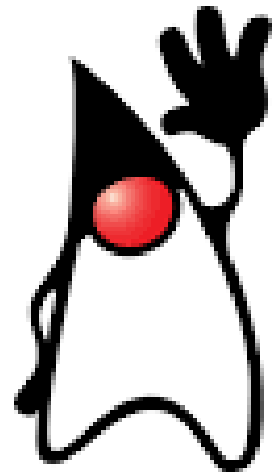


# Java OO programming



- Klasgroep: 1ELO en 1ICT
- Opleiding: Prof. Bachelor Elektronica-ICT
- Lokaal: G.A. (E036) - zie ook valven/studentenportaal
- Tijdstip: Woensdag lestijd 2 (9u45-11u10)
- Docent: Kristien Van Assche
- Contact: [kristien.vanassche@odisee.be](mailto:kristien.vanassche@odisee.be)
- Handboek (opt): Java A Beginner's Guide, 6<sup>th</sup> edition  
by Herbert Schildt  
Publisher: McGraw-Hill Osborne Media  
ISBN: 9780071809252

# RECAP SEM1 - concept

Demo klasse

-> main

Logic klasse

- > datavelden
- > constructoren
- > methoden

# Cf. examen semester1

**Artikel** Om het stockbeheer van je winkel te optimaliseren schrijf je een applicatie die toelaat om per artikel een overzicht te genereren van de aantallen verschillende maten (kledingmaten, schoenmaten, lengtes, ...) die je ervan verkocht. Hiertoe schrijf je een aparte klasse **Artikel**. Om deze klasse te demonstreren schrijf je meteen ook een klasse **Demo** waarin je een *main* methode opneemt met daarin een object van type **Artikel**.

**Enquete** Schrijf een applicatie die toelaat om een enquete te doen bij een aantal deelnemers. Hiertoe schrijf je een aparte klasse **Enquete**. Om deze klasse te demonstreren schrijf je meteen ook een klasse **DemoEnquete** waarin je een *main* methode opneemt met daarin een object van type **Enquete**.

# RECAP SEM1 – opbouw programma

```
public class DemoEnquete {  
  
    public static void main(String[] args) {  
        Enquete e = new Enquete(...);  
  
        for (int i = 0; i < 10; i++) {  
            e.vulEnqueteIn(...);  
        }  
  
        System.out.println(  
            e.geefMeestGekozenAntwoord() );  
    }  
}
```

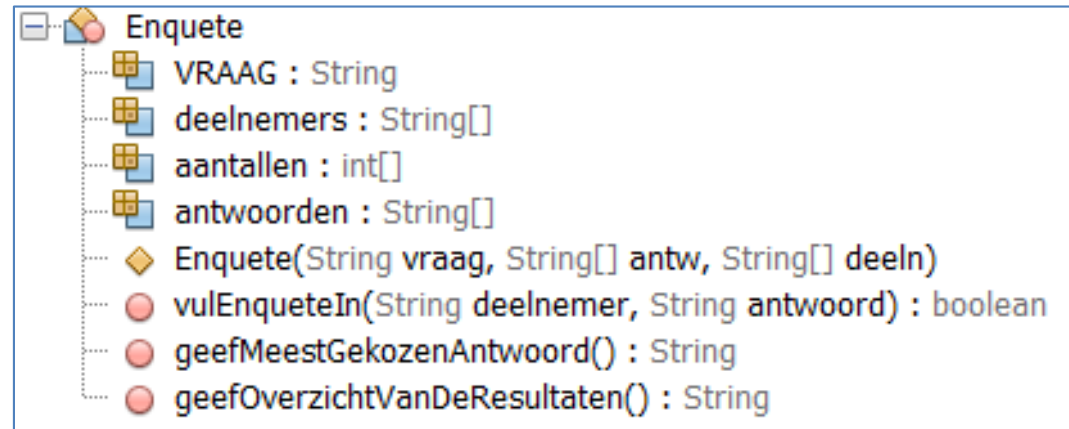
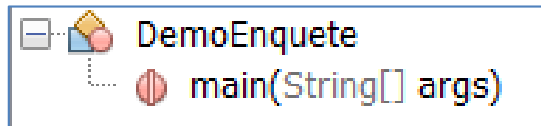
```
public class Enquete {  
    String vraag;  
    String[] antwoorden;  
    int[] aantallen;  
    String[] deelnemers;
```

```
    public Enquete(...) {  
        ...  
    }
```

```
    public boolean vulEnquete in(...) {  
        ...  
    }
```

```
    public String geefMeestGekozenAntwoord() {  
        ...  
    }  
}
```

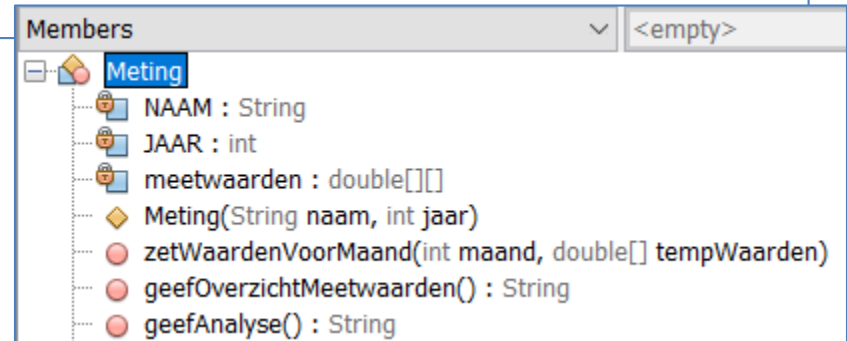
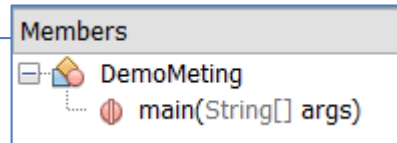
# Netbeans Navigator view



# eerste labo sem2: herhaling van de '2-klassen-approach'

Schrijf een toepassing die toelaat om maandelijks een reeks meetwaarden in te geven, en dat voor alle maanden van het jaar. Nadat alle gemeten data voor een jaar ingegeven is, kan je er statistische informatie over opvragen, zoals: Wat was de minimum, resp. maximum gemeten waarde en wanneer werd die opgemeten? In welke maand werd de laagste, resp. hoogste gemiddelde waarde bekomen? Je kan deze klasse gebruiken om bijvoorbeeld temperatuursmetingen bij te houden: Welke dag was de koudste, resp. warmste dag van het jaar? Wat was de gemeten temperatuur op die bewuste dag? Welke maand had de hoogste gemiddelde temperatuur?

Maak een nieuw Netbeans project voor deze opgave. Je toepassing zal uit 2 klassen bestaan: een logische klasse `Meting` en een presentatieklasse `DemoMeting`. De functionaliteit van je logische klasse `Meting` kan je verifiëren aan de hand van de JUnit testen die voor deze opgave meegeleverd zijn.



## Noot:

*Info over 'Jagged arrays',  
zie bijlage op einde van deze presentatie*

# Overzicht les1

Klassen & objecten

Console toepassingen & grafische toepassingen

Publieke & private methoden

Javadoc

UML klassendiagramma's

Velden & lokale variabelen (+ scope)

Statische methoden & statische velden

Sleutelwoorden static & final

Opbouw Java programma: Presentatie & logica

Private velden & publieke getters/setters

# Java OO programming

- 6 stp (2 stp voor theorie + 4 stp voor labo)
- 1,5u theorie/week
- 1,5u labo/week
- Individueel labo halfweg semester
- Vakoverschrijdend project, samen met Relational Databases
- *Mogelijke lancering van scantestjes – niet op punten*
- Individueel labo in EP2
- Online Toledo theorie examen in EP2



# Java OO Programming (B-ODISEE-JPW274)

6 studiepunten  Nederlands  36 uren  Tweede semester

 Van Assche Kristien (coördinator) | [Meer](#)

Kernteam Elektronica-ICT Gent

## Toelichting

## Toelichting 2e examenkans

### OLA Java OO Programming (Theorie) [2 STP]

- **100%:** online Toledo examen (gesloten boek, geen externe bronnen)

### OLA Java OO Programming (Lab) [4 STP]

- **10%:** permanente evaluatie op basis van een labobundel (specifieke richtlijnen worden via Toledo meegedeeld)
- **20%:** Individueel labo, halfweg het semester, gemaakt op computer (op computer, geen externe bronnen)
- **40%:** Individueel labo examen, op het einde van het semester (op computer, geen externe bronnen)
- **30%:** Individueel te maken project, in combinatie met OLA Relational Databases (Het gedeelte m.b.t. gegevensbanken wordt in de OPO Relational Databases gequoteerd)

OPGELET: zie examenreglement

- Het OPO-cijfer wordt automatisch berekend: Daarbij wordt een OLA-score van 7/20 of lager beschouwd als een 'extreem onvoldoende', waardoor het eindresultaat voor het hele opleidingsonderdeel zal herleid worden naar de laagste OLA-score.

- Een OLA-cijfer lager dan 10 wordt niet meegenomen van eerste naar tweede zittijd.

OPO TOLEREERBAAR

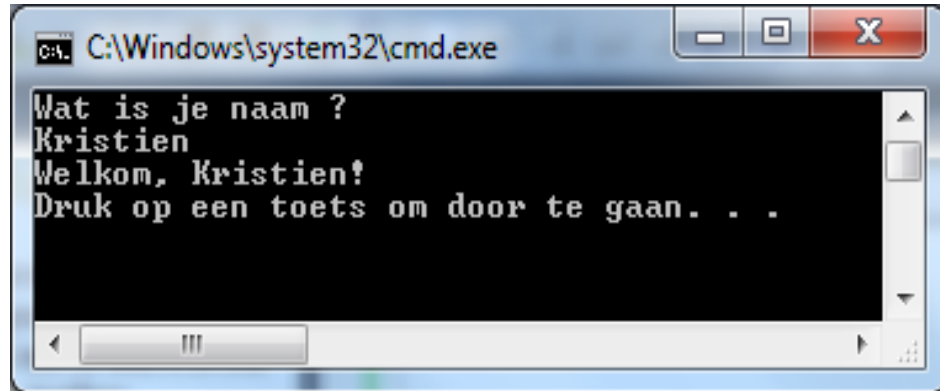
← Afstudeerrichting Elektronica →

← Afstudeerrichting ICT →

1ICT		2ICT		3ICT
<b>sem1</b>	<b>sem2</b>	<i>sem1</i>	<i>sem2</i>	<i>Sem1</i>
<b>Java</b>	<b>Java</b>	<i>C#</i>	<i>C#</i>	C, C#, Java, ...
<b>Console toep</b>	<b>Console en Grafische toepassingen</b>			
<b>teksteditor + Netbeans</b>	<b>Netbeans</b>	Visual Studio	Visual Studio	Netbeans, Eclipse, Visual Studio, ...

# Console toepassingen in Java

Vanuit DOS-prompt:



```
C:\Windows\system32\cmd.exe

Wat is je naam ?
Kristien
Welkom, Kristien!
Druk op een toets om door te gaan. . .
```

Compileren:

Uitvoeren:

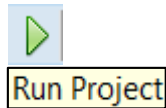
➤ **javac.exe** Groet.java

➤ **java.exe** Groet

➤ **javac** Groet.java

➤ **java** Groet

Vanuit Netbeans IDE:

A screenshot of the NetBeans IDE's 'Output' window. The window has tabs for 'ges', 'Output X', and 'Test Results'. The 'Output' tab is active, showing a 'Debugger Console X' and a 'Labo JavaOO Hoofdstuk 1 (run) X' tab. The console output shows the program's execution: 'run:', 'Wat je naam ?', 'Tommy', 'Wat is je leeftijd?', '18', 'Heb je nog interesse voor ICT? (j/n)', 'j', 'Veel plezier !', and 'BUILD SUCCESSFUL (total time: 12 seconds)'.

