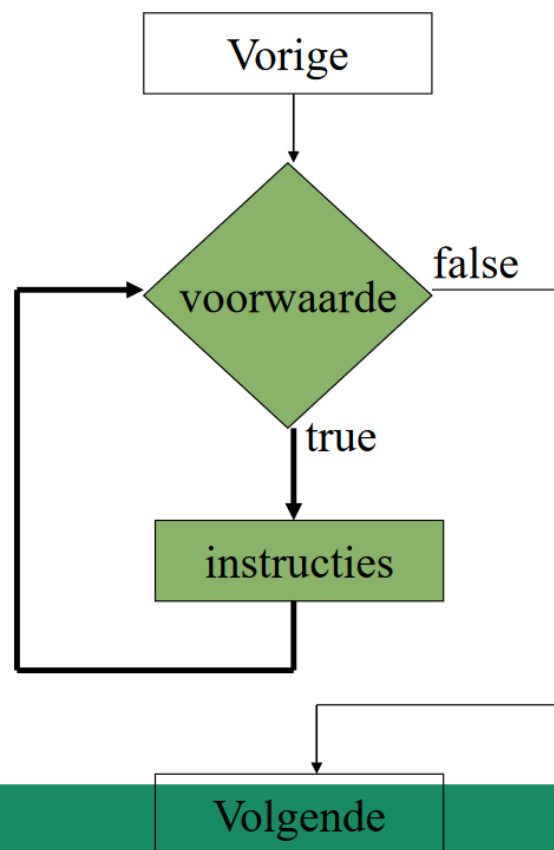
```
while (booleaanse uitdrukking) {  
    ...  
}
```

Algemene vorm van een do/while-lus:

```
do {  
    ...  
} while (booleaanse uitdrukking);
```

ITERATIES (WHILE)

```
Vorige;  
  
while (voorwaarde) {  
    //instructies  
}  
  
Volgende;
```

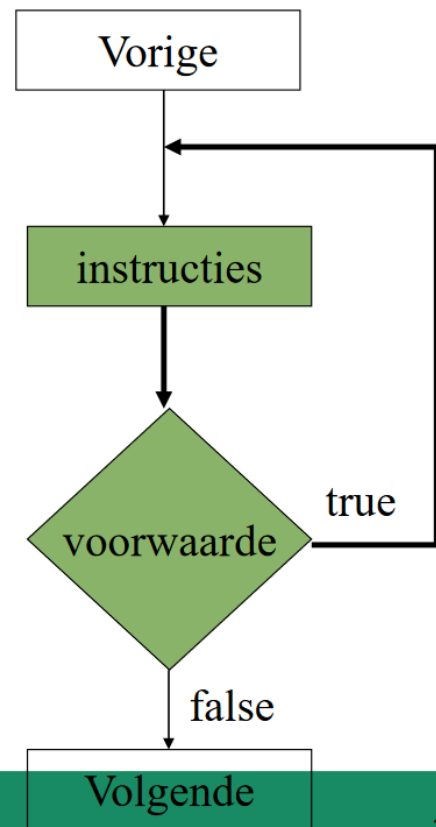


→ De voorwaarde moet bepalend zijn

→ Instructies worden mogelijks niet uitgevoerd

ITERATIES (DO-WHILE)

```
Vorige;  
  
do {  
    //instructies  
} while (voorwaarde);  
  
Volgende;
```



→ De voorwaarde moet bepalend zijn

→ Instructies worden minimaal 1x uitgevoerd

WANNEER WELKE LOOP GEBRUIKEN?

- Gebruik de **for – lus** wanneer je **vooraf weet hoe vaak** je zal moeten herhalen
- Gebruik **do – while** wanneer je **minstens 1x** het codeblok moet uitvoeren
- Gebruik **while** wanneer je **vooraf helemaal niet weet** of en hoelang je zal moeten herhalen

LABO 7

Oefening 4:

ProcentKlinkers: Schrijf een programma dat een woord inleest en bepaalt uithoeveel procent klinkers het bestaat.

```
14 public class ProcentKlinkers {  
15     public static void main(String[] args) {  
16         Scanner scan = new Scanner(System.in);  
17         System.out.println("Geef een willekeurige woord in:");  
18         String woord = scan.next();  
19         double aantalLetters = woord.length() - 1;  
20         int letterPlaats = 0;  
21         double aantalKlinkers = 0;  
22         do{  
23             char letter = woord.charAt(letterPlaats);  
24             if (letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter == 'u' || letter == 'y'){  
25                 aantalKlinkers++;  
26             }  
27             else{  
28                 letterPlaats++;  
29             }  
30             letterPlaats++;  
31         }while(letterPlaats < aantalLetters);  
32         System.out.println(aantalKlinkers);  
33         System.out.println(woord.length());  
34         double percentage = ((aantalKlinkers / woord.length()) * 100);  
35         System.out.println("Het percentage aan klinkers is: " + percentage + " %");  
36     }  
37 }  
38
```

⇒ Zoveel mogelijk dubbel gebruiken voor kommagetallen en percentageberekeningen.

Oefening 5:

KortsteTekst: Schrijf een programma dat de kortste van een reeks ingelezen strings bepaalt. De ingave stopt door "stop" in te geven.

```
14 public class KortsteTekst {
15     public static void main(String[] args) {
16         Scanner scan = new Scanner(System.in);
17         String s1;
18         String kortsteString = "";
19         String stop = "stop";
20         int i = 0;
21
22         do{
23             System.out.println("Geef een zin in (of stop om te stoppen): ");
24             s1 = scan.next();
25             if (s1.toLowerCase().equals(stop)) {
26                 }
27             else if(i == 0){
28                 kortsteString = s1;
29             }
30             else{
31                 if(s1.length() < kortsteString.length()){
32                     kortsteString = s1;
33                 }
34                 else{
35                     }
36             }
37             i++;
38             }while(!s1.equals(stop));
39             System.out.println(kortsteString + " is de kortste string");
40         }
41     }
42 }
43 }
```

Do{

}while(!s1.equals("stop")) om te kijken of een string NIET gelijk is aan een de tekst. Zie de !.

