HO
GENT

H5 ArrayList en arrays

Table of Contents

1. Doelstellingen

Na het bestuderen van dit hoofdstuk ben je in staat

- een **ArrayList** te **implementeren** in Java
- een **enhanced for-statement** te **implementeren** in Java
- een **eendimensionale array** te **implementeren** in Java
- het **verschil** tussen een ArrayList en een array te **kennen**
- een **tweedimensionale array** te **implementeren** in Java

2. Inleiding

In dit hoofdstuk maken we kennis met datastructuren waarin **meer dan één gegeven** kan bewaard worden. Alle gegevens binnen deze datastructuur dienen wel van **hetzelfde type** te zijn: ofwel zijn het objecten van dezelfde klasse, ofwel zijn het waarden van hetzelfde primitieve type.

Je gebruikt een ArrayList of array dus om gerelateerde groepen waarden te verwerken.

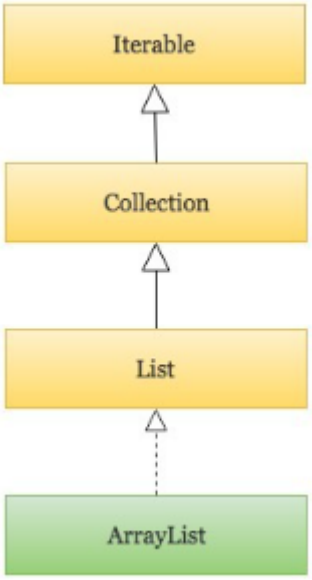
Voorbeeld: een lijst met Boek-objecten, een array met punten van studenten (type double), …

3. ArrayList

3.1. Definitie

Een ArrayList is een dynamische lijst. Dit wil zeggen dat de lijst gemakkelijk kleiner of groter kan gemaakt worden. Bovendien zijn in een ArrayList duplicaten mogelijk, dus hetzelfde gegeven kan meerdere keren voorkomen.

We maken hier gebruik van de generieke klasse ArrayList<E>, die eigenlijk resizable arrays (zie verder) voorstelt. Deze klasse is een implementatieklasse van de interface List<E>, die een sequentie van elementen uit de klasse E voorstelt. Een List<E> is dan weer een subinterface van Collection<E>, die de root interface is in de collection hiërarchie en dus gewoon een groep elementen van de klasse E, al dan niet met duplicaten en al dan niet met een volgorde voorstelt.



3.2. Terminologie

- element: één gegeven dat in de lijst is opgeborgen
- index: het volgnummer van het element in de lijst

De index van een ArrayList begint altijd bij 0!

- lengte of grootte: het aantal elementen dat de lijst bevat

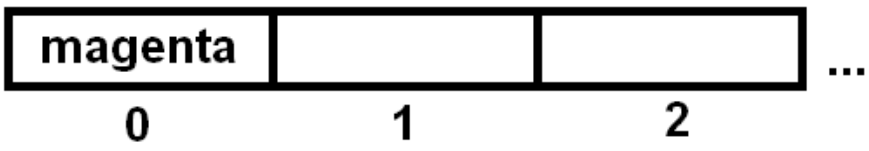
3.3. Illustratie van de methodes add, set, get en remove

In dit eerste voorbeeld gebruiken we een lijst van Strings die kleuren voorstellen.

```java
package cui;

import java.util.ArrayList;
import java.util.List;

public class ArrayListString
{
    public static void main(String[] args)
    {
        List<String> colorList = new ArrayList<>(); ①

        colorList.add("magenta"); ②
        colorList.add("yellow");  ③
        colorList.add(1, "green"); ④

        String oudeKleur = colorList.set(1, "black");
        System.out.printf("Het oorspronkelijke kleur was %s.%n", oudeKleur); ⑤

        String kleur = colorList.get(2);
        System.out.printf("Kleur op index 2 is %s.%n", kleur); ⑥

        colorList.add("black");
        colorList.remove("black"); ⑦

        kleur = colorList.remove(1);
        System.out.printf("De kleur die verwijderd werd was %s.%n", kleur); ⑧
    }
}
```

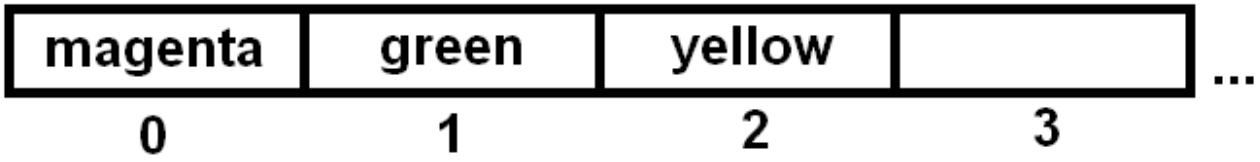
① Declaratie en creatie (instantiëring) van de lijst. We gebruiken hier de diamond operator: bij de creatie hoeft het type niet meer vermeld te worden. Let wel op alle tekens, vergeet ze zeker niet: new ArrayList<>();

② De kleur "magenta" wordt aan de lijst toegevoegd.

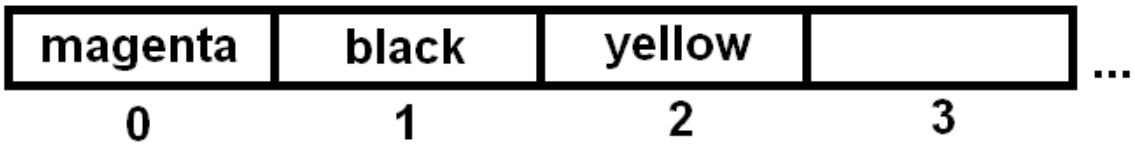


③ De kleur "yellow" wordt achteraan de lijst toegevoegd.