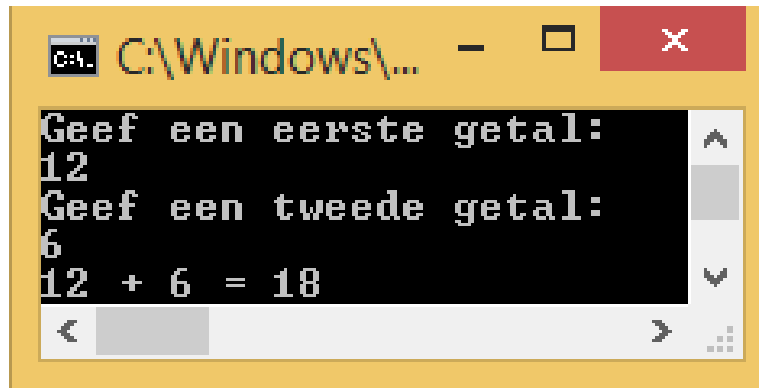
Console toepassing & interactie met de gebruiker:

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

```
System.out.println("Wat is je naam?");  
String naam = sc.nextLine();
```

```
System.out.println("Wat is je leeftijd?");  
int leeftijd = sc.nextInt();
```

```
System.out.println("Heb je nog interesse voor ICT? (j/n)");  
boolean interesse = (sc.next().charAt(0) == 'j');
```



- Interactieve input via de Scanner klasse:

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

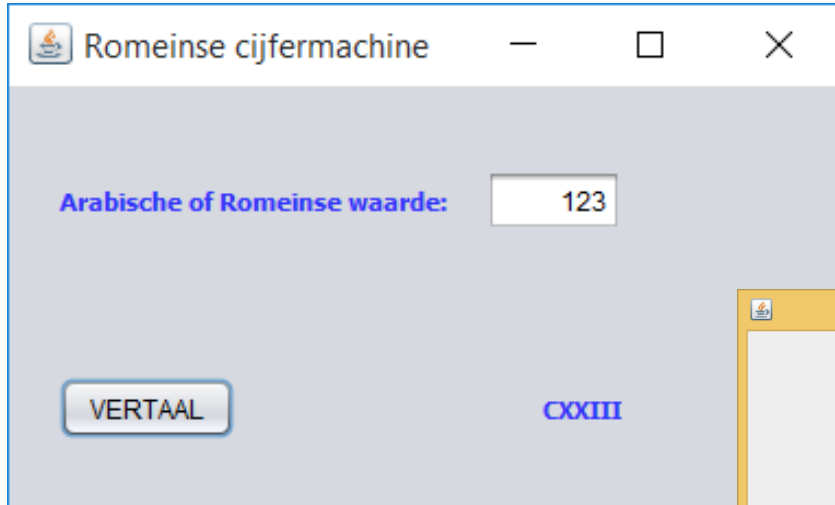
#### Alternatieven:

- Hard gecodeerde data
- Input uit String-object, typisch via Scanner en delimiter

```
String datum = "28/01/2019";  
Scanner sc = new Scanner(datum).useDelimiter("/");
```

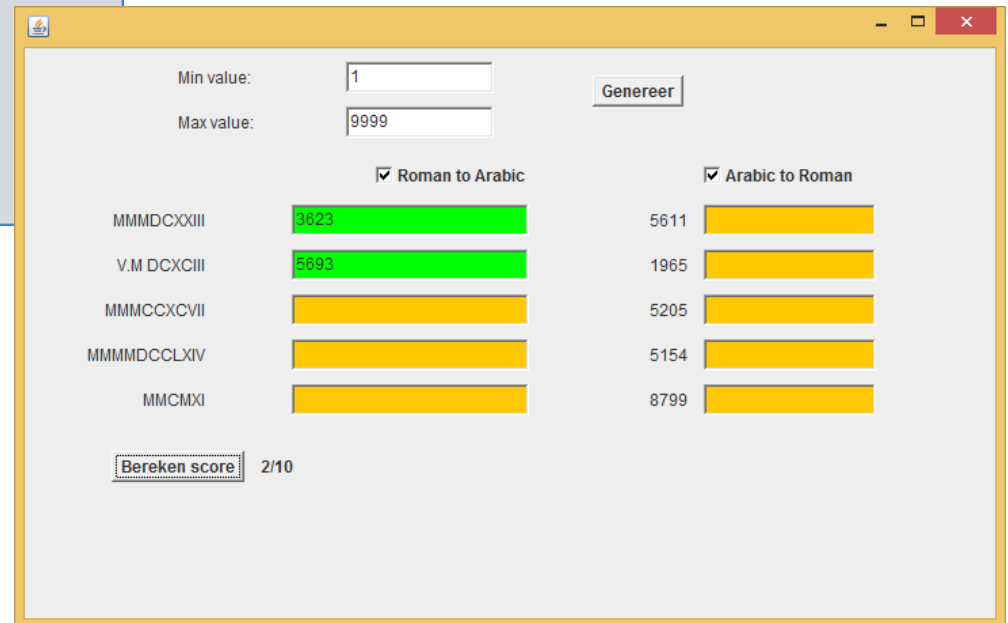
- Input via parameters van de main() methode
- Input uit bestand
- Input uit databank
- ...

# Grafische toepassingen in Java

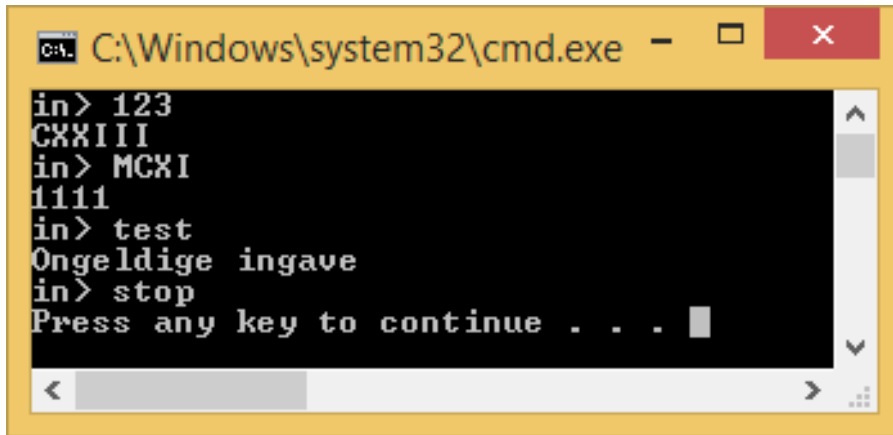


GUI-componenten:

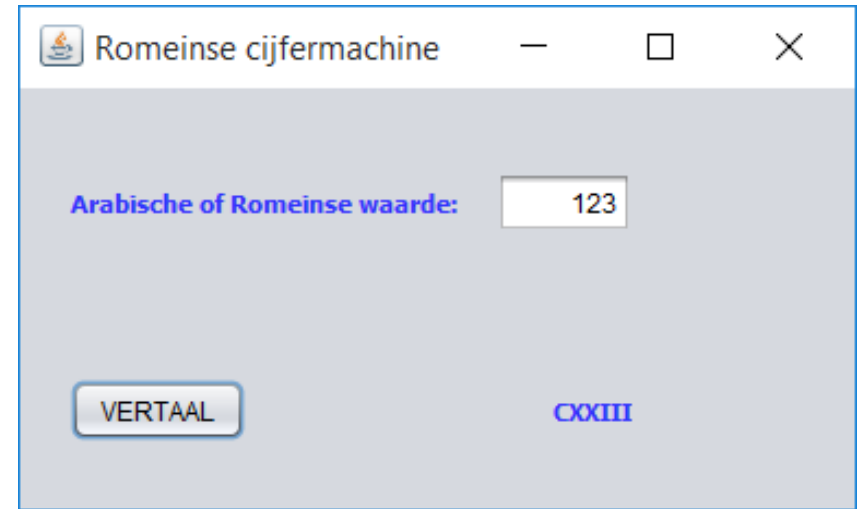
- Tekstvelden
- Labels
- Knoppen
- Checkboxes
- ...



# Console toepassing en Grafische toepassing voor eenzelfde logica



```
in> 123
CXXIII
in> MCKI
1111
in> test
Ongeldige ingave
in> stop
Press any key to continue . . .
```

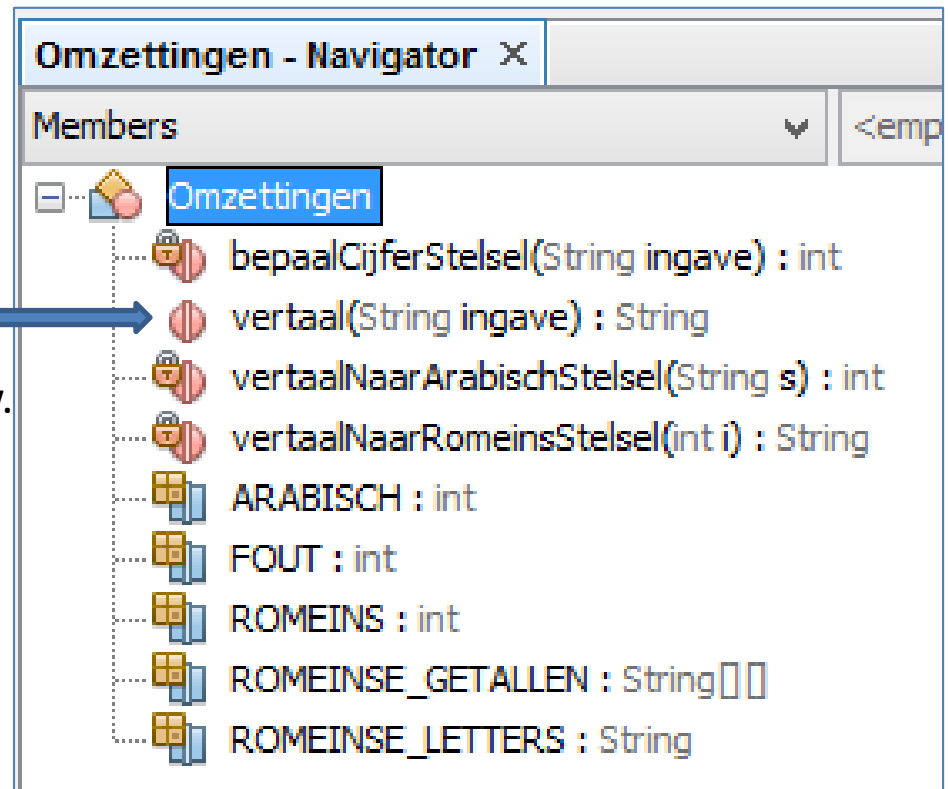


- ➔ 'Omzettingen' is een klasse die de **logica** van het vertalen tussen getalstelsels bevat
- ➔ Kan zowel door console toepassing als door grafische toepassing worden gebruikt!!!

# Logische klasse publieke en private methoden

Methoden die **extern** toegankelijk moeten zijn definieer je 'public': 🔒

Methoden enkel voor **intern** gebruik, m.a.w. enkel opgeroepen binnen de klasse zelf, definieer je 'private': 🔒



=> Zie later: eerste principe van SOLID = Single Responsibility



# Javadoc

## verwijzing naar andere methoden via @see

```
/**
 * Van hieruit wordt de gepaste conversie methode opgeroepen.
 *
 * @param ingave de tekst die moet vertaald worden naar het ander cijferstelsel
 * @return de waarde volgens het andere cijferstelsel
 *
 * @see #bepaalCijferStelsel(String)
 * @see Omzettingen#vertaalNaarRomeinsStelsel(int)
 * @see Omzettingen#vertaalNaarArabischStelsel(String)
 */
public static String vertaal(String ingave) {
    if (ingave == null || ingave.length() == 0) {
        return "Geen ingave";
    }

    int stelsel = bepaalCijferStelsel(ingave);

    switch (stelsel) {
        case ARABISCH:
            return vertaalNaarRomeinsStelsel(Integer.parseInt(ingave));
        case ROMEINS:
            int waarde = vertaalNaarArabischStelsel(ingave);
            return "" + (waarde > 0 ? waarde : "Ongeldige ingave");
        default:
            return "Ongeldige ingave";
    }
}
```

RomeinseCijferMachine.java

Omzettingen.java

GrafischVenster.java

MMIX  $\leftrightarrow$  2009

ExtendedGUI.java

DEMO !