LABO 8

Om ervoor te zorgen dat de while-lus stopt: **woord = scan.nextLine();** i.p.v **woord = scan.next();**

HFST 4 KLASSEN, OBJECTEN EN METHODEN + VERVOLG METHODEN (DEEL HFST 6), DEEL 1

WAT IS EEN KLASSE?

Elk Java programma bestaat uit de definitie van enkele klassen in combinatie met een main methode.

Een **klasse** zelf is een blauwdruk, i.e. een abstracte beschrijving van een nieuw type, en drukt uit hoe een element van dat type (wat men een object noemt) gebouwd moet worden. Het legt tevens vast welke **data** het **object** bevat en welke **operaties** op die data toegelaten zijn. De operaties worden beschreven aan de hand van **methoden**, de data aan de hand van variabelen. Beide (methoden en variabelen) worden **members** van de klasse genoemd. De data members worden ook **data velden**, **fields** of **instance variabelen** genoemd.

ALGEMENE VORM VAN EEN KLASSE

```
public class NaamVanDeKlasse{  
    // Een klasse bestaat uit data members en methoden  
  
    // declaratie van de instance variabelen  
    type var1;  
    type var2;  
    ...  
  
    // declaratie van de methoden  
    public returnType naamMethode1(parameters){  
        // body van methode1  
    }  
    public returnType naamMethode2(parameters){  
        // body van methode2  
    }  
    ...  
}
```

EEN NIEUW TYPE VOOR VOERTUIGEN

```
public class Voertuig{  
    int passagiers; // max aantal passagiers  
    int capaciteit; // max aantal liters  
    brandstof  
    double verbruik; // verbruik  
}
```

Merk op, op dit moment is alleen het nieuwe type zelf gedefinieerd. Er zijn nog geen objecten (elementen van het type voertuig) aangemaakt!

EEN PROGRAMMA MET 1 VOERTUIG OBJECT (EN 2 KLASSEN !)

```
public class DemoVoertuig {
// dit programma zal voertuigen aanmaken en gebruiken
public static void main(String[] args){
    Voertuig miniBus = new Voertuig();
    double afstand;
    //waarden toekennen aan de data
    miniBus.passagiers = 7;
    miniBus.capaciteit = 80;
    miniBus.verbruik = 6.0;

    //bereken de afstand die afgelegd kan worden met een volle tank
    afstand = miniBus.capaciteit * miniBus.verbruik;
    System.out.println("De miniBus kan " + miniBus.passagiers + " personen
        vervoeren en een afstand van " + Math.round(afstand * 100)/100.0 +
        "kilometers afleggen met een volle tank ");
}
}
```

EEN PROGRAMMA MET 2 VOERTUIGEN (EN 2 KLASSEN)

```
public class Demo2Voertuig {
// dit programma zal voertuigen aanmaken en gebruiken
public static void main(String[] args){
    Voertuig miniBus = new Voertuig();
    Voertuig _2pk = new Voertuig();
    double afstand1, afstand2;
    //velden van de miniBus
    miniBus.passagiers = 7;
    miniBus.capaciteit = 80;
    miniBus.verbruik = 6.0;

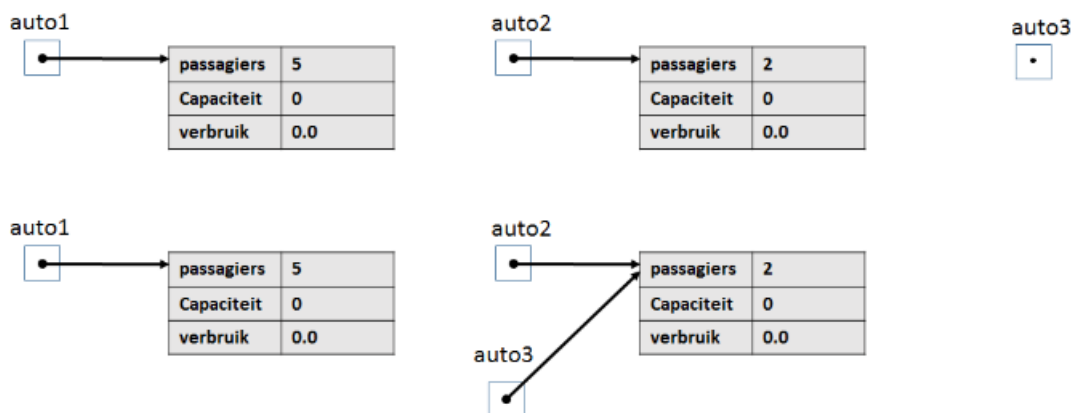
    //velden van de _2pk
    _2pk.passagiers = 4;
    _2pk.capaciteit = 30;
    _2pk.verbruik = 10.0;

    //bereken de afstand die afgelegd kan worden met een volle tank
    afstand1 = miniBus.capaciteit * miniBus.verbruik;
    afstand2 = _2pk.capaciteit * _2pk.verbruik;
    System.out.println("De miniBus kan " + miniBus.passagiers + " personen
        vervoeren en een afstand van " + Math.round(afstand1 * 100)/100.0 + "
        kilometers afleggen met een volle tank ");
    System.out.println("Het _2pk'tje kan " + _2pk.passagiers + " personen
        vervoeren en een afstand van " + Math.round(afstand2 * 100)/100.0 + "
        kilometers afleggen met een volle tank ");
}
```