④ Op index 1 wordt kleur "green" toegevoegd.



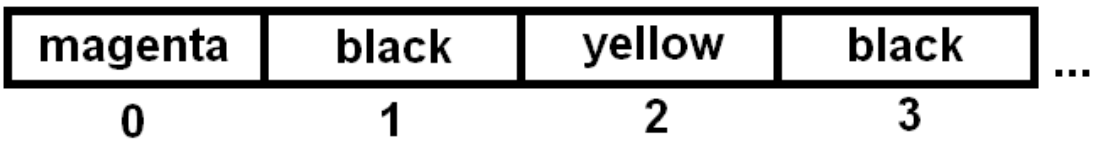
⑤ Op index 1 wordt de kleur gewijzigd naar "black".



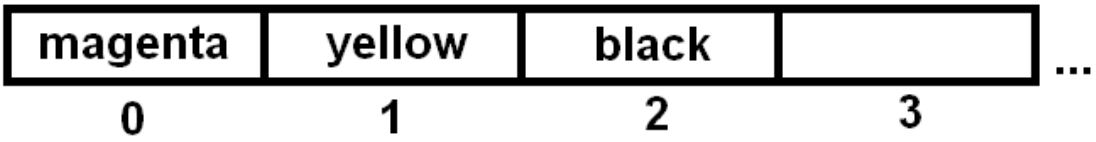
Op het scherm wordt weergegeven: `"Het oorspronkelijke kleur was: green"`

⑥ Op het scherm wordt weergegeven: `"Kleur op index 2 is yellow"`

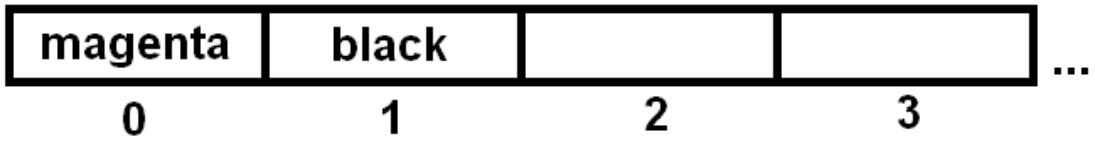
⑦ Na het toevoegen van de kleur "black" (nogmaals) aan de lijst, ziet die er als volgt uit:



Het verwijderen van "black" uit de lijst heeft als resultaat:



⑧ Tenslotte verwijderen we ook nog het element op index 1, met als resultaat:



Op het scherm wordt weergegeven: `"De kleur die verwijderd werd was yellow"`

3.4. Illustratie van de methodes contains, size en indexOf

Het tweede voorbeeld gebruikt een lijst van kommagetallen. Je kan echter geen List<double> definiëren aangezien in de definitie List<E> de elementen van E objecten moeten zijn. We gebruiken dus de wikkelklasse (wrapper class) Double hiervoor.

Het type van de elementen van een ArrayList moet altijd een klasse zijn, geen primitief type!

```
package cui;

import java.util.ArrayList;
import java.util.List;

public class ArrayListDouble
{
    public static void main(String[] args)
    {
        List<Double> decGetallenList = new ArrayList<>();
        decGetallenList.add(5.5);
        decGetallenList.add(7.0);
        decGetallenList.add(4.25);

        if (decGetallenList.contains(7.0)) ①
        {
            System.out.println("Het getal 7 komt voor");
        }
        System.out.printf("aantal getallen in de lijst: %d%n", decGetallenList.
size()); ②

        int index = decGetallenList.indexOf(4.25); ③
        if (index != -1)
        {
            System.out.printf("getal 4.25 komt voor op index %d%n", index);
        }
    }
}
```

① De methode `contains` bepaalt of een waarde al dan niet in de lijst voorkomt (true of false).

② De methode `size()` geeft het aantal elementen in de lijst terug.

③ De methode `indexOf()` bekijkt of de meegegeven parameter voorkomt in de lijst en zo ja, op welke index. De index of de waarde -1 (indien de parameter niet voorkomt) wordt teruggegeven.

Het runnen van deze code geeft volgende uitvoer:

```
Het getal 7 komt voor
aantal getallen in de lijst : 3
getal 4.25 komt voor op index : 2
```

3.5. Enhanced for

Syntax van het "enhanced for" statement:

```
for (parameter : naam_van_de_collectie)
    statement
```

Werking: Bij elke iteratie (elke keer dat je door de lus gaat) gebeurt het volgende:

- uit de collectie wordt het volgende element genomen
- dit element wordt bijgehouden in een variabele
- de body van de lus wordt uitgevoerd op dit element

Voorbeeld:

Hier maken we als collectie gebruik van een lijst van gehele getallen. Ook hier dient weer de wikkelklasse Integer gebruikt te worden: List<Integer> dus.

```
package cui;

import java.util.ArrayList;
import java.util.List;

public class ArrayListInteger
{
    public static void main(String[] args)
    {
        int a = 6, b = 3, c = 8, d = 5;
        List<Integer> integerList = new ArrayList<>();

①
        integerList.add(a);
        integerList.add(b);
        integerList.add(c);
        integerList.add(d);

        for (int element : integerList) ②
        {
            System.out.printf("%d  ", element);
        }

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

① Achtereenvolgens worden de waarden van a, b, c en d toegevoegd aan de lijst. Merk op dat dit variabelen van het primitieve type int zijn, terwijl je objecten nodig hebt om toe te voegen aan de lijst. De omzetting van int naar Integer gebeurt echter automatisch dankzij **autoboxing**.

② De elementen uit de lijst worden doorlopen met een enhanced for en zo één voor één afgebeeld op het scherm.

Het resultaat van dit programma is:

```
6  3  8  5
```

Gebruik **NOOIT** een gewone for-lus om een ArrayList te doorlopen. Dit is namelijk niet performant! In plaats van een enhanced for kan je ook een iterator gebruiken (zie OOSD II).

Dus **niet**:

```
for (int i=0; i < integerList.size(); i++)
   System.out.printf("%d  ", integerList.get(i));
```

We gebruiken de enhanced for alleen om gegevens uit de lijst te lezen. Je kan de gegevens **NIET wijzigen**, dus het inlezen van de gegevens die in de lijst moeten komen en het verwijderen van bepaalde elementen zullen we op een andere manier moeten oplossen!

4. Eendimensionale arrays

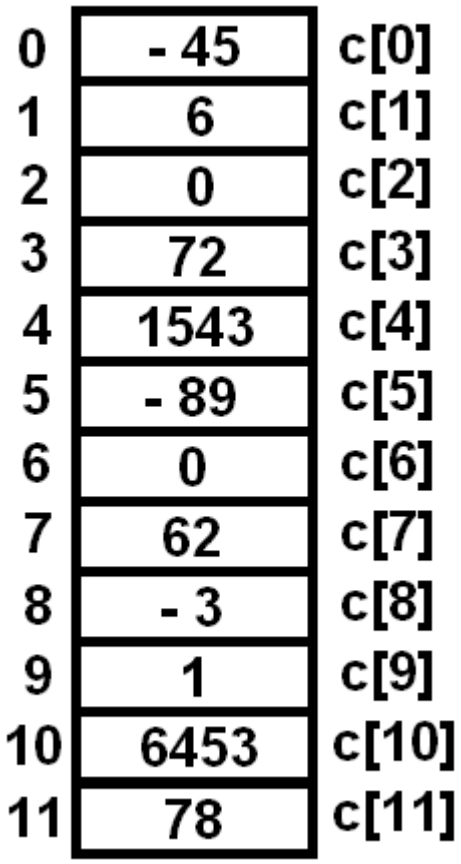
4.1. Definitie

Een array kan net als een ArrayList een reeks gegevens bevatten, maar het is een **statische** entiteit, wat wil zeggen dat je de grootte vooraf moet vastleggen en daarna niet meer kan wijzigen.

De grootte of lengte van de array is dus deze vastgelegde capaciteit.

4.2. Terminologie

De array c zou er bijvoorbeeld als volgt kunnen uitzien in het geheugen:



Er zijn dus 12 elementen, met index van 0 tot en met 11 aanwezig. De lengte van deze array, die je kan opvragen via het **public final attribuut length** is dus 12.

```
System.out.println(c.length);
```

levert met andere woorden de waarde 12 op.

Let op de syntax: je mag niet length() schrijven want dan zou het een methode zijn!

Doordat het attribuut length **public** is kan je de waarde ervan opvragen zonder dat je een getter nodig hebt. Vanwege het feit dat het attribuut ook nog eens **final** is, kan je de lengte van de array niet (per ongeluk) wijzigen.

De naam (referentie) van de array is c. Als je echter informatie nodig hebt over een van de elementen, dan heb je ook de index nodig. Met behulp van vierkante haakjes kan je zo een individuele naam voor elk element vormen.

```
System.out.println(c[7]);
```

geeft zo bijvoorbeeld 62 als output.

De index kan ook berekend worden aan de hand van een expressie die een geheel getal oplevert.

Bijvoorbeeld:

```
int a = 6, b = 2;
System.out.println(--c[a+b]);
System.out.println(c[c.length-1]);
```

geeft als resultaat:

```
-4
78
```

4.3. Prioriteiten

Haakjes (zowel vierkante **[]** als ronde **()**) hebben de hoogste prioriteit:

Voorbeeld:

```
c[1+3] = 10 + 2 * 4 - (2 - 1)
  c[4] = 10 + 2 * 4 - 1
  c[4] = 10 + 8 - 1
  c[4] = 17
```

Algemeen overzicht:

Operators	Associativity	Type
() [] .	left to right	highest
++ --	right to left	unary postfix
++ -- + - ! (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&	left to right	boolean logical AND
^	left to right	boolean logical exclusive OR
|	left to right	boolean logical inclusive OR
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

4.4. Declaratie en creatie van een array

Arrays zijn **objecten** die je kan doorgeven en gebruiken net als andere objecten.

Om een array te gebruiken, dien je een variabele te declareren, een array-object te creëren en de variabele doen verwijzen naar het array-object.

Een array-object creëren doe je met de operator new (cfr creatie van een object):

- Eerst moet je een variabele declareren die naar een array-object kan verwijzen.

```
int c[];
```

- Daarna kan je een array-object creëren en er met de variabele naar verwijzen.