The screenshot shows a Java Swing window titled "RomeinseCijferMachine". It features a light gray background and a yellow title bar. At the top, there are two input fields: "Min value:" with the value "1" and "Max value:" with the value "1000". To the right of these fields is a button labeled "Genereer". Below the input fields, there are two checkboxes: "Roman to Arabic" (checked) and "Arabic to Roman" (unchecked). The main area of the window displays a list of Roman numerals on the left and their corresponding Arabic values on the right. The values are displayed in green bars. The list includes: DLXXVIII (578), DXXV (525), DCCXXX (empty bar), LVI (56), and CXLVIII (148). At the bottom left, there is a button labeled "Bereken score" and a score of "4/5".

Roman Numeral	Arabic Value
DLXXVIII	578
DXXV	525
DCCXXX	
LVI	56
CXLVIII	148

# Klasse versus object

Een **klasse** is een blauwdruk, i.e. een abstracte beschrijving van een **niet-primitief type**.

```
class Hond {  
}
```

```
class String {  
}
```

Een klasse drukt uit **hoe** een element van dat type (wat men een **object** noemt) **gebouwd** moet worden.

i.e. de **constructoren** van de klasse

```
public Hond(String naam)  
➔ Hond object = new Hond("blacky");
```

Het legt tevens vast **welke data** het object bevat en **welke operaties** (op die data) toegelaten zijn.

i.e. de **datavelden** van de klasse

```
String naam;
```

i.e. de **methoden** van de klasse

```
public String spreek();
```

# object(en) van een klasse

Gebruik vanuit andere context:

```
Hond blacky = new Hond("Blacky", true);  
System.out.println(blacky.geefInfo());
```

Cf. sem1:  
vanuit main methode in klasse DemoHond.java

Klassedefinitie:

```
class Hond {  
    String naam;  
    boolean rashond;  
  
    public Hond(String par1, boolean par2) {  
        naam = par1;  
        rashond = par2;  
    }  
  
    public String geefInfo() {  
        return naam +  
            rashond ? " (Rashond)" : "";  
    }  
}
```

# UML klassendiagramma's

## Unified Modelling Language



Zowel de 'data' als de 'operaties' worden "members van de klasse" genoemd

*Alternatieve terminologie:*

- *datavelden*
- *fields*
- *member variabelen*
- *instantie variabelen*

Soorten methoden:

- Standaard constructor
- Niet-standaard constructoren
- Statische en niet-statische methoden

```
class Hond {
```

```
//Automatisch gegenereerde  
//standaard constructor
```

```
}
```

```
class Hond {
```

```
String naam;  
boolean rashond;
```

```
public Hond(String par1, boolean par2) {...}  
public String geefInfo() {...}
```

```
}
```

Opm: Geen standaard constructor meer beschikbaar

## Hond

```
//standaard constructor  
public Hond()
```

## Hond

```
String naam  
boolean rashond
```

```
//niet-standaard constructor  
public Hond(String naam, boolean rashond)
```

```
//andere methoden  
public String geefInfo()
```

# Datavelden versus lokale variabelen

```
public class Demo {  
    public static void main(String... args) {  
        Test obj = new Test();  
        obj.voerUit();  
    }  
}
```

```
public class Test {  
    int waarde = 4;  
  
    public void voerUit() {  
        int waarde = 10;  
        System.out.println(this.waarde);  
    }  
}
```

Wat komt er op het uitvoerscherm?

# Constructor overloading

## Method overloading

```
Rekenblad b1 = new Rekenblad(4, 8);  
Rekenblad b2 = new Rekenblad(4, 8, '+');
```

```
System.out.println(b1.rekenUit('+');  
System.out.println(b2.rekenUit());
```

### Rekenblad

```
int term1  
int term2  
char operator
```

```
public Rekenblad(int term1, int term2)  
public Rekenblad(int term1, int term2,  
                  char operator)
```

```
public int rekenUit(char operator)  
public int rekenUit()
```



# Javadoc

```
/**
 * Voert rekenkundige bewerking volgens operator uit op term1 en term2
 *
 * @see #rekenUit(char)
 * @return Het resultaat van de berekening
 */
public int rekenUit() {
    return rekenUit(operator); //oproep van overloaded methode
}

/**
 * Voert een gegeven rekenkundige bewerking uit op term1 en term2
 *
 * @param operator De operator (+, -, *, / of %)
 * @return Het resultaat van de berekening
 */
public int rekenUit(char operator) {
    ...
}
```

```
public class RekenenViaParametersMain {  
    public static void main(String[] args) {  
        if (args.length != 3) {  
            System.out.println("Volgende input wordt verwacht: <getal1> <operator> <getal2>");  
            return;  
        }  
  
        int getal1 = Integer.parseInt(args[0]);  
        char operator = args[1].charAt(0);  
        int getal2 = Integer.parseInt(args[2]);  
  
        Rekenblad b = new Rekenblad(getal1, getal2, operator);  
        System.out.println(b.rekenUit());  
    }  
}
```

```
> C:\voorbeeldenJava> java RekenenViaParametersMain
```

```
> C:\voorbeeldenJava> Volgende input wordt verwacht: <getal1> <operator> <getal2>
```

```
> C:\voorbeeldenJava> java RekenenViaParametersMain 12 + 18
```

```
> C:\voorbeeldenJava> 30
```

```

public class RekenenViaParametersMainBis {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Als input wordt verwacht:
                                \n <getal1> <getal2>
                                \n of
                                \n <getal1> <getal2> <operator>");
            return;
        }

        int getal1 = Integer.parseInt(args[0]);
        int getal2 = Integer.parseInt(args[1]);

        char operator;
        if (args.length == 3) {
            operator = args[2].charAt(0); //via parameters van de main
        } else {
            System.out.println("Welke bewerking wil je doen?");
            operator = new Scanner(System.in).next().charAt(0); //interactief
        }

        Rekenblad b = new Rekenblad(getal1, getal2, operator);
        System.out.println(b.rekenUit());
    }
}

```

```

> java RekenenViaParametersMainBis 12 8 +
> 20

```

```

> java RekenenViaParametersMainBis 2 18
> Welke bewerking wil je doen?
> *
> 36

```

# Meerdere objecten van een klasse

```
public class DemoSchaap {  
    public static void main(String[] args) {  
        Schaap s1 = new Schaap();  
        s1.blaat();  
        s1.blaat();  
        s1.slaap();  
  
        Schaap s2 = new Schaap();  
        for (int i = 0; i < 3; i++) {  
            s2.blaat();  
        }  
    }  
}
```

Beee  
Beee  
Zzzz

Beee  
Beee  
Beee

s1 →



s2 →



Referenties  
op stack

Objecten  
op heap

# Niet printen in logische klasse!

DemoSchaap.java



```
public class Schaap {  
    public void blaas() {  
        System.out.println("Beee");  
    }  
  
    public void slaap() {  
        System.out.println("Zzzz");  
    }  
}
```

```
public class DemoSchaap {  
    public static void main(String[] args) {  
        Schaap[] schaapjes = new Schaap[10];  
  
        for (int i = 0; i < schaapjes.length; i++) {  
            schaapjes[i] = new Schaap();  
        }  
  
        for (int i = 0; i < schaapjes.length; i+=2) {  
            System.out.println(schaapjes[i].slaap());  
        }  
    }  
}
```



```
public class Schaap {  
    public String blaat() {  
        return "Beee";  
    }  
  
    public String slaap() {  
        return "Zzzz";  
    }  
}
```

```
Zzzz  
Zzzz  
Zzzz  
Zzzz  
Zzzz
```

# GUI's hebben geen consolevenster en verzorgen hun uitvoer op andere manier

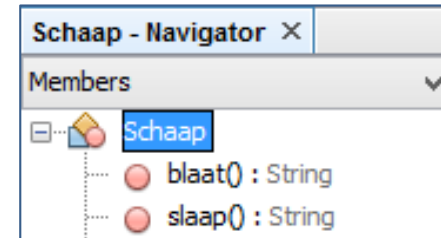
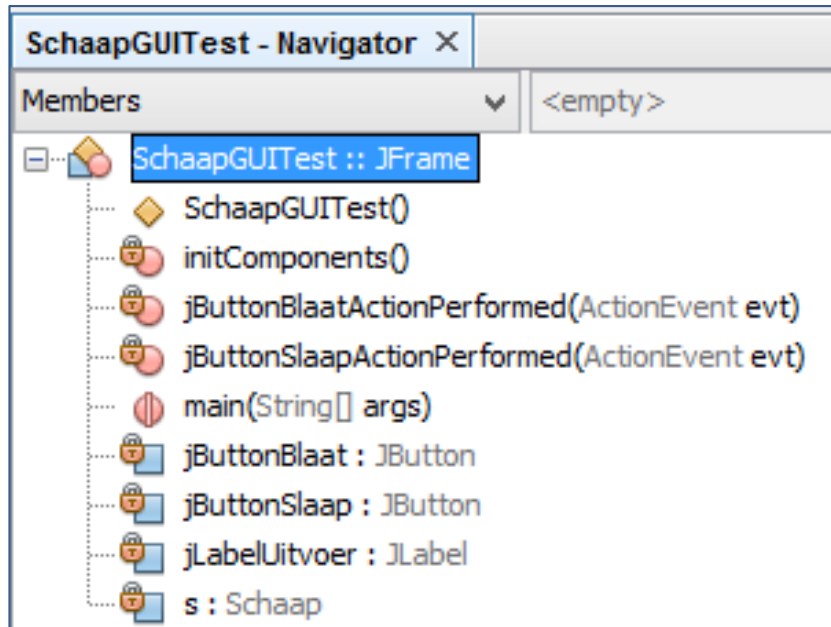
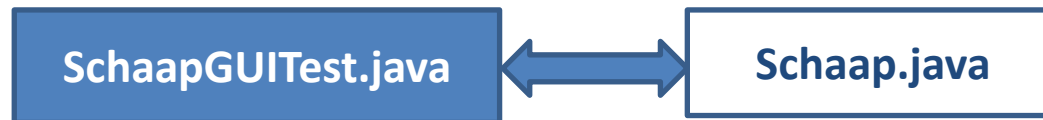


```
this.jLabelUitvoer.setText(s.blaas());
```



```
this.jLabelUitvoer.setText(s.slaap());
```

# Netbeans Navigator venster





# Statische vs. niet-statische methode

`public void slaap()`

➔ aanspreken via 'object' van de klasse Schaap

```
Schaap s1 = new Schaap();  
s1.slaap();
```

`public static void slaap()`

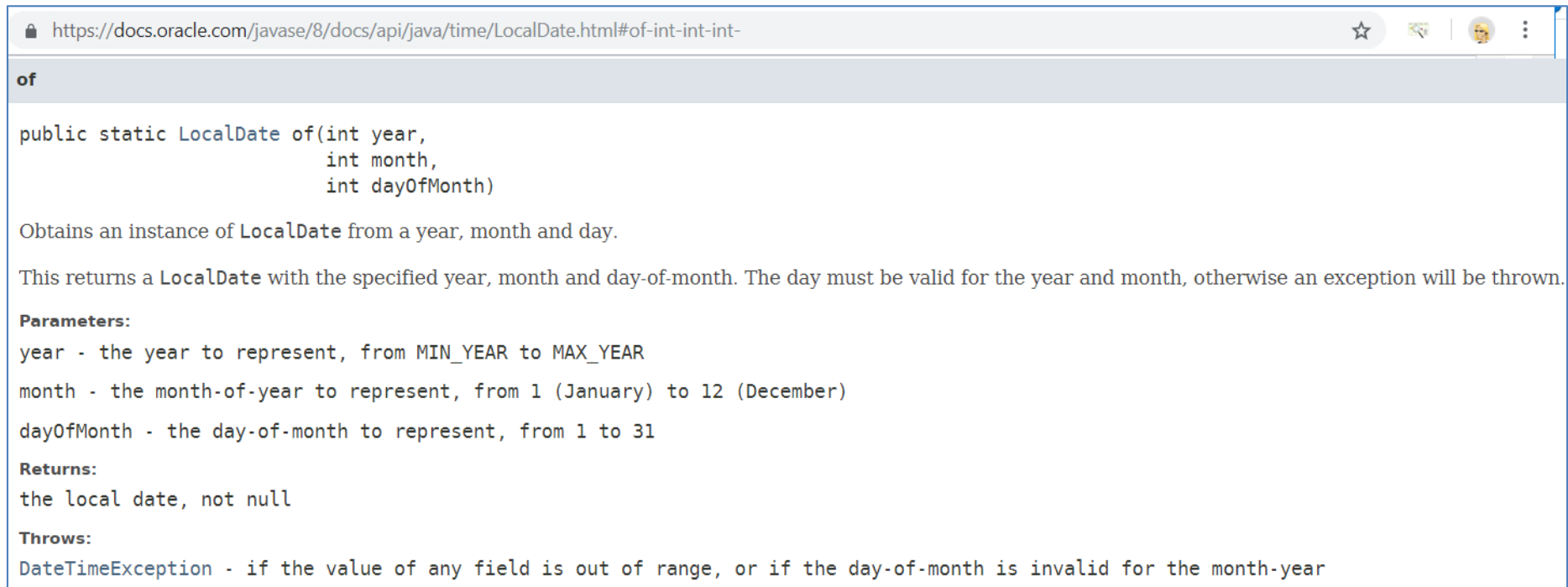
➔ aanspreken op klasseniveau

```
Schaap.slaap();
```

Niet zinvol in dit geval!!