KOPIËREN VAN OBJECTEN

```
public class Voertuigen2 {  
  
    public static void main(String[] args){  
        Voertuig auto1 = new Voertuig();  
        Voertuig auto2, auto3 ; // declaratie  
        auto2 = new Voertuig(); // ken een plaats uit het  
            geheugen toe  
        System.out.println(auto1 + " " + auto2); // auto3  
            heeft nog geen geheugenplaats !  
        auto1.passagiers = 5;  
        auto2.passagiers = 2;  
  
        auto3 = auto2; // laat auto3 naar dezelfde  
            geheugenplaats wijzen!  
        System.out.println(auto3);  
        System.out.println(auto3.passagiers); // -> 2  
    }  
}
```

ELK OBJECT HEFT ZIJN EIGEN DATA



METHODEN

- Methoden zijn subroutines (functies) die de data van je klasse kan manipuleren.
- Vaak is het ook de enige manier om (veilig) toegang te verlenen tot die data.
- Elke methode voert 1 welbepaalde taak uit.
- Een methode kan input krijgen via parameters.
- Een methode kan output teruggeven via een return
- Elke methode is volledig beschreven via zijn definitie, deze beschrijving geeft alle nodige informatie om een methode aan te roepen en het resultaat op te vangen

Algemene vorm van een methode

```
returnType methodeNaam (parameters) {  
    // body van de methode  
}
```

Definitie van een methode

```
returnType methodeNaam (parameters)
```

RETURN

Return is een voorbehouden Java keyword waarmee een methode de controle terug kan geven aan de oproeper en dus de methode beëindigt. Return geeft je methode **een exit-point**.

RETURN MET EEN WAARDE

return kan ook een waarde teruggeven aan de oproeper. Wanneer dit gebeurt moet deze waarde van hetzelfde type zijn als het returntype in de methodedefinitie van de methode. Wanneer de methode geen waarde teruggeeft dan staat het returntype van de methode op **void**. Een methode hoeft geen return te bevatten. In dit geval worden alle expressies van de body van de methode uitgevoerd tot aan de afsluitende accolade. Deze methode heeft altijd returntype **void**.

METHODEN MET PARAMETERS

Via **parameters** kan je input geven aan een methode. Deze parameters kunnen in de body van de methode gebruikt worden zoals een lokale variabele. Deze variabele krijgt pas een waarde toegekend bij oproep van deze methode. Men zegt dan dat de **formele parameter** de waarde van de **actuele parameter** krijgt bij een methode-oproep. Een methode kan meerdere parameters als input krijgen. (terwijl er max 1 returnwaarde kan zijn) Wanneer de oproep de controle teruggeeft aan de oproeper, verdwijnt de binding tussen formele en actuele parameter.

```
public class VoertuigParam{  
    int passagiers; // max aantal passagiers  
    int capaciteit; // max aantal liters brandstof  
    double verbruik; // verbruik  
  
    // bereken de afstand met een volle tank  
    public double afstand () {  
        double afstand = Math.round(capaciteit * verbruik * 100) / 100.0;  
        return afstand;  
    }  
  
    // gegeven de af te leggen weg, bereken de nodige brandstofhoeveelheid  
    public double berekenBrandstof(double km){  
        return (Math.round(km / verbruik * 100) / 100.0);  
    }  
}
```

CONSTRUCTOR METHODEN

Een constructor methode **initialiseert een object**. Deze methode heeft dezelfde naam als de naam van de klasse. Een constructormethode heeft wel **geen returntype**. Men maakt onderscheid tussen **de standaard (of default) constructor** en **geparameteriseerde constructoren**. De default constructor wordt automatisch voor je aangemaakt wanneer je een nieuwe klasse implementeert. Deze zal al je data velden initialiseren met hun default waarden (0, 0.0, false and null) Een geparameteriseerde constructor schrijf je zelf! Deze kan als input enkele parameters nemen, waarmee je de datavelden van de klasse zelf kan initialiseren. Let wel, wanneer je zelf expliciet een constructor schrijft zal er geen default constructor meer voor je gegenereerd meer worden!

```
public class VoertuigConstr{
    int passagiers; // max aantal passagiers
    int capaciteit; // max aantal liters brandstof
    double verbruik; // verbruik

    // default Constructor
    public VoertuigConstr(){
        passagiers = 0;
        capaciteit = 0;
        verbruik = 0.0;
    }

    // niet-default constructor
    public VoertuigConstr(int pas, int cap, double verb){
        passagiers = pas;
        capaciteit = cap;
        verbruik = verb;
    }

    // methoden
    ...
}
```

HFST 4 KLASSEN, OBJECTEN EN METHODEN + VERVOLG METHODEN (DEEL HFST 6), DEEL 2

ZELF METHODEN SCHRIJVEN MET PARAMETERS EN RETURNTYPES

Voorbeelden methodedefinities

```
public int genereerGetal()  
public int increment(int getal1)  
public int berekenSom(int getal1, int getal2)  
public char geefEersteLetter (String s)  
public void setLeeftijd (int leeftijd)  
public boolean isOud(short leeftijd)
```

Een methode heeft steeds 1 functionaliteit te volbrengen :

Daarom :

- denk goed na over input (parameters) en return (wat moet de methode berekenen en teruggeven als resultaat.
- Splits in nuttige deelfunctionaliteiten als de methode te lang wordt !
- Wanneer de methode geen resultaat teruggeeft (bv omdat een dataveld van waarde veranderd moet worden, dan geef je **void** terug

```
public char geefEersteLetter (String s)
```

```
public char geefEersteLetter(String s) {  
    return s.charAt(0);  
}
```

```
public void setLeeftijd (int leeftijd)
```

```
public void setLeeftijd(int leeftijd){  
    this.leeftijd = leeftijd; // dataveld leeftijd uit de klasse Persoon  
}
```