```
c = new int[ 12 ];
```

- OF je kan de twee acties ineens combineren:

```
int c[] = new int[ 12 ];
```

Bij de creatie worden dus 2 dingen vermeld:

- het **type** van de elementen van de array
- de **grootte** van de array, dus het aantal elementen dat erin kan opgenomen worden

In het voorbeeld hierboven is het type **int** en het aantal elementen **12**.

c is een variabele die naar een array-object verwijst. M.a.w. c is een **referentie**.

De declaratie kan je op 2 manieren doen:

```
int[] a, b, c;
OF
int a[], b[], c[];
```

In beide gevallen zijn a, b en c variabelen die naar een array van gehele getallen kunnen verwijzen. Het verschil ligt hem hierin dat je bij de eerste manier duidelijk aangeeft dat het datatype van de variabelen een array van ints is en je bij de tweede manier per variabele aangeeft of het al dan niet om een array gaat die gedeclareerd wordt. De tweede manier volgt de syntax van C, maar de eerste manier is de meest gebruikte en heeft dus eigenlijk de voorkeur. Toch is het uiteindelijk maar een kwestie van stijl en is het belangrijkst om consistent te werken.

Bij de declaratie mag je nog geen lengte opgeven!
Voorbeeld van een foute declaratie: int[10] getallen;

Voorbeeld van de creatie van een array

```
double[] a, b;
a = new double[ 10 ];
b = new double[ 20 ];
OF
double[] a = new double[ 10 ];
double[] b = new double[ 20 ];
OF
double[] a = new double[ 10 ], b = new double[ 20 ];
```

4.5. Initialisatie van de array-elementen

4.5.1. Initialisatie door opsomming

In het volgende voorbeeld wordt een array van gehele getallen geïnitialiseerd door de elementen op te sommen.

```
package cui;

public class InitArray
{
    public static void main(String args[])
    {
        int[] array = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37}; ①

        System.out.printf("%s%8s%n", "Index","Value"); ②

③
        for (int counter = 0; counter < array.length; counter++)
        {
            System.out.printf("%5d%8d%n", counter, array[ counter]);
        }
    }
}
```

① De elementen van de array worden geïnitialiseerd. De items staan tussen accolades {} en worden onderling gescheiden door een komma.

Voorbeeld:

int[] n = { 10, 20, 30, 40, 50 }; Creëert een array, bestaande uit 5 elementen.

index	**0**	**1**	**2**	**3**	**4**
n	10	20	30	40	50

De operator **new** is niet nodig. Je kan dus ook geen lengte opgeven, maar die wordt automatisch berekend uit het aantal opgegeven elementen.

② Dit zijn de headers van de tabel die je als output krijgt. Er werd geen breedte opgegeven voor de eerste kolom, dus die gebruikt het aantal karakters van het woord "Index" (5) als breedte. De tweede kolom heeft een opgegeven breedte van 8 karakters.

③ Hier wordt telkens de waarde van de index en de waarde van het element geprint, eveneens op veldbreedtes van respectievelijk 5 en 8. De waarde van **array.length** is het aantal elementen in de array (dus hier 10), waardoor de lus 10 keer doorgelopen wordt en counter telt van 0 tot en met 9. De waarde van het array-element op index counter vind je dan weer terug door array[counter] op te vragen.

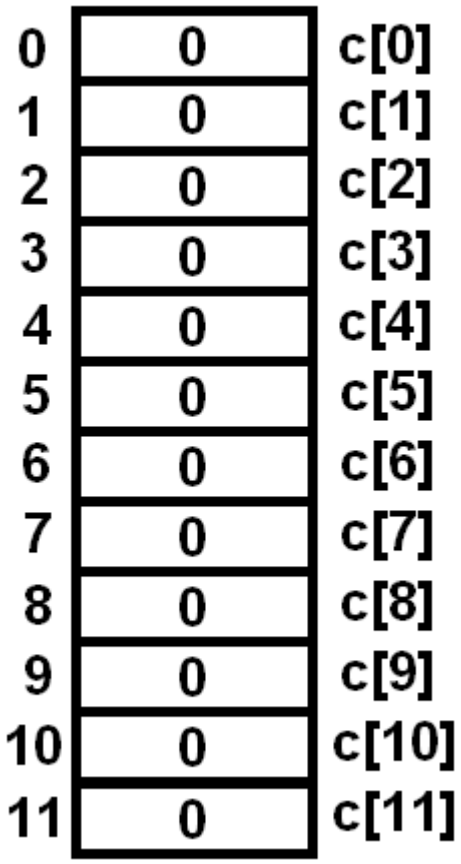
De uitvoer ziet er dus als volgt uit:

Index	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

4.5.2. Initialisatie na creatie

Elk element van de array wordt **automatisch geïnitialiseerd** bij de creatie van de array.

In het voorbeeld waar `int[] c = new int[12];` werd gedeclareerd en gecreëerd, krijgen dus alle elementen van de array de waarde nul aangezien 0 de beginwaarde is voor een **int**.



Als de elementen van array van het type **boolean** zijn, dan worden ze geïnitialiseerd op **false**.

Als de elementen van array **objecten** zijn, dan is hun beginwaarde **null**.
Voorbeeld van een array van objecten: `String b[] = new String[100];`

In het volgende codevoorbeeld wordt de array, die uit 10 elementen bestaat, na creatie opgevuld met de even getallen van 2 tot en met 20.