# LocalDate.of

<https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html#of-int-int-int->



A screenshot of a web browser displaying the Oracle Java API documentation for the `LocalDate.of` method. The browser's address bar shows the URL `https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html#of-int-int-int-`. The page title is "of". The documentation includes the following code snippet:

```
public static LocalDate of(int year,
                           int month,
                           int dayOfMonth)
```

Obtains an instance of `LocalDate` from a year, month and day.

This returns a `LocalDate` with the specified year, month and day-of-month. The day must be valid for the year and month, otherwise an exception will be thrown.

**Parameters:**

- `year` - the year to represent, from `MIN_YEAR` to `MAX_YEAR`
- `month` - the month-of-year to represent, from 1 (January) to 12 (December)
- `dayOfMonth` - the day-of-month to represent, from 1 to 31

**Returns:**

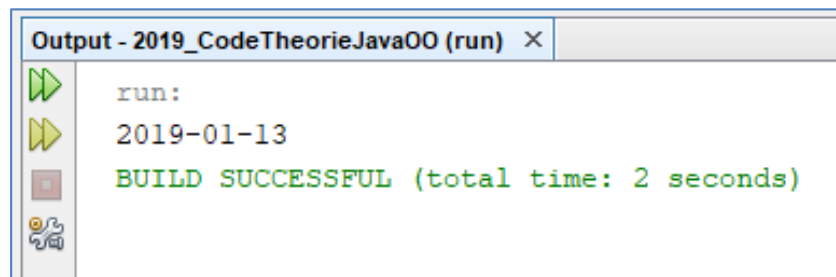
the local date, not null

**Throws:**

`DateTimeException` - if the value of any field is out of range, or if the day-of-month is invalid for the month-year

```
import java.time.LocalDate;

public class DemoLocalDate {
    public static void main(String[] args) {
        LocalDate today = LocalDate.of(2019, 1, 13);
        System.out.println(today);
    }
}
```



A screenshot of a Java IDE's output window titled "Output - 2019\_CodeTheorieJava00 (run)". The window shows the output of the program execution:

```
run:
2019-01-13
BUILD SUCCESSFUL (total time: 2 seconds)
```

# Statische methoden

```
public class Demo {  
    public static void main(String[] args) {  
        Student[] studenten = new Student[] {  
            new Student("Kris", 18),  
            new Student("Karel", 19),  
            new Student("Peter", 18)  
        };  
        drukStudenten(studenten);  
        sorteerOpLeeftijd(studenten);  
        drukStudenten(studenten);  
    }  
}
```

```
private static void sorteerOpLeeftijd(Student[] studenten) {  
    //implementatie van het bubblesort sorteeralgoritme  
    for (int i = 0; i < studenten.length; i++) {  
        for (int j = i + 1; j < studenten.length; j++) {  
            if (studenten[i].leeftijd > studenten[j].leeftijd) {  
                Student temp = studenten[i];  
                studenten[i] = studenten[j];  
                studenten[j] = temp;  
            }  
        }  
    }  
}
```

```
private static void drukStudenten(Student[] studenten) {  
    for (Student s : studenten) {  
        System.out.println(s.naam + " (" + s.leeftijd + ")");  
    }  
}
```

Noot: Later zal je zien dat het niet nodig is om 'zelf' een sorteeralgoritme te implementeren (cf. *Comparable* – *compareTo*)

Kris (18)  
Karel (19)  
Peter (18)

Kris (18)  
Peter (18)  
Karel (19)

# Variant met statische methoden in andere klasse

```
public class Demo {  
    public static void main(String[] args) {  
        Student[] studenten = new Student[] {  
            new Student("Kris", 18),  
            new Student("Karel", 19),  
            new Student("Peter", 18)  
        };  
        StudentHelper.druk(studenten);  
        StudentHelper.sorteerOpLeeftijd(studenten);  
        StudentHelper.druk(studenten);  
    }  
}
```

```
public class StudentHelper {  
    public static void sorteerOpLeeftijd(Student[] studenten) {...}  
    public static void sorteerOpNaam(Student[] studenten) {...}  
    public static void druk(Student[] studenten) { ...}  
    private static void swap(Student[] rij, int i, int j) {...}  
}
```

# OO-approach: Niet-statische variant

```
public class Demo {  
    public static void main(String[] args) {  
        Student[] studenten = new Student[] { ..., ..., ... };  
        StudentenClub club = new StudentenClub(studenten);  
        club.drukStudenten();  
  
        club.sorteerOpNaam();  
        club.drukStudenten();  
    }  
}
```

**Presentatieklasse**

**Logische klasse**

```
public class StudentenClub {  
    private Student[] studenten;  
  
    public StudentenClub(Student[] studenten) {  
        this.studenten = studenten;  
        sorteerOpLeeftijd();  
    }  
  
    private void sorteerOpLeeftijd() { ...}  
    private void swap(Student[] rij, int i, int j) {...}  
  
    public void sorteerOpNaam() { ...}  
    public void drukStudenten() {...}  
}
```

# Statisch vs. niet-statisch veld

```
public class Demo {  
    public static void main(String[] args) {  
        Student[] studenten = {  
            new Student("Kristien"),  
            new Student("Katja"),  
            new Student("Peter")  
        };  
  
        for (Student s : studenten) {  
            System.out.println(s.studNr + ": " + s.naam);  
        }  
    }  
}
```

1: Kristien  
2: Katja  
3: Peter

Wat is de leeftijd van deze 3 studenten ?!

```
public class Student {  
    static int teller;  
    int studNr;  
    String naam;  
    int leeftijd;  
  
    public Student(String naam) {  
        studNr = ++teller;  
        this.naam = naam;  
    }  
  
    public Student(String n, int l) {  
        studNr = ++teller;  
        naam = n;  
        leeftijd = l;  
    }  
}
```

# Statische constante

= een veld dat door alle objecten van de klasse gedeeld wordt én slechts 1 keer een waarde kan krijgen, ofwel bij declaratie ofwel in de constructor

```
public class Poll {  
    static final int MAX_DEELNEMERS = 10;  
}
```

```
public class SalesData {  
    static final int MAX_NR_WEEKS;  
    double[][] sales;  
  
    public SalesData(int aantal) {  
        MAX_NR_WEEKS = aantal;  
        sales = new double[aantal][8];  
    }  
}
```

```
public class HighScores {  
    static final int MAX_ITEMS;  
    String game;  
    String[][] scores; //wie welke score  
  
    public HighScores(String game, int max) {  
        MAX_ITEMS = max;  
        this.game = game;  
        scores = new String[MAX_ITEMS][2];  
    }  
}
```

# Standaard & niet-standaard constructor

- Bij aanwezigheid van een niet-standaard constructor wordt GEEN standaard constructor meer gegenereerd. Wil je die ook, dan moet je die expliciet schrijven
- Zorg voor consistente naamgeving constructor parameter t.o.v. veldnaam
- Vermijd null pointer excepties!

## Schaap

String kleur  
String locatie

public Schaap()  
public Schaap(String kleur, String locatie)

public String blaat()  
public String slaap()  
public String geefInfo()

```
public class Schaap {  
    String kleur;  
    String locatie;  
  
    public String (String kleur, String locatie) {  
        this.kleur = kleur;  
        this.locatie = locatie;  
    }  
}
```

```
    public Schaap() {  
        this.kleur = "wit";  
        this.locatie = "wei";  
    }  
}
```



# Publieke velden? Geen goed idee...

In klasse Persoon:

```
public class Persoon {  
    public String naam;  
    public String voornaam;  
    public int leeftijd;  
    public boolean geslacht;  
}
```

In Demo klasse:

```
Persoon p1 = new Persoon();  
p1.naam = "Van Assche";  
p1.voornaam = "Kristien";  
p1.leeftijd = -11;  
p1.geslacht = true;
```

# OPLOSSING: Privaat veld met 'gecontroleerde' toegang via setter-methode 😊

In klasse Persoon:

```
public class Persoon {  
    ...  
    private int leeftijd;  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd >= 0) {  
            this.leeftijd = leeftijd;  
        }  
    }  
}
```

# Aangepast gebruik vanuit demoklasse

```
Persoon p1 = new Persoon();  
p1.naam = "Van Assche";  
p1.voornaam = "Kristien";  
p1.setLeeftijd(-11); //ongeldige leeftijd wordt niet doorgevoerd  
p1.geslacht = true;  
System.out.println(p1.leeftijd); //veld niet toegankelijk...
```

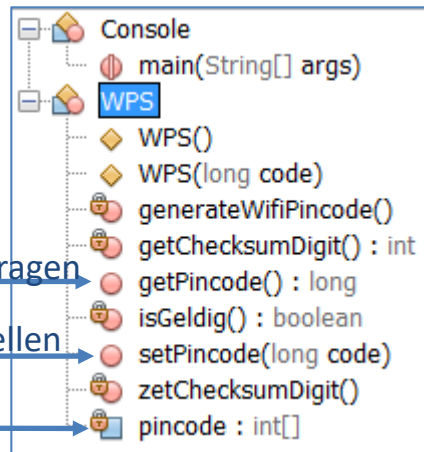
# OPLOSSING: Voorzie ook een getter voor toegang tot de private data 😊

```
public class Persoon {  
    ...  
    private int leeftijd;  
  
    public void setLeeftijd(int leeftijd) {  
        if (leeftijd >= 0) {  
            this.leeftijd = leeftijd;  
        }  
    }  
  
    public int getLeeftijd() {  
        return leeftijd;  
    }  
}
```

# Aangepast gebruik vanuit demoklasse

```
Persoon p1 = new Persoon();  
p1.naam = "Van Assche";  
p1.voornaam = "Kristien";  
p1.setLeeftijd(-11); //ongeldige leeftijd wordt niet doorgevoerd  
p1.geslacht = true;  
System.out.println(p1.getLeeftijd());
```