Hiermee duid je het dataveld van het huidige object aan (zie verder semester2 Java OO)

ARRAYS ALS PARAMETERS EN RETURNTYPES

```
public boolean zoek(int[] rij, char c);
```

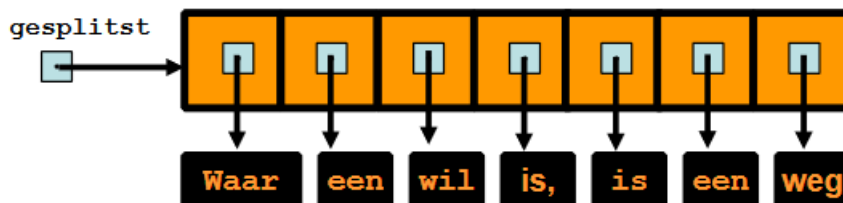
```
public boolean zoek(char[] rij, char c) {  
    boolean gevonden = false;  
  
    for (int i = 0; i < rij.length; i++) {  
        if (rij[i] == c) {  
            gevonden = true;  
            break;  
        }  
    }  
  
    return gevonden;  
}
```

```
public boolean zoek(char[] rij, char c) {  
    for (int i = 0; i < rij.length; i++) {  
        if (rij[i] == c) {  
            return true;  
        }  
    }  
  
    return false;  
}
```

```
// methode split van de klasse String  
public String[] split(String s);
```

```
String mijnTekst = "Waar een wil is, is een weg";
```

```
String[] gesplitst = mijnTekst.split(" ");
```



```
String[] gesplitst = mijnTekst.split(",");
```



⇒ **String opdelen**: `String[] gesplitst = tekst.split(",");`

ALGEMEEN : METHODEN MET PARAMETERS EN RETURN TYPE

```
public <returntype> methodeVanKlasseX(<parameterlijst,incl type>)
```

```
<returntype> resultaat;  
resultaat = <objectVanKlasseX>.methodeVanKlasseX(parameterwaarden);
```

klasse String

```
String obj = new String("hallo");
```

Oproepen:

Methodedefinities:

```
int resultaat;
```

```
public int indexOf(char c)
```

```
resultaat = obj.indexOf('x');
```

```
char resultaat;
```

```
public char charAt(int i)
```

```
resultaat = obj.charAt(2);
```

```
String resultaat;
```

```
public String substring(int i,  
                        int j)
```

```
resultaat = obj.substring(1,4);
```

STATISCHE METHODEN

deze worden vooraf gegaan door het keyword **static**

wanneer je een methode wil maken die **onafhankelijk** is **van de data members** in de klasse en die dus ook onafhankelijk is van gelijk welk object van die klasse dan kan je best de methode static maken. Je hoeft dan ook geen object aan te maken bij methode aanroep, **je roept ze aan via de klasse!**

Vb : de methoden in de klasse Math zijn static !

```
double getal = Math.random();
```

STATISCHE METHODEN IN DE MAIN KLASSE

```
public class RijSorteer {  
  
    public static void main(String[] args) {  
  
        int[] rij = {3, 1, 6, 2, 5, 10, 9, 8, 4, 7};  
  
        //afdrukken  
        drukRij(rij);  
        //sorteren  
        sorteerRij(rij);  
        //afdrukken  
        drukRij(rij);  
  
    }  
    ...  
}
```

```

public static void drukRij(int[] rij) {
    System.out.print("inhoud array : (");

    int i;
    for (i = 0; i < rij.length - 1; i++) {
        System.out.print(rij[i] + ", ");
    }

    System.out.print(rij[i] + " )\n");
}

...

```

```

public static void sorteerRij(int[] rij){
    int index;
    for (int i = 0; i < rij.length; i++) {
        int indexKleinste = i;

        //ga op zoek te gaan naar de index van het kleinste
        //element in de rij, startend vanaf een gegeven positie
        for (int j = i + 1; j < rij.length; j++) {
            if (rij[j] < rij[indexKleinste]) {
                indexKleinste = j;
            }
        }

        index = indexKleinste;

        if (i != index) {
            //verwissel i met index
            int temp = rij[i];
            rij[i] = rij[index];
            rij[index] = temp;
        }
    }
}

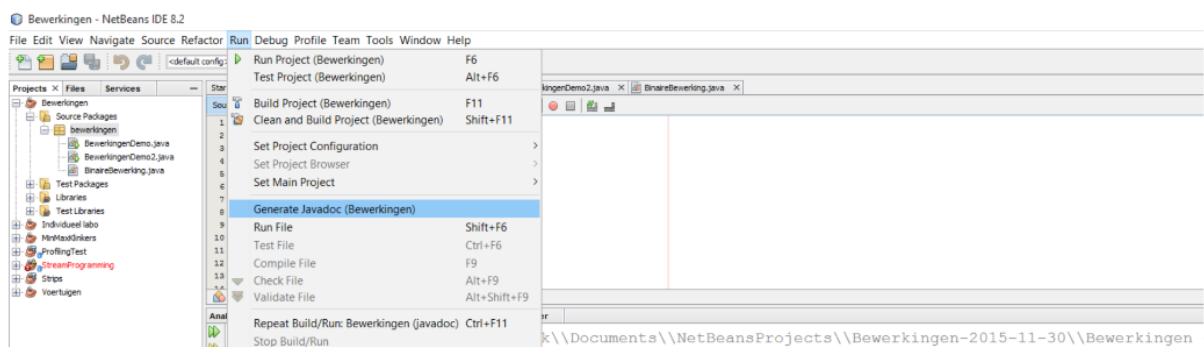
} // klasse RijSorteer

```

JAVADOC --TAGS :

@author - @version - @param - @return - @see

GENERATE JAVADOC



METHOD OVERLOADING

In een programma kan je meerdere methode definities hebben met **dezelfde naam**, maar met een **andere parameterlijst**: → **Method Overloading**



→ Minder of meer parameters



LABO 10 (LABO 9 HOORT BIJ HFDST. 5)

Theorie is voldoende.