```
package cui;

public class InitArrayEvenNumbers
{
    public static void main(String[] args)
    {
        final int ARRAY_LENGTH = 10; ①
        int array[] = new int[ARRAY_LENGTH]; ②

③
        for (int counter = 0; counter < array.length; counter++)
        {
            array[ counter] = 2 + 2 * counter;
        }

④
        System.out.printf("%s%8s%n", "Index", "Value");
        for (int counter = 0; counter < array.length; counter++)
        {
            System.out.printf("%5d%8d%n", counter, array[ counter]);
        }
    }
}
```

① De constante ARRAY_LENGTH wordt gedeclareerd. In de rest van de code kan nu gebruik gemaakt worden van deze constante telkens we de grootte van de array nodig hebben.

② De referentie `array` wordt gedeclareerd: ze kan nu verwijzen naar een array-object, dat uit ints zal bestaan.

We laten de referentie `array` verwijzen naar een array-object dat uit **10 ints** bestaat.

Elk element van array wordt automatisch geïnitialiseerd met de waarde 0.

③ Hier wordt de waarde van elk array element bepaald. Hierbij wordt de teller `counter` gebruikt om de elementen van de array in een lus op te vullen.

counter	array[counter]	waarde
0	array[0] = 2 + 2 * 0	2
1	array[1] = 2 + 2 * 1	4
2	array[2] = 2 + 2 * 2	6
n	array[n] = 2 + 2 * n	2+2*n

④ Dit is dezelfde code als in het vorig voorbeeld om de array te laten zien.

Het resultaat bij het runnen van deze code, is:

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

4.6. Andere toepassingen met arrays

4.6.1. Histogram

In het volgende programmavoorbeeld tonen we een histogram. Dit is een staafdiagram die de frequentie van verschillende mogelijke waarden grafisch voorstelt. In dit voorbeeld gaat het om de puntenverdeling bij een examen, die wordt opgedeeld in intervallen.

Bijvoorbeeld als de array als volgt gedefinieerd zou zijn:

```
int array[] = {2, 5, 3};
```

dan wordt er weergegeven:
00-09: **
10-19: *****
   20: ***

In het codevoorbeeld ziet de array er zo uit:

```
int array[] = {8, 0, 6, 7, 11, 9, 13, 5, 17, 2, 1}
```

dus de uitvoer is dan:

```
Grade distribution:
00-09: ********
10-19:
20-29: ******
30-39: *******
40-49: ***********
50-59: *********
60-69: ************
70-79: *****
80-89: ****************
90-99: **
  100: *
```

Hoe we daarbij komen, leggen we uit aan de hand van volgende code:

```
package cui;

public class BarChart
{
    public static void main(String[] args)
    {
        int[] array = {8, 0, 6, 7, 11, 9, 13, 5, 17, 2, 1};

        System.out.println("Grade distribution:");

①
        for (int counter = 0; counter < array.length; counter++)
        {
②
            if (counter == array.length - 1) ③
            {
                System.out.printf("%5d: ", counter * 10); ④
            }
            else ⑤
            {
                System.out.printf("%02d-%02d: ", counter * 10, counter * 10 + 9); ⑥
            }

⑦
            for (int stars = 1; stars <= array[counter]; stars++)
            {
                System.out.print("*");
            }

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

① We doorlopen de array in een lus. De variabele `counter` loopt van 0 tot de arraylengte (in het voorbeeld dus 11).

② Eerst gaan we de legende opbouwen. We willen bij elke "staaf" (horizontale lijn met sterretjes) uitleg schrijven over het interval waarin de scores zich bevinden. In dit geval gaat het om scores per tiental (0-9, 10-19, 20-29, ...) behalve de laatste "groep" die enkel uit de perfecte score (hier: 100) bestaat.

③ Het eerste geval doet zich voor wanneer `counter` gelijk is aan de index van het laatste element van de array `array`. Dit gebeurt wanneer `counter array.length - 1` is.

④ In dat geval komt in de legende het getal `counter*10`. We schrijven dit getal op veldbreedte 5 zodat een ander interval (bestaande uit 2 getallen van 2 cijfers, gescheiden door een liggend streepje) even breed is.

⑤ In alle andere gevallen (dus als het niet om de laatste "groep" scores gaat), behoort de score tot

een interval dat in de legende zal getoond worden.

⑥ Het eerste interval is bijvoorbeeld [0,9] en het tweede [10,19]. Algemeen heeft dit interval dus een ondergrens gelijk aan `counter*10` en een bovengrens `counter*10+9`. Voor het eerste interval moeten we bij de onder- en bovengrens een extra 0 tonen zodat de legende even breed is als in de andere gevallen. Dit bekomen we door %02d te gebruiken: een veldbreedte van 2 karakters, waarbij de "lege plaatsen" worden ingevuld met extra nullen.

⑦ In deze lus wordt het benodigde aantal sterretjes getoond, zodat er een "staaf" gevormd wordt van een lengte die overeenkomt met de waarde van het element van de array. Bijvoorbeeld als `counter=4` dan is `array[counter]=11` en is de output dus * (11 sterretjes).

4.6.2. Enquête

In het volgende voorbeeld werd aan 20 studenten een enquêtevraag gesteld. Ze geven een getal op een schaal tussen 1 en 5 waarmee ze de kwaliteit van het eten op school beoordelen. Hierbij betekent de waarde 1 "afschuwelijk" en de waarde "5" excellent. Alle gehele waardes tussen 1 en 5 zijn ook mogelijk als input en duiden een gradatie tussen deze twee extremen aan.