# Voorbeeld: Wifi pincode



```
public class Console {
    public static void main(String[] args) {
        WPS wps1 = new WPS();
        System.out.println("pincode1: " + wps1.getPincode());

        wps1.setPincode(75244989);
        System.out.println("pincode1': " + wps1.getPincode());

        WPS wps2 = new WPS(98882557);
        System.out.println("pincode2: " + wps2.getPincode());

        wps2.setPincode(12345678);
        System.out.println("pincode2': " + wps1.getPincode());
    }
}
```

```
run:
pincode1: 18352955
pincode1': 75244989
pincode2: 98882557
Exception in thread "main" java.lang.IllegalArgumentException: ongeldige pincode
|   at logica.WPS.setPincode(WPS.java:25)
|   at presentatie.Console1.main(Console1.java:18)
C:\Users\kristien.vanassche\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

# De stack, de heap en de statische ruimte

Maak een rij voor de dagen van de week en vul er enkele op.  
Wat wordt bewaard op resp. stack, heap en wat in statische ruimte?

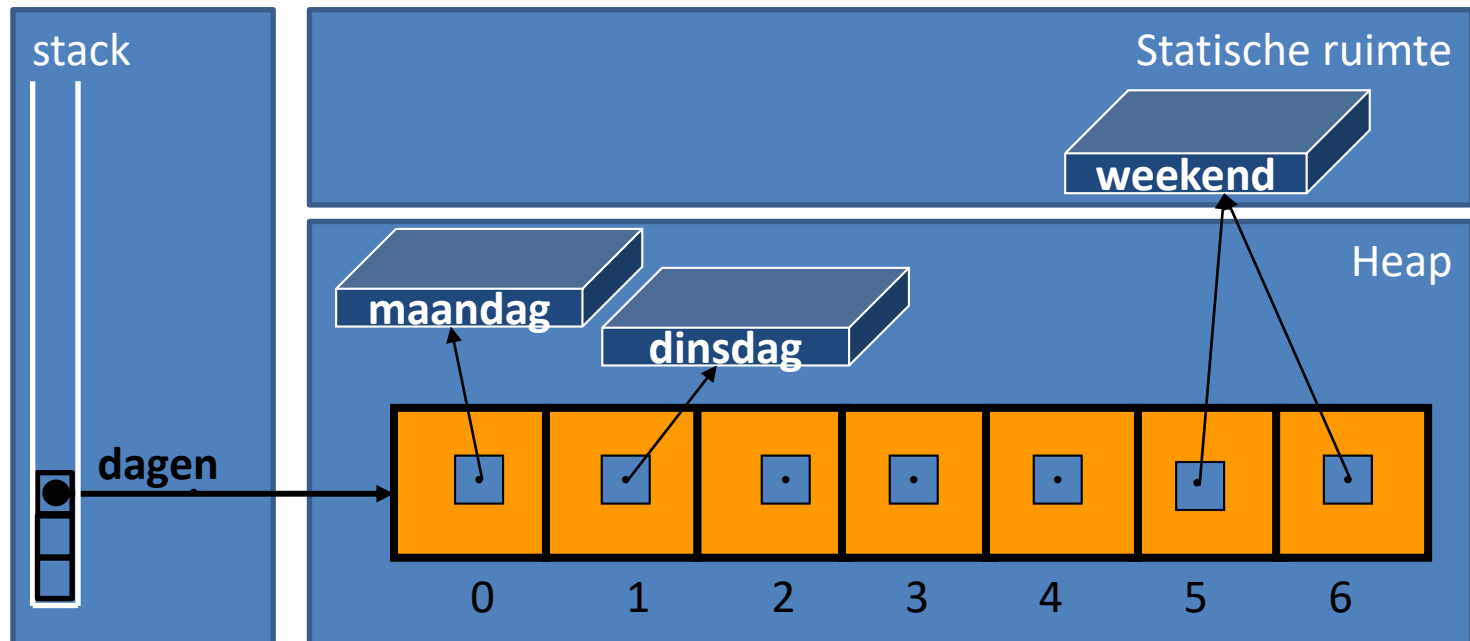
```
String[] dagen = new String[7]; //STAP1+STAP2
```

```
dagen[0] = new String("maandag"); ➔ niet-standaard constructor !
```

```
dagen[1] = new String("dinsdag"); ➔ niet-standaard constructor !
```

```
dagen[5] = "weekend";
```

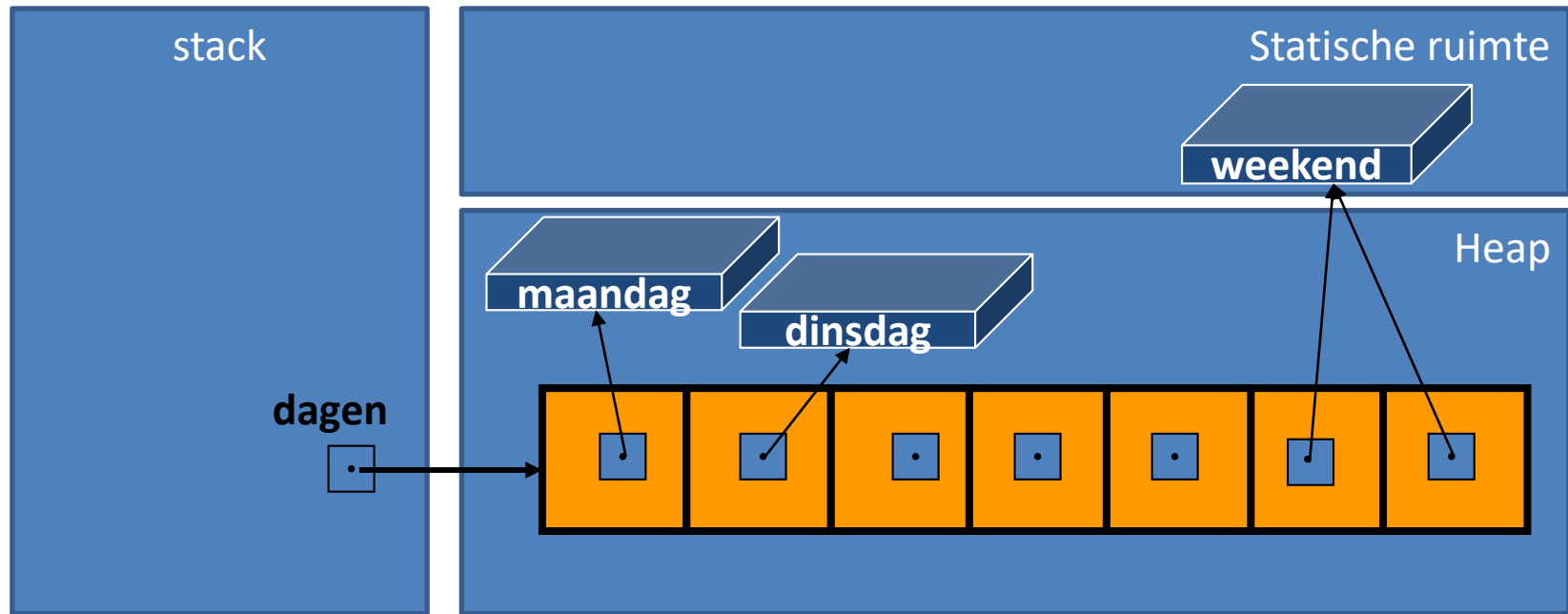
```
dagen[6] = "weekend";
```



OPM: `dagen[2].charAt(0); //NullPointerException`



Druk de dagen af op het consolevenster



Klassieke for-lus

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

foreach-lus

```
for (String dag : dagen) {  
    System.out.println(dag);  
}
```

```
maandag  
dinsdag  
null  
null  
weekend  
weekend
```

# Cf. semester 1 – hoofdstuk Arrays

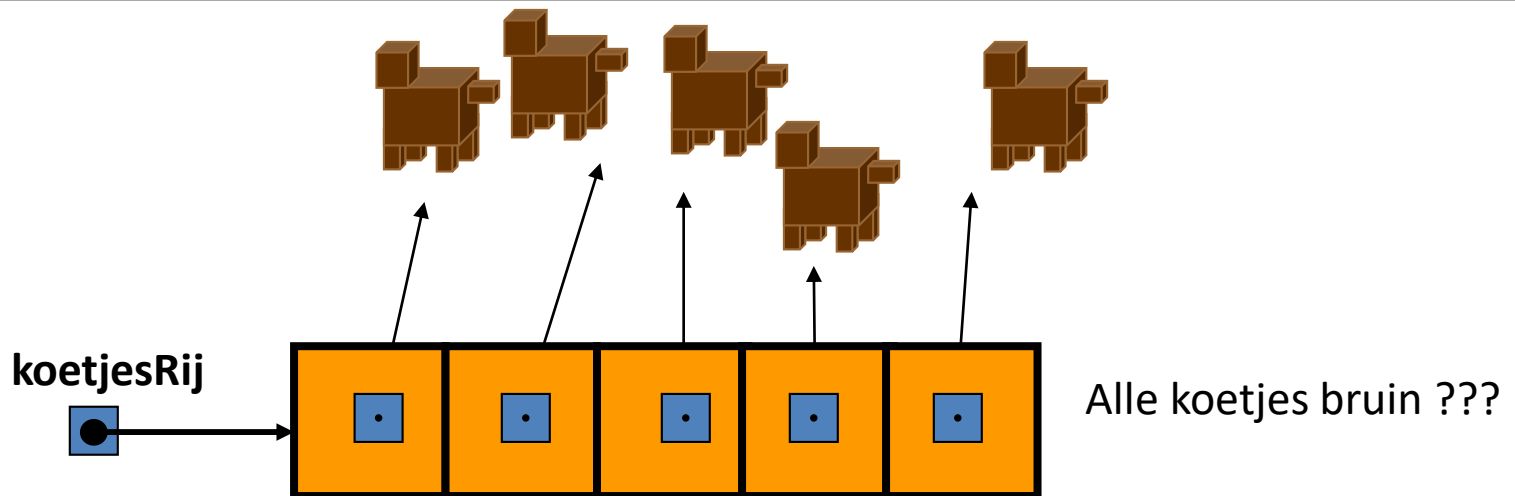
Zet 5 koeien in een rij

```
Koe[] koetjesRij; //STAP1
```

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

```
//STAP3
```

```
for (int i = 0; i < koetjesRij.length; i++) {  
    koetjesRij[i] = new Koe("bruin");  
}
```



```
for (int i = 0; i < koetjesRij.length; i++)  
    System.out.println(koetje);  
}
```

Koe@15db9742  
Koe@6d06d69c  
Koe@7852e922  
Koe@4e25154f  
Koe@70dea4e


```
for (Koe koetje : koetjesRij) {  
    System.out.println(koetje.getKleur());  
}
```

bruin  
bruin  
bruin  
bruin  
bruin

## 2-DIM RIJ

```
String[][] emoticons = new String[3][2];  
emoticons[0][0] = new String(":-)");  
emoticons[0][1] = new String("smiling");  
  
emoticons[1][0] = new String(":-D");  
emoticons[1][1] = new String("laughing");  
  
emoticons[2][0] = new String(":'-(");  
emoticons[2][1] = new String("crying");
```

emoticons



: -)	smiling
: -D	laughing
: '-(	crying

```

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
        System.out.print(emoticons[i][j] + "\t");
    }
    System.out.println("");
}

```

```

for (int i = 0; i < emoticons.length; i++) {
    for (int j = 0; j < emoticons[i].length; j++) {
        System.out.print(emoticons[i][j] + "\t");
    }
    System.out.println("");
}