Hfst 5 ARRAYS (1 DIM + 2 DIM), DEEL 1

ARRAYS

- Een array is een **datastructuur**, deze kan een collectie van variabelen bijhouden en er via 1 variabelenaam naar refereren.
- Let wel een array kan alleen een collectie van homogene data bijhouden, m.a.w. elk element in de array moet van hetzelfde type zijn.
- Een array in Java is geïmplementeerd als een object
- Een array kan meerdere dimensies hebben

ARRAYS : DECLARATIE, INSTANTIATIE EN INITIALISATIE

```
//1) declaratie van een array
int[] balRij;

//2) instantiatie van een array
balRij = new int[6];

//3) initialisatie van de array
for (int i = 0 ; i < balRij.length ; i++) {
    balRij[i] = (int) (Math.random() * 45) + 1;
}
```

EEN ARRAY ALS OBJECT



Lengte van de rij opvragen:

```
int lengte = balRij.length;
```

Let op **NIET: length()**
dit is geen methode
maar data

(Laatste) element van de rij opvragen:

```
int laatstGetrokkenBal = balRij[5];
```

```
int laatstGetrokkenBal = balRij[lengte - 1];
```

```
int laatstGetrokkenBal = balRij[balRij.length - 1];
```

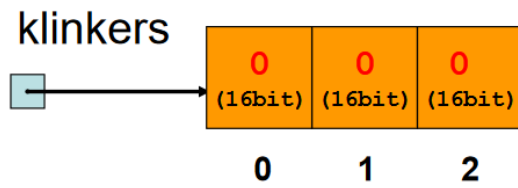
→ Druk de elementen van de rij af:

```
for (int idx = 0; idx < balRij.length; idx++) {
    System.out.print(balRij[idx] + " ");
}
```

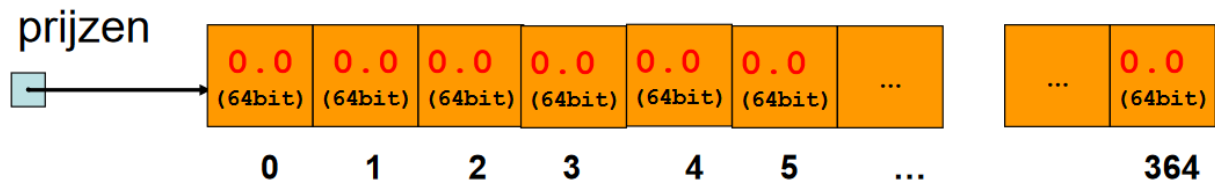
INSTANTIATIE

```
//STAP1: declaratie  
char[] klinkers;
```

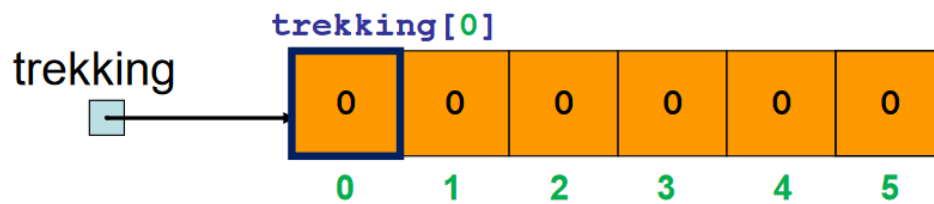
```
//STAP2: instantiatie  
klinkers = new char[3];
```



```
//STAP1+2 samen: declaratie én instantiatie  
double[] prijzen = new double[365];
```

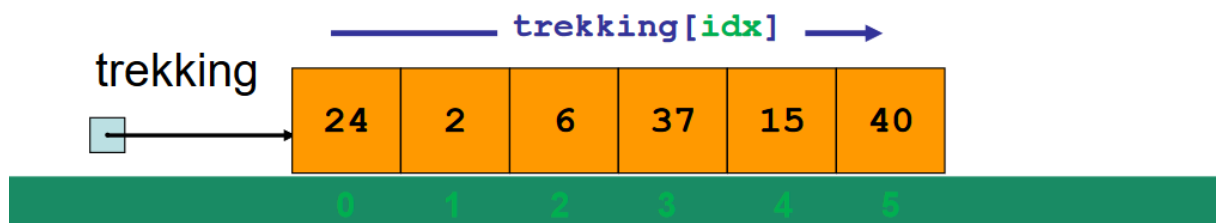


INITIALISATIE



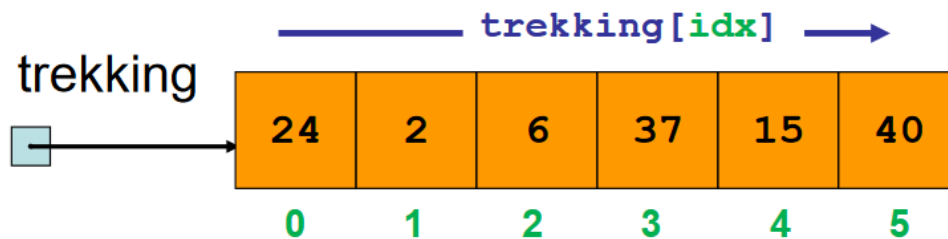
//STAP3: het opvullen van de array met concrete waarden

```
trekking[0] = (int) (Math.random() * 45) + 1;  
trekking[1] = (int) (Math.random() * 45) + 1;  
...  
trekking[5] = (int) (Math.random() * 45) + 1;
```