Gegeven volgende code:

```
package cui;

public class StudentPoll
{
    public static void main(String args[])
    {
        int[] responses = {1, 2, 5, 4, 3, 5, 2, 1, 3, 3, 1, 4, 3, 3, 3, 2, 3, 3, 2,
14}; ①
        int[] frequency = new int[6]; ②

③
        for (int answer = 0; answer < responses.length; answer++)
        {
            ++frequency[responses[answer]];
        }

④
        System.out.printf("%s%10s%n", "Rating", "Frequency");

⑤
        for (int rating = 1; rating < frequency.length; rating++)
        {
            System.out.printf("%6d%10d%n", rating, frequency[rating]);
        }
    }
}
```

① De array `responses` bevat de antwoorden van de studenten op de enquêtevraag. Hierbij dient opgemerkt te worden dat deze normaliter wellicht 1 voor 1 zouden worden ingelezen. Ook

vermelden we nog dat het aantal elementen in de array voor de rest van het programma van geen belang is, zodat de enquête ook aan meer of minder personen kan worden afgenomen.

② De tweede array, `frequency`, stelt een frequentietabel voor. Dit betekent dat elk element een frequentie (= aantal voorkomens) voorstelt van een bepaalde waarde (of groep van waarden). Hier willen we tellen hoeveel keer elk van de mogelijke scores 1 tot en met 5 voorkomt. Hiervoor hebben we dus 5 array-elementen nodig. We kiezen echter voor een array met een extra element omdat we dan de score kunnen gebruiken als index voor de frequentietabel. Index 0 wordt in dit geval niet gebruikt. Elk element wordt automatisch geïnitialiseerd op 0.

③ We overlopen alle antwoorden uit de array `responses` in een lus. De variabele `answer` telt van 0 tot het aantal elementen in de array `responses` en dient als index van deze array. Zo halen we telkens een element uit de array op (in het voorbeeld: achtereenvolgens 1, 2, 5, 4, ...) en dit gebruiken we opnieuw als index voor de array `frequency`. Het element dat we op deze index vinden, verhogen we telkens met 1.

De eerste keer dat we deze lus doorlopen, zullen alle elementen in de array `frequency` dus nog steeds op 0 staan, behalve het element op index 1 (omdat `responses[0]` de waarde 1 bevat), dat met 1 verhoogd werd en dus de waarde 1 bevat. Na de tweede doorloop staat ook het element op index 2 op 1, na de derde doorloop ook het element op index 5, enzoverder.

Merk op dat we met de oorspronkelijke code hierdoor een foutmelding krijgen bij het laatste array-element uit `responses`, aangezien dit als waarde **14** heeft, wat geen geldige index is voor de array `frequency`. Het programma probeert namelijk om 1 toe te voegen aan `frequency[14]`, maar deze index gaat buiten de grenzen van de array. Java laat dit niet toe.

De JVM kijkt elke array index na: deze moet groter of gelijk aan 0 zijn en kleiner dan de lengte van de array — dit noemen we **bounds checking**.

Als de index ongeldig blijkt, dan genereert Java een zogenaamde exception om aan te geven dat er iets fout ging in het programma bij het uitvoeren.

Exception handling wordt uitvoerig behandeld in OOSD II.

In OOSD I gaan we nog geen fouten opvangen (catch), maar kan je ze wel al gooien (throw).

Vaak kan je ook vermijden dat er een exception optreedt: bewaak telkens goed de grenzen (= bounds) van de index van een array! Zo voorkom je heel wat fouten.

Vb. in bovenstaande code:

```
for (int answer = 0; answer < responses.length; answer++)
    if (responses[answer] >= 0 && responses[answer] < frequency
.length)
        ++frequency[responses[answer]];
```

④ Hier worden opnieuw de headers van de tabel in de uitvoer getoond. Voor "Rating" is er geen veldbreedte voorzien, dus deze kolom krijgt de breedte van het woord "Rating", zijnde 6

karakters. Voor "Frequency" is de veldbreedte 10 karakters.

⑤ In deze lus wordt per mogelijke score uit de array `frequency` de waarde getoond die in het array-element opgeslagen werd, zijnde het aantal keer dat deze score voorkomt. Merk op dat de score 0 en het bijhorende array-element op index 0 overgeslagen wordt aangezien deze score niet kan voorkomen.

Zo krijgen we zonder wijziging in de code deze output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 14 out of bounds for length 6
        at cui.StudentPoll.main(StudentPoll.java:13)
```

Het laatste element van de array `responses` zorgt voor de exception en heeft als resultaat dat niet alleen de lus die deze score behandelt, afgebroken wordt, maar ook dat de rest van de code niet meer uitgevoerd wordt.

Wanneer we het laatste element uit de array responses wijzigen naar 4 ipv 14, dan wordt de uitvoer:

```
Rating Frequency
     1         3
     2         4
     3         8
     4         3
     5         2
```

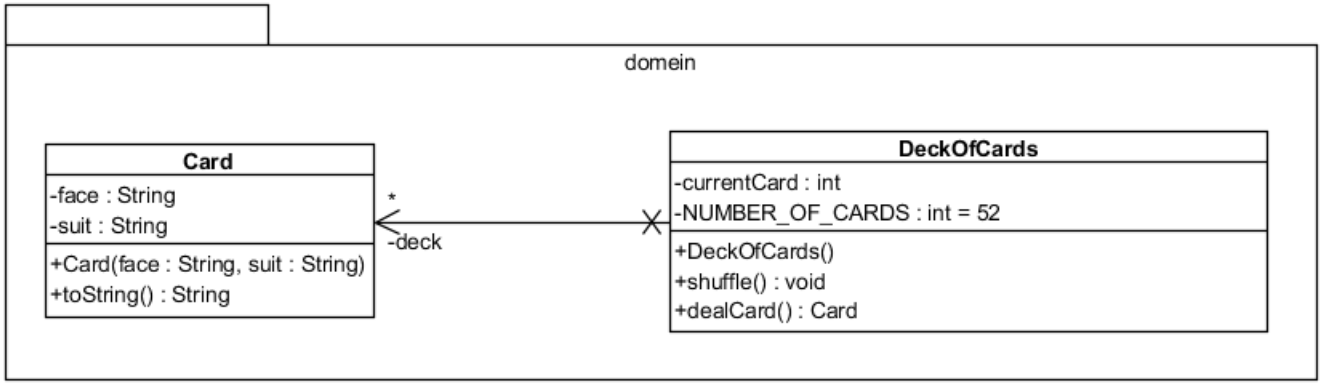
4.7. Uitgebreid voorbeeld: speelkaarten door elkaar schudden en uitdelen

In deze case study gaan we 52 speelkaarten bijhouden in een array. Met andere woorden: de array bevat referenties naar objecten van de domeinklasse `Card`.

Uit de analyse halen we volgende 2 domeinklassen:

- Card stelt één speelkaart voor. Een kaart heeft een “waarde” (Eng: face) en een “kleur” (Eng: suit). Vb. Klaverenvier heeft als waarde vier en als kleur klaveren en hartenheer heeft als waarde heer en als kleur harten.
- DeckOfCards stelt een volledig kaartspel voor, bestaande uit 52 speelkaarten. Deze klasse zal als attribuut een array van speelkaarten (Card-objecten) bevatten.

Het DCD van dit voorbeeld ziet er dus als volgt uit:



De domeinklasse Card bevat volgende code:

```
package domein;

public class Card
{
①
    private final String face;
    private final String suit;

②
    public Card(String face, String suit)
    {
        this.face = face;
        this.suit = suit;
    }

③
    @Override
    public String toString()
    {
        return face + " of " + suit;
    }
}
```

① 2 final attributen: de waarde en kleur van de kaart zijn na het aanmaken niet meer wijzigbaar!

② In de constructor worden de face en suit ingesteld zonder verdere controles aangezien we geen kaarten zullen construeren die niet bestaan.

③ De methode toString beschrijft de tekstweergave van een Card:

- Voorbeeld in het Engels: "queen of spades" - je gebruikt dus de formule `face+" of "+suit`
- In het Nederlands: “schoppenvrouw” - je gebruikt dus de formule `suit+""+face`

In de domeinklasse DeckOfCards vinden we volgende code terug:

```
package domein;
```

```
public class DeckOfCards
{
    private Card[] deck; ①
    private int currentCard; ②
    private final int NUMBER_OF_CARDS = 52; ③

    public DeckOfCards()
    {
④
        String[] faces = {"Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
                          "Eight", "Nine", "Ten", "Jack", "Queen", "King"};
        String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};

        deck = new Card[NUMBER_OF_CARDS]; ⑤

⑥
        for (int count = 0; count < deck.length; count++)
        {
            deck[count] = new Card(faces[count % 13], suits[count / 13]);
        }
    }

⑦
    public void shuffle()
    {
        currentCard = 0; ⑧

⑨
        for (int first = 0; first < deck.length; first++)
        {
⑩
            int second = (int) (Math.random() * NUMBER_OF_CARDS);
⑪
            Card temp = deck[first];
            deck[first] = deck[second];
            deck[second] = temp;
        }
    }

⑫
    public Card dealCard()
    {
⑬
        if (currentCard < deck.length)
        {
            return deck[currentCard++]; ⑭
        }
        else
        {
            return null; ⑮
        }
```

```
    }
}
```

① Er wordt een variabele genaamd `deck` bijgehouden, die verwijst naar een array bestaande uit Card-objecten

② De variabele `currentCard` houdt de index (waarde uit het interval [0,51]) bij van de volgende uit te delen kaart

③ De constante `NUMBER_OF_CARDS` duidt aan uit hoeveel kaarten het `deck` straks zal bestaan

④ In de constructor worden eerst 2 arrays van Strings gedefinieerd. De array `faces` bevat alle mogelijkheden (13 stuks) voor de waarde van een kaart. De array `suits` bevat de 4 mogelijkheden voor de kleur.

⑤ Het `deck` wordt gecreëerd en bestaat uit zoveel array-elementen van het type Card als de constante `NUMBER_OF_CARDS` aangeeft.

⑥ Alle elementen van de array deck krijgen een waarde. Hiervoor wordt telkens een object van de klasse Card gecreëerd dat toegekend wordt aan het array-element op index `count`:

als count = 0: deck[0] = new Card("Ace", "Hearts");
als count = 1: deck[1] = new Card("Deuce", "Hearts");
als count = 2: deck[2] = new Card(“Three", "Hearts");
…
als count = 12: deck[12] = new Card("King", "Hearts");
als count = 13: deck[13] = new Card("Ace", "Diamonds");
als count = 14: deck[14] = new Card("Deuce", "Diamonds");
…

⑦ De methode shuffle gebruikt een algoritme dat met één doorloop van de array in een lus de kaarten door elkaar schudt.

⑧ Eerst wordt de variabele currentCard weer op 0 gezet, zodat na het schudden van de kaarten opnieuw met de eerste kaart van de stapel (`deck[0]`) begonnen wordt.

⑨ We doorlopen alle kaarten van de stapel en gebruiken als index voor de eerste te wisselen kaart alle mogelijke indexen van 0 tot de lengte van de array (52, niet inbegrepen).

⑩ Als tweede index gebruiken we een random getal uit het interval [0,52[. Daarvoor gebruiken we de methode random van de klasse Math en vermenigvuldigen die met het aantal mogelijkheden. Dit levert een kommagetal op dat we nog moeten casten naar een geheel getal.

⑪ Tenslotte wisselen we de twee Card-objecten op index `first` en `second`.
Bijvoorbeeld: indien `deck[first]` verwijst naar het Card-object ("Four","Clubs") en `deck[second]` verwijst naar Card-object ("Queen","Spades"), dan zal Card-object ("Queen","Spades") toegekend worden aan `deck[first]` en Card-object ("Four","Clubs") aan `deck[second]`.
Indien we twee waarden willen wisselen, dan hebben we steeds een hulpvariabele nodig. Hier: `temp`. Daarin bewaren we tijdelijk de waarde van de eerste variabele, die we dan kunnen overschrijven met de waarde van de tweede variabele. In de tweede variabele kennen we tenslotte de waarde van de hulpvariabele (`temp`) toe.