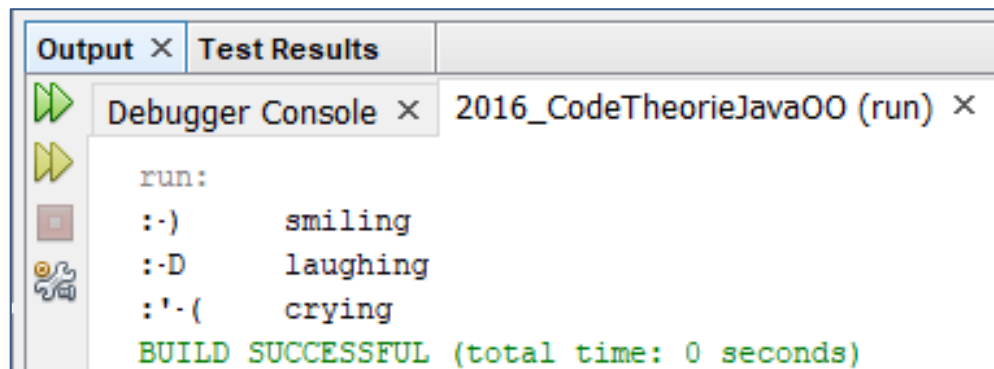
```

```

for (String[] row : emoticons) {
    for (String elt : row) {
        System.out.print(elt + "\t");
    }
    System.out.println("");
}

```

Uitvoer:



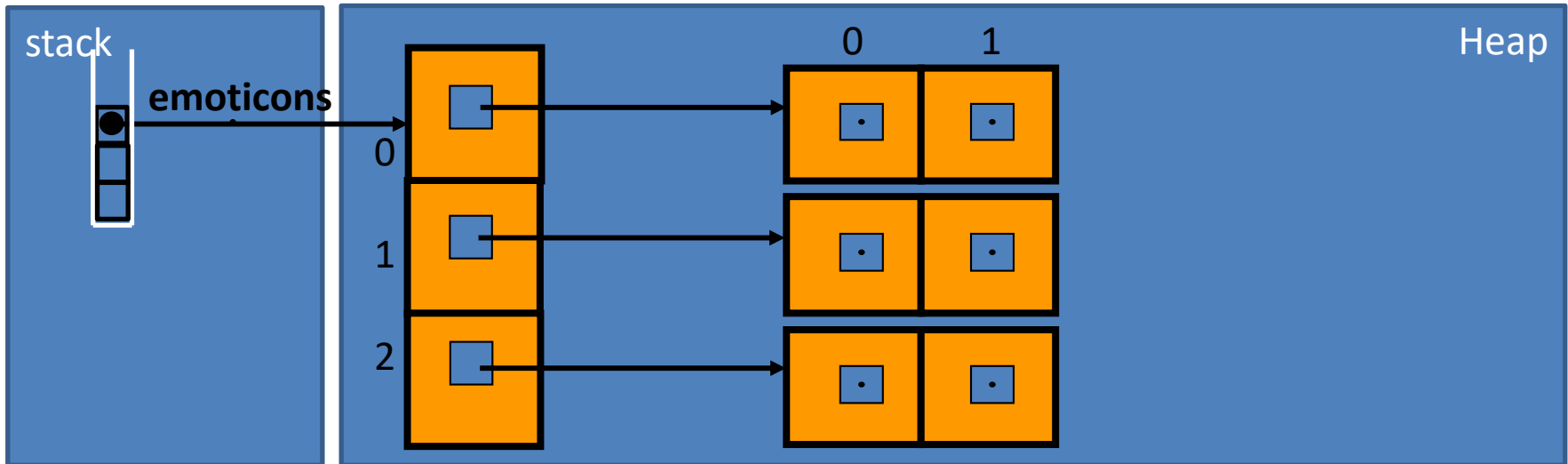
```

Output × Test Results
run:
:-)    smiling
:D     laughing
:\'-(   crying
BUILD SUCCESSFUL (total time: 0 seconds)

```

## Interne voorstelling van 2-DIM rij

```
String[][] emoticons = new String[3][2];
```



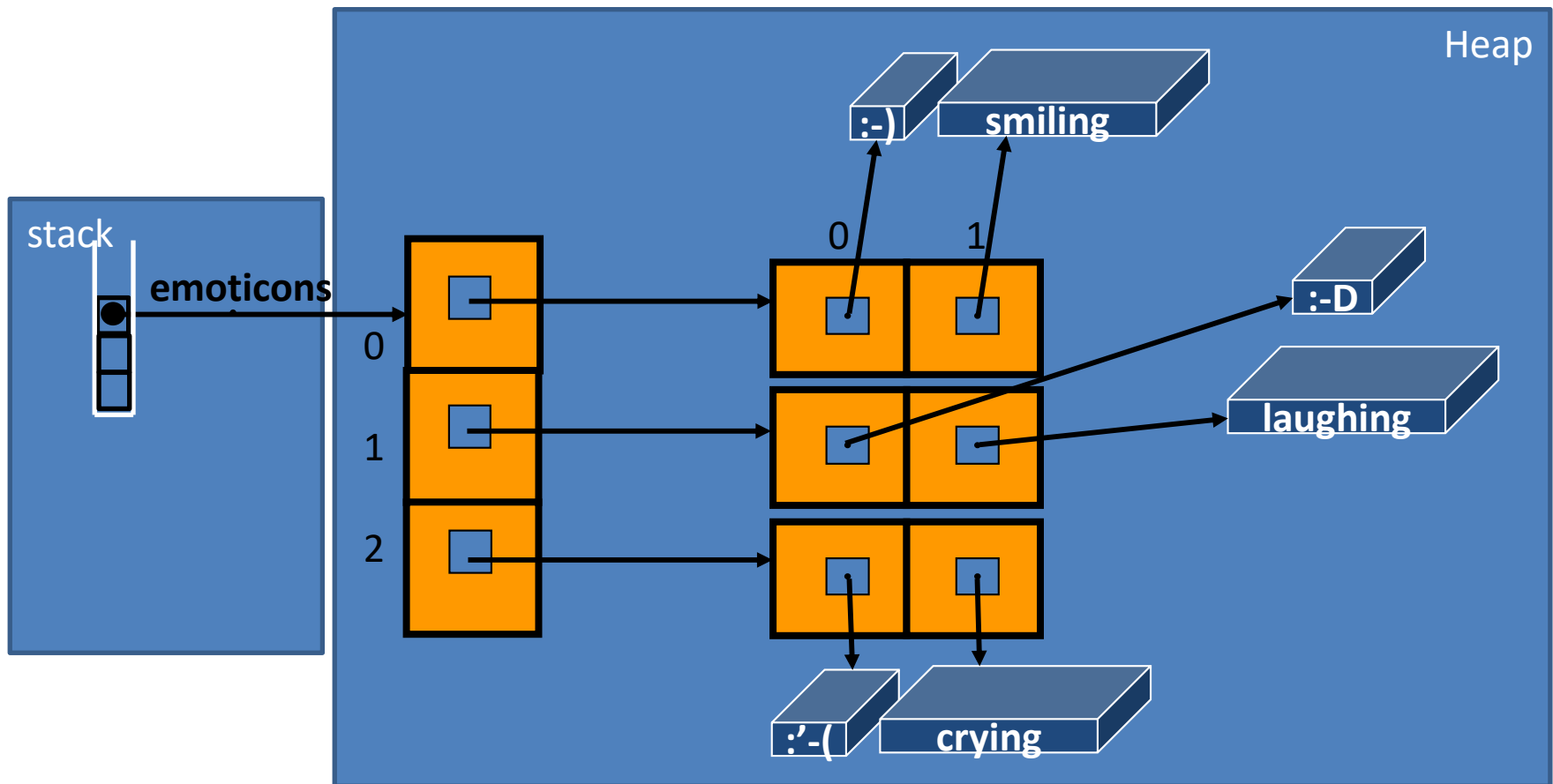
```
for (String[] row : emoticons) {  
    for (String elt : row) {  
        System.out.print(elt + "\\t");  
    }  
    System.out.println("");  
}
```

Uitvoer:

```
Output × Test Results  
Debugger Console × 2016_CodeTheorieJavaOO (run) ×  
run:  
null    null  
null    null  
null    null  
BUILD SUCCESSFUL (total time: 0 seconds)
```

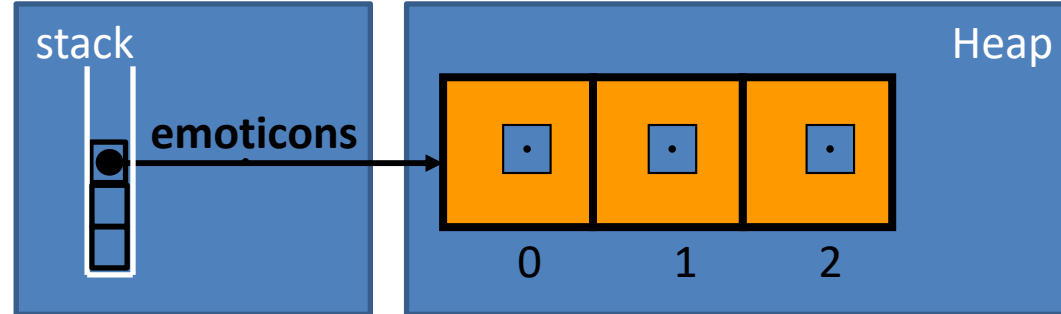
```
String[][] emoticons = new String[3][2];
```

```
emoticons[0][0] = new String(":-)");  
emoticons[0][1] = new String("smiling");  
emoticons[1][0] = new String(":-D");  
emoticons[1][1] = new String("laughing");  
emoticons[2][0] = new String(":'-(");  
emoticons[2][1] = new String("crying");
```



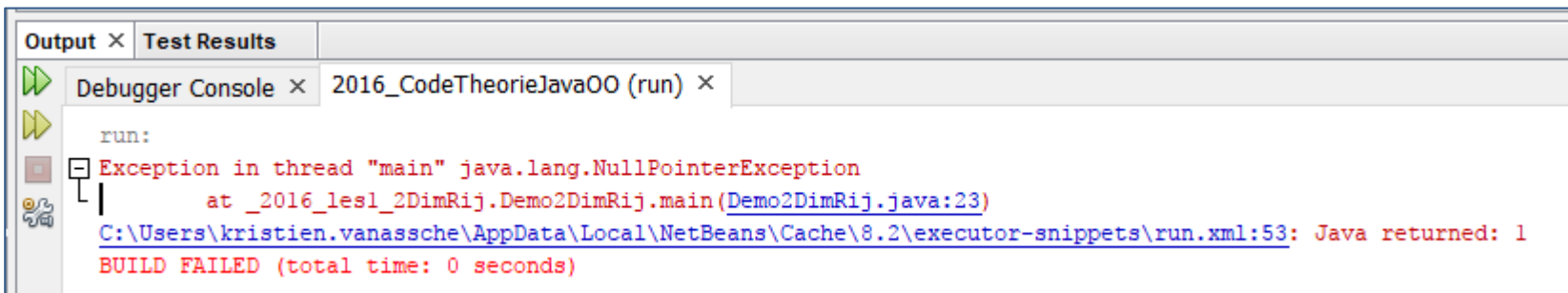
## Alternatief

```
String[][] emoticons = new String[3][];
```



```
for (String[] row : emoticons) {  
    for (String elt : row) {  
        System.out.print(elt + "\t");  
    }  
    System.out.println("");  
}
```

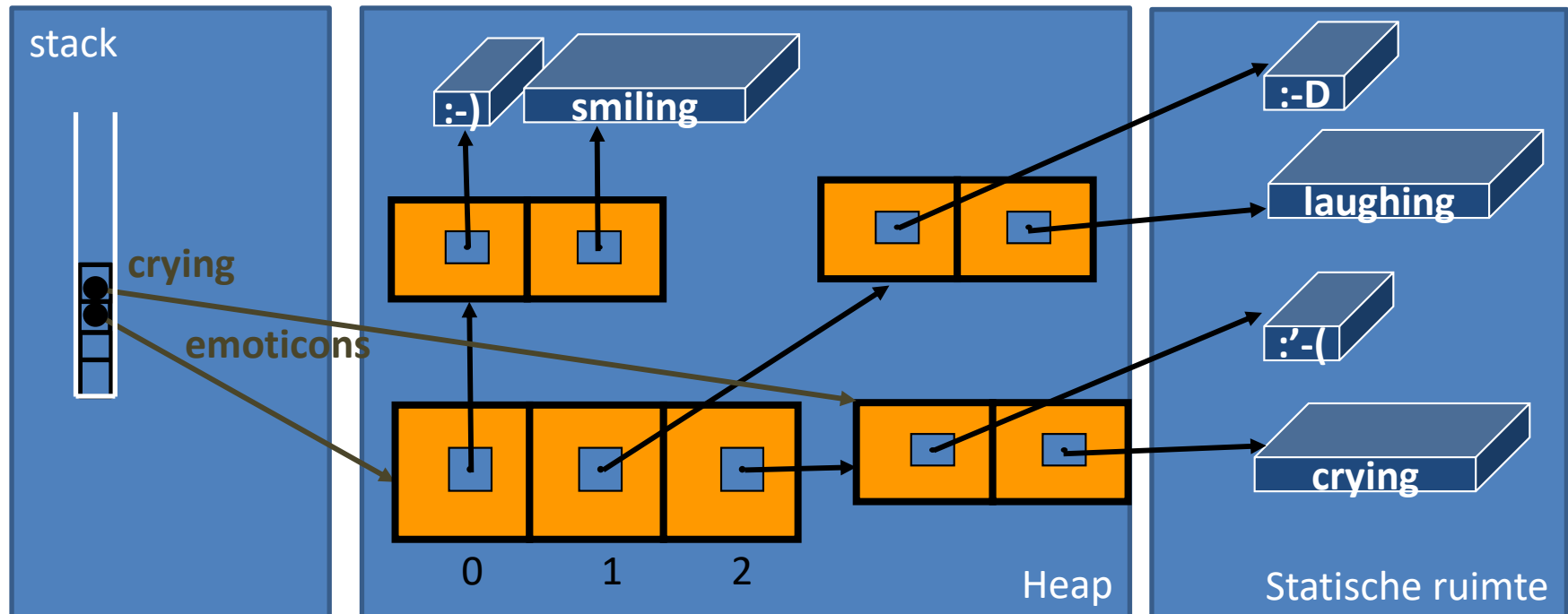
## Uitvoer:





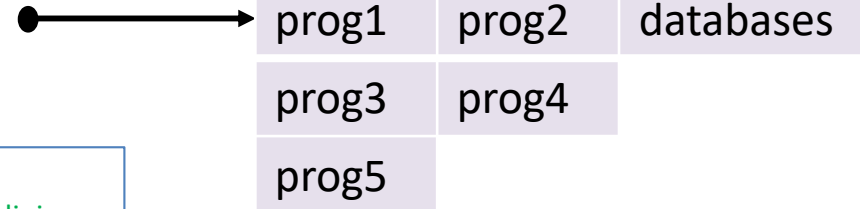
Alloceren  
van tweede dimensie:

```
String[][] emoticons = new String[3][];  
  
emoticons[0] = new String[]{new String( ":-)" ,  
                             new String("smiling")};  
  
emoticons[1] = new String[]{" :-D", "laughing"};  
  
String[] crying = {" :-(" , "crying"};  
emoticons[2] = crying;
```



# Jagged array

programma



Cf. project Java Fundamentals

```
public class Studieprogramma {
    private static final int MAX_JAREN; // maximum aantal toegelaten studiejaren
    private String[][] programma; // declaratie van 2-DIM array van vakken

    public StudieProgramma() {
        this(5); //constructor overloading
    }

    public Studieprogramma(int maxJaren) {
        MAX_JAREN = maxJaren;
        programma = new String[this.MAX_JAREN][]; //init eerste dimensie
    }

    public Studieprogramma(int maxJaren, String[] vakkenEersteJaar) {
        this(maxJaren); //constructor overloading

        programma[0] = vakkenEersteJaar; //init tweede dimensie, rij 0
    }

    public void inschrijven(int jaar, String[] vakken) {
        if (jaar >= 0 && jaar < MAX_JAREN) {
            programma[jaar] = vakken; //init tweede dimensie, rij 'jaar'
        }
    }
}
```

```
for (String[] rij : programma) {
    for (String elt : rij) {
        System.out.print(elt + "\t");
    }
    System.out.println("");
}
```

Uitvoer:

```
Output × Test Results
Debugger Console × 2016_CodeTheorieJavaOO (run) ×

run:
prog1  prog2  databases
prog3  prog4
prog5
BUILD SUCCESSFUL (total time: 0 seconds)
```

```


public class Meting {
    private double[][] meetwaarden; // declaratie van 2-DIM array van meetwaarden

    public Meting() {
        meetwaarden = new double[12][]; //init eerste dimensie
    }

    public void zetWaardenVoorMaand(int maand, double[] meetwaarden) {
        if (maand >= 0 && maand < 12) {
            meetwaarden[maand] = meetwaarden; //init tweede dimensie
        }
    }
}

```

## meetwaarden



JAN	1	1.5	0.5	...	2	5	4.5	3
FEB	10	5	6	...				
MRT	9	11	12.5	...	8	8.5	10	9.5
APR	12	13	16	...	12.5	9	10.5	
...	...							
...	...							
DEC	3	3.5	2	...				

# Geheugenvoorstelling van een concreet object / een array

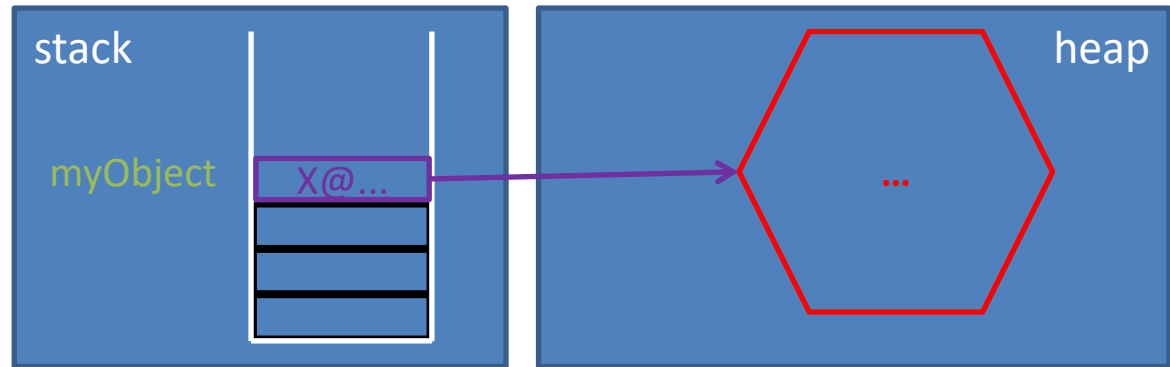
# Geheugenvoorstelling van een concreet object / een array

Java code

Geheugenvoorstelling

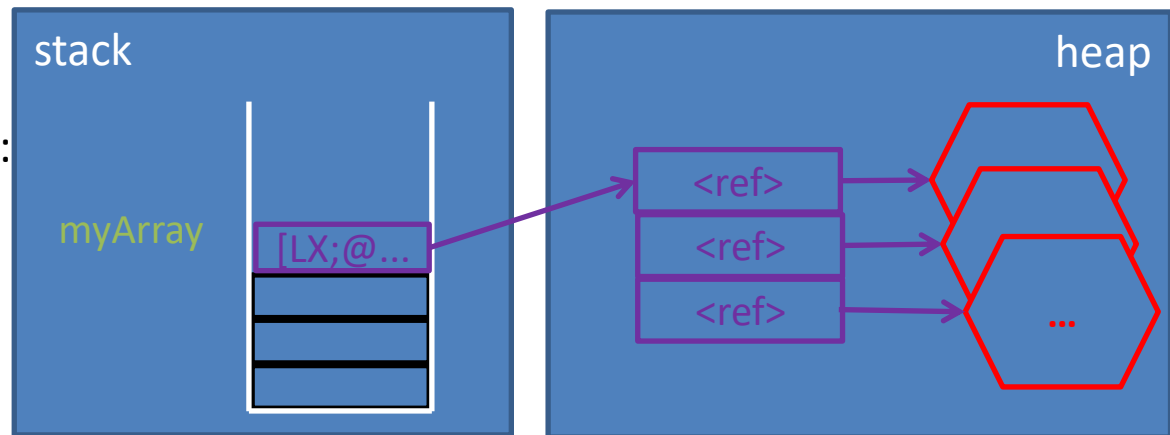
Aanmaken van 1 object:

```
X myObject = new X(...);
```



Aanmaken van een rij van objecten:

```
X[] myArray = new X[3];
```



# Rij als datatype

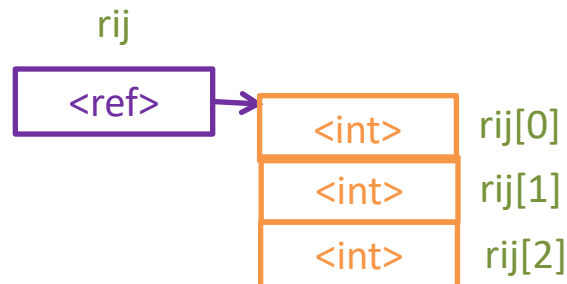
- Rij-variabele is een referentie



- Rij-elementen zijn steeds van hetzelfde type

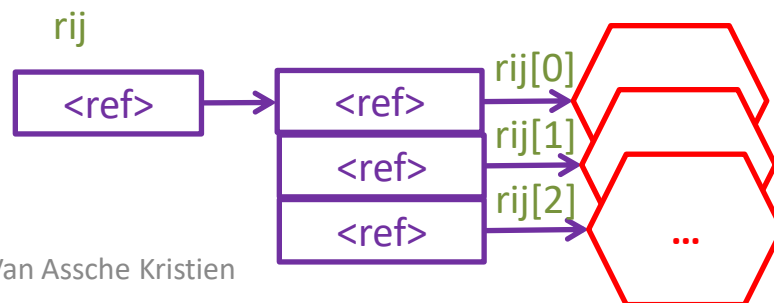
- van het primitieve type: bv. rij van getallen

(elk element is een elementaire waarde)



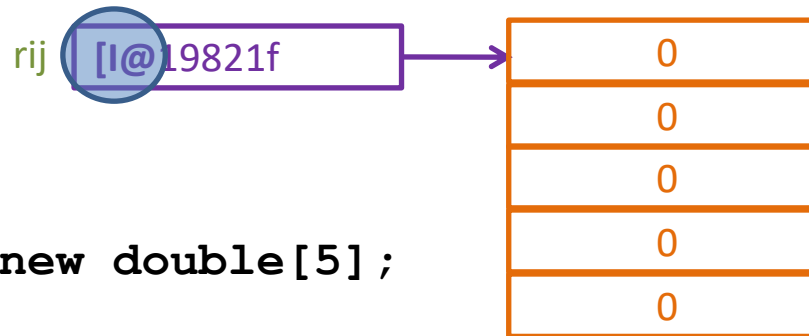
- van het object-type: bv. rij van koetjes, rij van strings, ...

(elk element is een referentie naar een object)

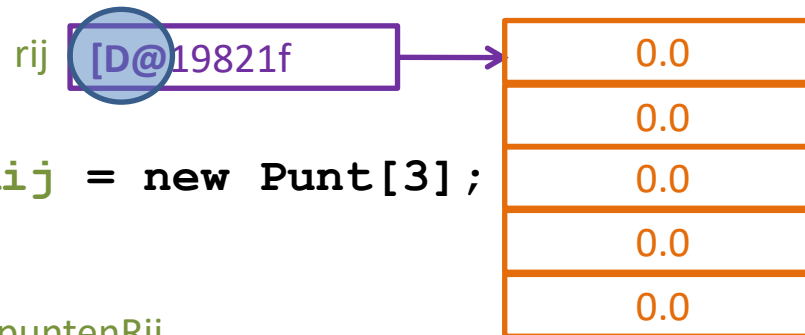


# Voorstelling van 1-DIM rij

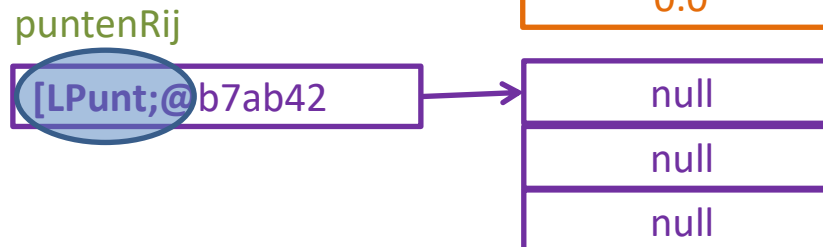
```
int[] rij = new int[5];
```