INITIALISATIE (FOR LUS)

```
for (int idx = 0; idx < trekking.length; idx++) {  
    trekking[idx] = (int) (Math.random() * 45) + 1;  
}
```



INITIALISATIE (INITIALIZER SYNTAX)

= via initializer syntax kan declaratie, instantiatie en initialisatie gebundeld worden in 1 java instructie

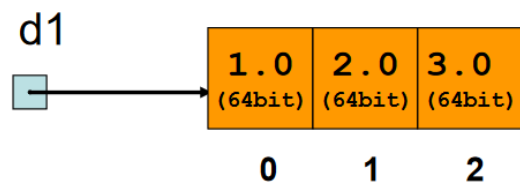
```
char[] klinkers = {'a', 'e', 'i', 'o', 'u'};
```

```
int[] priemgetallen = {2, 3, 5, 7, 11, 13, 17, 19};
```

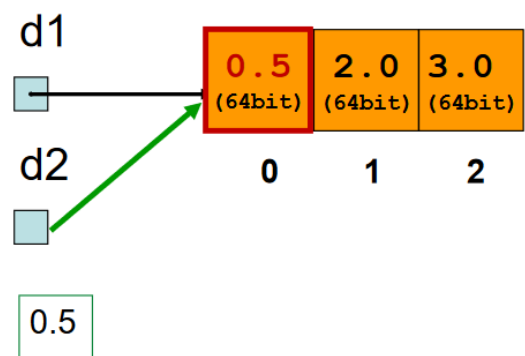
```
String[] dagen = { "maandag", "dinsdag", "woensdag",  
                  "donderdag", "vrijdag",  
                  "zaterdag", "zondag"};
```

EEN ARRAY IS EEN OBJECT

```
double[] d1 = {1.0, 2.0, 3.0};  
double[] d2;
```

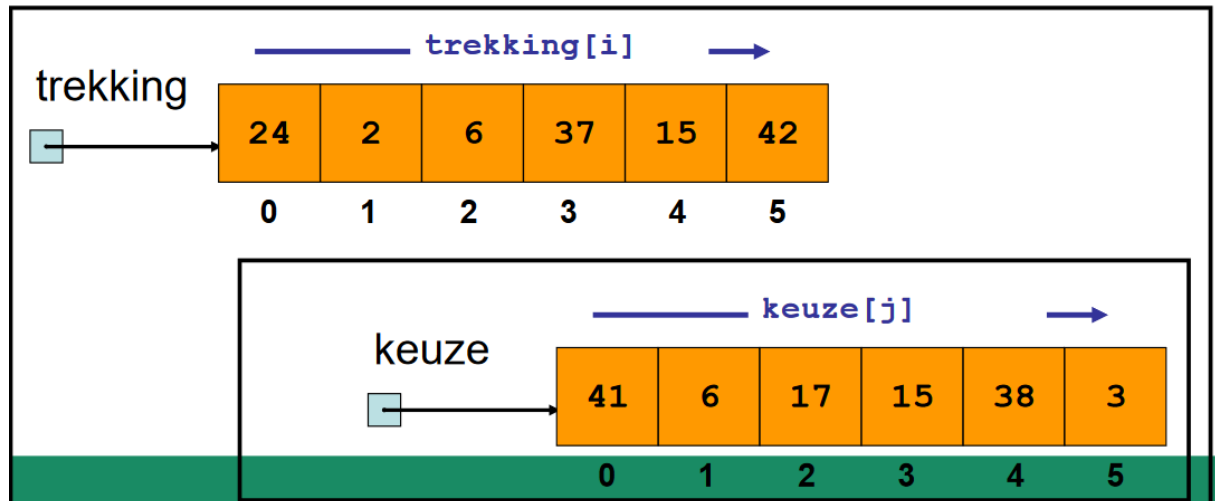


```
d2 = d1;  
d2[0] /= 2;  
System.out.println(d1[0]);
```



LOOPEN OVER ARRAYS

```
for (int i = 0; i < 6; i++) {  
    for (int j = 0; j < 6; j++) {  
        if (trekking[i] == keuze[j]) {  
            aantalJuist++;  
        }  
    }  
}
```

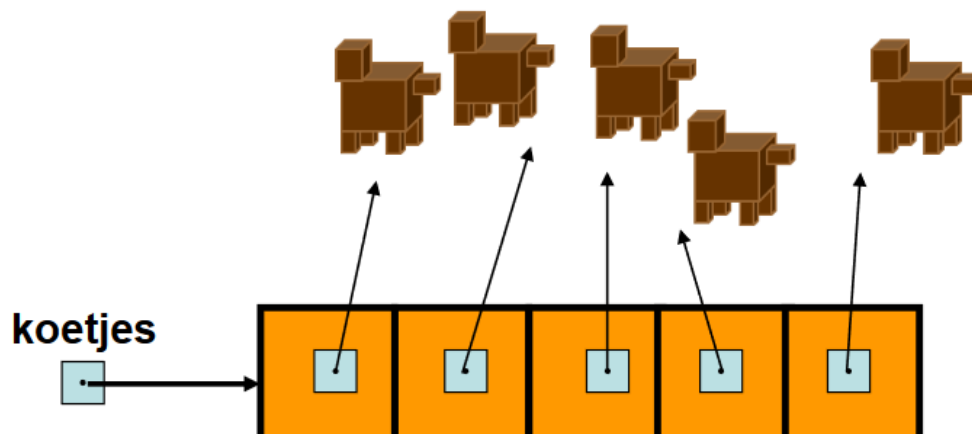


ARRAYS OPVULLEN MET OBJECTEN

```
Koe[] koetjes; //STAP1
```

```
koetjes = new Koe[5]; //STAP2
```

```
for (int i = 0; i < koetjes.length; i++) {  
    koetjes[i] = new Koe(); //STAP3  
}
```



LABO 9

Oefening 1:

RijOmgekeerd: Lees 10 getallen in een array en druk ze op het scherm in omgekeerde volgorde.

```
15 public class RijOmgekeerd {
16     public static void main(String[] args) {
17         int[] getallenArray = new int[10];
18         int[] getallenArrayOmgekeerd = new int[getallenArray.length];
19         Scanner scan = new Scanner(System.in);
20         System.out.println("Geef 10 getallen in: ");
21
22         for (int i = 0; i < getallenArray.length; i++){
23             System.out.println("Geef het " + (i+1) + " de getal in: ");
24             getallenArray[i] = scan.nextInt();
25         }
26         int i = 0;
27         for (int j = getallenArray.length - 1; j >= 0; j--){
28             getallenArrayOmgekeerd[i] = getallenArray[j];
29             i++;
30         }
31         System.out.println("De getallen array is: " + Arrays.toString(getallenArray));
32         System.out.println("De omgekeerde getallen array is: " + Arrays.toString(getallenArrayOmgekeerd));
33     }
34 }
35
36
```

Om de array uit te printen: `Arrays.toString(arrayNaam)`

Om een array om te draaien (naar een andere array):

```
int i = 0;
```

```
for (int j = getallenArray.length - 1; j >= 0; j--){
    getallenArrayOmgekeerd[i] = getallenArray[j];
    i++;
}
```

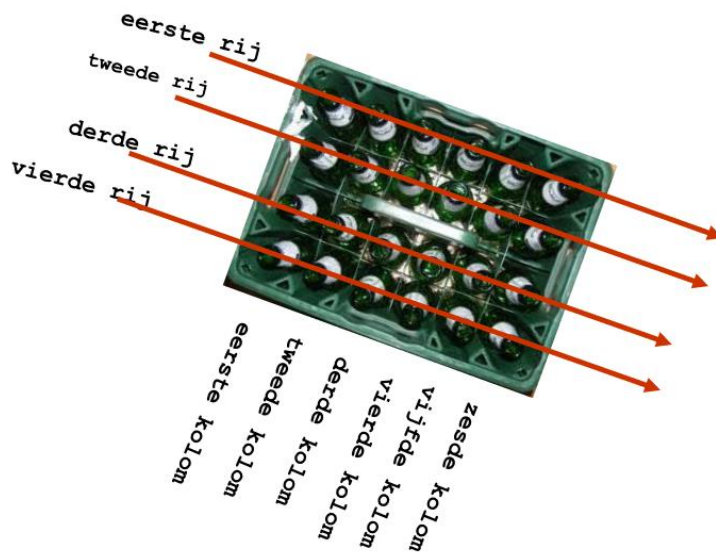
Een for-lus kijkt naar de waarde aan het BEGIN van zijn loop:

Vb. `for (int i = 0; i < array.length; i++){`

voor hij deze code gaat uitvoeren gaat hij kijken of `i` al dan niet kleiner is dan `array.length` en als die voorwaarde voldaan is voert hij de code uit.

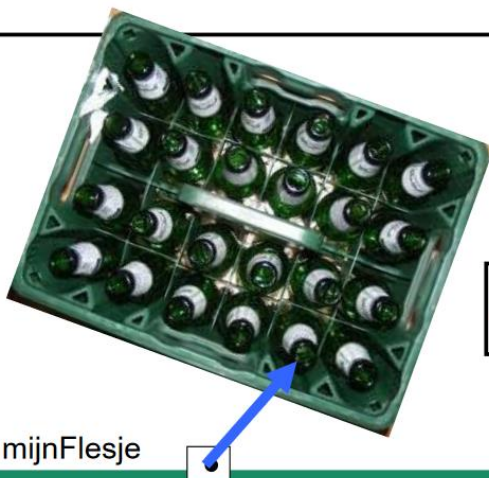
HFST 5 ARRAYS (1 DIM + 2 DIM), DEEL 1

ORDENEN IN 2 DIMENSIES : RIJEN EN KOLOMMEN



```
boolean[][] krat;           //declaratie van een 2 dim array  
krat = new boolean[4][6]; //aanmaken van het 2 dim array object
```

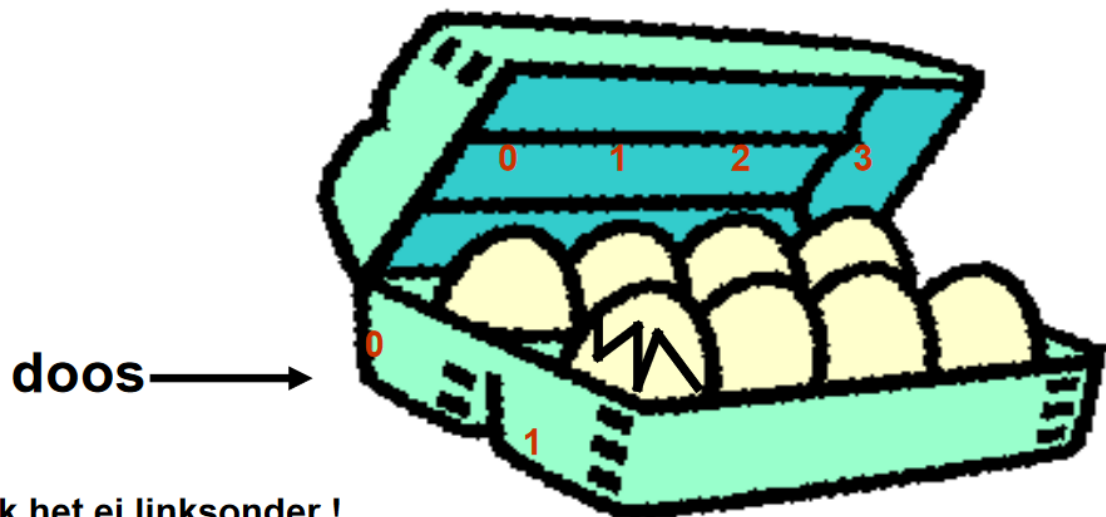
```
boolean[][] krat; //declaratie van een 2-dimensionale array  
krat = new boolean[4][6]; //aanmaken van het array object  
  
//opvullen met flesjes  
for (int rij = 0; rij < 4; rij++) { //vul de rijen  
    for (int kol = 0; kol < 6; kol++) { //vul elk elem  
        krat[rij][kol] = true;  
    }  
}
```



```
//neem een blikje uit het krat  
boolean mijnFlesje = krat[3][4];
```