```
double[] rij = new double[5];
```



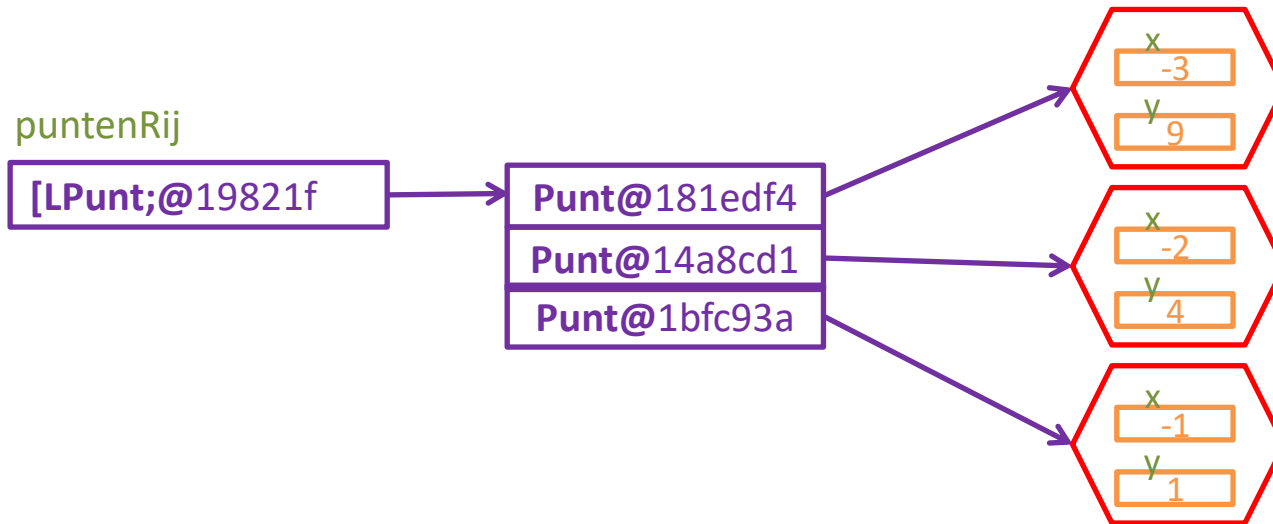
```
Punt[] puntenRij = new Punt[3];
```



```
Punt[] puntenRij = new Punt[3];

int x = -3;
int y;

for (int i = 0; i < puntenRij.length; i++) {
    y = kwadraat(x);
    puntenRij[i] = new Punt(x, y);
    x++;
}
```

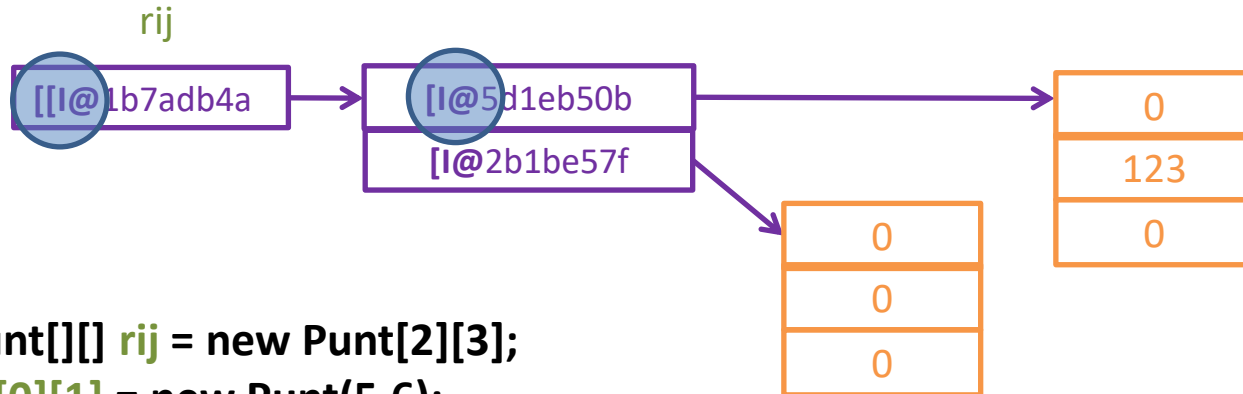




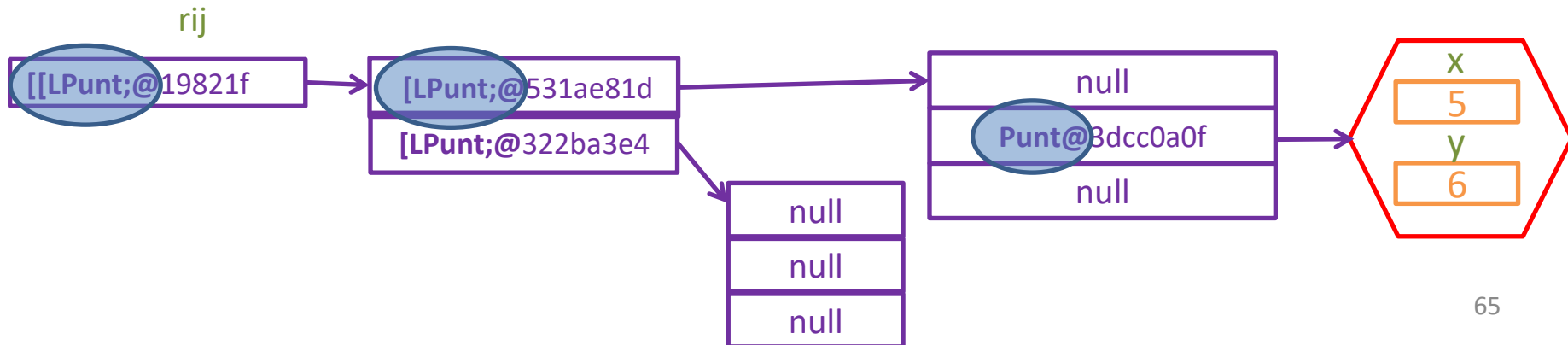
# Voorstelling van 2-DIM rij

0	123	0
0	0	0

```
int[][] rij = new int[2][3];  
rij[0][1] = 123;
```



```
Punt[][] rij = new Punt[2][3];  
rij[0][1] = new Punt(5,6);
```



# Afdrukken van locaties/waarden in het geheugen (stack/heap)

```
public void startRijVanGeheleGetallen() {  
    int[][] rij = new int[2][3];  
    rij[0][1] = 123;
```

```
    System.out.println( rij );
```

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

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

```
        }
```

```
    }
```

```
}
```

```
[[I@587b8be7
```

```
[I@171e1813
```

```
0  
123
```

```
0
```

```
[I@38be9340
```

```
0
```

```
0
```

```
0
```

```
public void startRijVanPunten() {  
    Punt[][] rij = new Punt[2][3];  
    rij[0][1] = new Punt(5,6);
```

```
    System.out.println( rij );
```

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

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

```
        }
```

```
    }
```

```
}
```

```
[[LPunt;@15b57dcb
```

```
[LPunt;@3e55a58f
```

```
null
```

```
Punt@68e86f41
```

```
null
```

```
[LPunt;@73e04a35
```

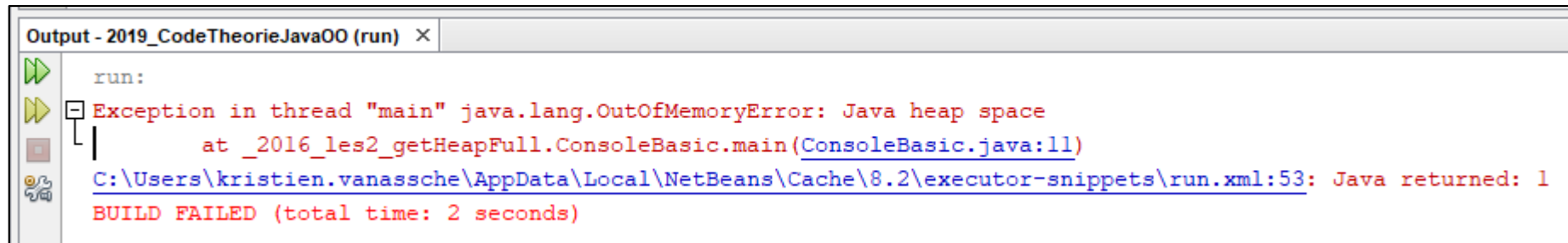
```
null
```

```
null
```

```
null
```

# Heap full – voorbeeld1

```
public class ConsoleBasic {  
    public static void main(String[] args) {  
        double[][][] data = new double[1000][1000][1000];  
    }  
}
```



The screenshot shows a Java IDE's output window titled "Output - 2019\_CodeTheorieJava00 (run)". It displays the following text:

```
run:  
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  
    at _2016_les2_getHeapFull.ConsoleBasic.main(ConsoleBasic.java:11)  
C:\Users\kristien.vanassche\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 2 seconds)
```

The error message indicates a "Java heap space" issue, which is a common cause of a "Heap full" error in Java. The stack trace points to the `main` method in `ConsoleBasic.java` at line 11, which corresponds to the line where the large 3D array is created in the code snippet above.

# Heap full – voorbeeld2

```
public class Console {  
    public static void main(String[] args) {  
        BigClass[] bc = new BigClass[100];  
        int i = 0;  
  
        while (true) {  
            bc[i] = new BigClass();  
            System.out.println("" + (++i));  
        }  
    }  
}
```

```
public class BigClass {  
    private double[][] data;  
  
    public BigClass() {  
        data = new double[1000][10000];  
    }  
}
```

Debugger Console × 2016\_CodeTheorieJavaOO (run) ×

```
run:  
1  
2  
3  
[ Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  
  at _2016_les2_getHeapFull.BigClass.<init>(BigClass.java:11)  
  at _2016_les2_getHeapFull.Console.main(Console.java:13)  
C:\Users\kristien.vanassche\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 6 seconds)
```

# Doordenkertje - vraag

```
public class Data {  
    private String[] a;  
    private int b;  
  
    public Data(String[] a, int b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public void update(){  
        for(int i = 0; i < a.length; i++) {  
            a[i] = a[i].toLowerCase();  
        }  
  
        b -= 10;  
    }  
}
```

```
public class Demo {  
    public static void main(String[] args) {  
        String[] a = {"EEN","TWEE"};  
        int b = 100;  
        Data data = new Data(a, b);  
        data.update();  
        System.out.println("a[1]=" + a[1] + "; b=" + b);  
    }  
}
```

## 2. Wat is de uitvoer van dit programma?

- a) a[1]=twee; b= 100
- b) a[1]=TWEE; b= 100
- c) a[1]=een; b= 90
- d) a[1]=twee; b= 90

# Doordenkertje - antwoord

```
public class Data {  
    private String[] a;  
    private int b;  
  
    public Data(String[] a, int b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public void update(){  
        for(int i = 0; i < a.length; i++) {  
            a[i] = a[i].toLowerCase();  
        }  
  
        b = b - 10;  
    }  
}
```

```
public class Demo {  
    public static void main(String[] args) {  
        String[] a = {"EEN", "TWEE"};  
        int b = 100;  
        Data data = new Data(a, b);  
        data.update();  
        System.out.println("a[1]=" + a[1] + "; b=" + b);  
    }  
}
```

Antwoord:

```
a[1]=twee; b=100
```

Bedenking:

Hoe komt het dat b niet, maar a[1] wel gewijzigd is?

# Verklaring

```
public class Data {  
    private String[] a;  
    private int b;  
  
    public Data(String[] aPar, int bPar) {  
        a = aPar;  
        b = bPar;  
    }  
  
    public void update(){  
        for(int i = 0; i < a.length; i++)  
            a[i] = a[i].toLowerCase();  
        b -= 10;  
    }  
  
    public String geefOverzichtVanDeVraag() {  
        return "a[1]=" + a[1] + "; b=" + b;  
    }  
}
```

- Scope van variabelen
- Principe van Pass-by-value

```
public class Demo {  
    public static void main(String[] args) {  
        String[] a = {"EEN", "TWEETWEET"};  
        int b = 100;  
        Data data = new Data(a, b);  
        data.update();  
        System.out.println("a[1]=" + a[1] + "; b=" + b);  
        System.out.println( data.geefOverzichtVanDeVraag());  
    }  
}
```

```
a[1]=twee; b=100  
a[1]=twee; b=90
```

# Toelichting 'pass-by-value'

Demo

```
String[] a = {"EEN", "TWEE"};  
int b = 100;
```

```
Data data = new Data(a, b);
```

Data