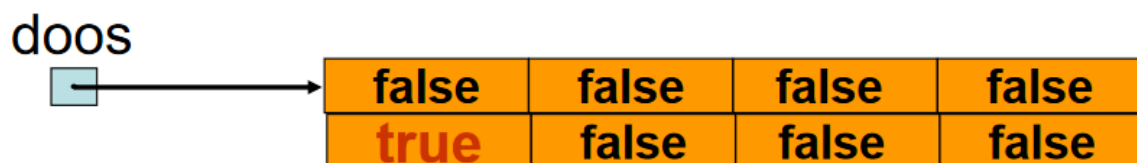
```
Geef het aantal rijen in:
2
Geef het aantal kolommen in:
4
0 0 0 0

1 1 1 1
```



```
//Breek het ei
linksonder
doos[1][0] = true;
```

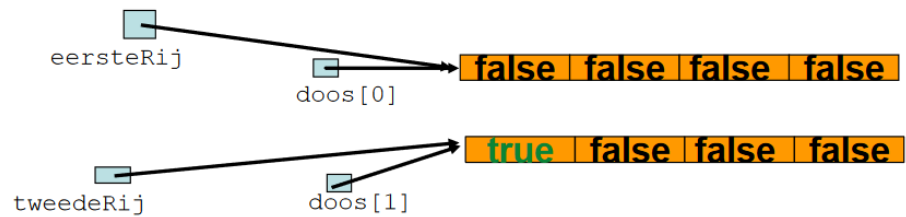
Voorstelling in het geheugen



```
boolean isGebroken;
```

```
isGebroken = doos[0][0]; //geeft false
isGebroken = doos[1][0]; //geeft true
```

EEN 2 DIM ARRAY IS EIGENLIJK 2 X EEN 1 DIM ARRAY

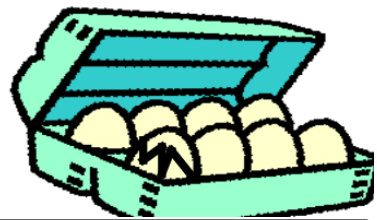


Als je van een tweedimensionele array slechts 1 dimensie opvraagt
dan verkrijg je een eendimensionele array

```
boolean[] eersteRij = doos[0];  
boolean[] tweedeRij = doos[1];
```

```
boolean isGebroken;
```

```
isGebroken = eersteRij[0]; //geeft false  
isGebroken = tweedeRij[0]; //geeft true
```



```
int aantalGebroken = 0;  
  
for (int i = 0; i < doos.length; i++) {  
    for (int j = 0; j < doos[i].length; j++) {  
        if (rij[i][j] == true) {  
            aantalGebroken++;  
        }  
    }  
}  
  
System.out.println("Er zijn " + aantalGebroken +  
    " eitjes gebroken");
```

DE DERDE DIMENSIE



Grondplan van een appartementsgebouw

app1	app2	app3	app4	app5	app6	app7	app8	app9	app1
app11	app12	app13	app14	app15	app16	app17	app18	app19	app2

```
public void voegNieuweHuurderToe() {
    Scanner scan = new Scanner(System.in);
    System.out.println("\nOp welke verdieping komt de nieuwe huurder [0, "
        + (HOOGTE - 1) + "] ?");
    int verdieping = scan.nextInt();

    System.out.println("Kies een appartement [1, " + (DIEPTE * BREEDTE) + "]");
    int keuzeAppt = scan.nextInt() - 1;

    int rij = keuzeAppt / BREEDTE;
    int kol = keuzeAppt % BREEDTE;

    if (appt[verdieping][rij][kol] == 0) {
        System.out.println("Met hoeveel komt men hier wonen ?");
        appt[verdieping][rij][kol] = scan.nextInt();
    }
    else {
        System.out.println("Hier woont al iemand...");
    }
}
```

```
Op welke verdieping komt de nieuwe huurder [0, 7] ?
1
Kies een appartement [1, 20]
15
Met hoeveel komt men hier wonen ?
12
```

LABO 11

```
public class WoordSpel {
```

```
    final int MAXBEURTEN;
```

```
    String[] woorden = {"donan", "appel", "peer", "tentoonstelling", "vijf", "zes", "zeven", "acht", "negen",  
"tien"};
```

```
    String huidigWoord;
```

```
    boolean[] isGeraden;
```

```
    int beurten;
```

```
    public WoordSpel() {
```

```
        this.MAXBEURTEN = 10;
```

```
        this.huidigWoord = woorden[(int) (Math.random() * 10)];
```

```
        isGeraden = new boolean[huidigWoord.length()];
```

```
        int beurten = 0;
```

```
    }
```

Bij de constructor van boolean[] isGeraden de this.isGeraden de this weglaten dus

```
isGeraden = new boolean[]....
```