```
deck[first] = deck[second];
deck[second] = deck[first];
```
lukt dus NIET!!!

⑫ De methode dealCard geeft de Card terug die zal uitgedeeld worden.

⑬ Eerst wordt bepaald of er nog een volgende kaart is om uit te delen.

⑭ Zo ja, dan wordt de kaart op index currentCard teruggegeven en schuift de index currentCard 1 positie op

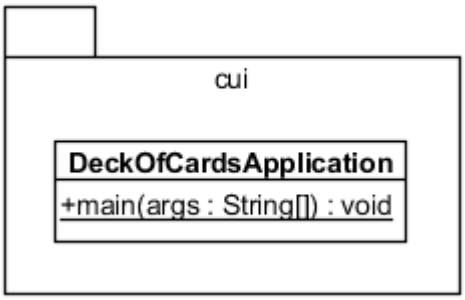
```
return deck[currentCard++];
```

is hetzelfde als

```
Card card = deck[currentCard];
currentCard++;
return card;
```

⑮ Als er geen kaart meer is, wordt "null" geretourneerd als teken dat alle kaarten uitgedeeld zijn

Het klassediagram van de cui-klasse bevat alleen de main-methode:



De bijhorende code is deze:

```
package cui;

import domein.Card;
import domein.DeckOfCards;

public class DeckOfCardsApplication
{
    public static void main(String args[])
    {
        DeckOfCards myDeckOfCards = new DeckOfCards(); ①
        myDeckOfCards.shuffle(); ②
        int teller = 1; ③
        Card kaart = myDeckOfCards.dealCard(); ④

        do
        {
            System.out.printf("%-19s", kaart); ⑤
            kaart = myDeckOfCards.dealCard(); ⑥

⑦
            if (teller % 4 == 0)
            {
                System.out.println();
            }

            teller++; ⑧
        }
        while (kaart != null); ⑨
    }
}
```

① We maken eerst een object van de klasse DeckOfCards om hiermee de non-static methodes `shuffle` en `dealCard` aan te kunnen roepen.

② Schud eerste de kaarten door elkaar.

③ Stel de kaartenteller in op 1.

④ Vraag de huidige (= eerste) kaart op om uit te delen.

⑤ Print de huidige kaart op een veldbreedte van 19 karakters, links uitgelijnd.

⑥ Vraag opnieuw de huidige (= volgende) kaart op.

⑦ Als de teller deelbaar is door 4, dan zijn er 4 kaarten naast elkaar getoond en moet de volgende kaart dus op de volgende regel komen. Daarom gebruiken we een println statement.

⑧ Verhoog de kaartenteller.

⑨ Herhaal de stappen vanaf het printen van de huidige kaart zolang er nog kaarten zijn. Als er geen kaarten meer zijn zal de huidige kaart null bevatten.

Een mogelijke uitvoer ziet er dan als volgt uit (4 kolommen met telkens 13 willekeurige kaarten, elk van de 52 kaarten komt precies 1 keer voor):

```
Nine of Hearts     Three of Diamonds  Six of Hearts      Four of Clubs
Jack of Diamonds   Eight of Hearts    Queen of Clubs     Eight of Diamonds
Three of Clubs     Five of Diamonds   Jack of Clubs      Ace of Hearts
Ten of Diamonds    King of Clubs      Four of Spades     Three of Hearts
King of Hearts     Eight of Clubs     Ten of Hearts      Eight of Spades
Seven of Spades    Deuce of Spades    King of Diamonds   Jack of Spades
Seven of Hearts    Nine of Clubs      Deuce of Hearts    King of Spades
Six of Clubs       Six of Diamonds    Jack of Hearts     Ten of Clubs
Nine of Spades     Queen of Spades    Ace of Spades      Deuce of Clubs
Three of Spades    Queen of Diamonds  Five of Spades     Ten of Spades
Five of Clubs      Ace of Clubs       Seven of Clubs     Six of Spades
Four of Hearts     Five of Hearts     Deuce of Diamonds  Queen of Hearts
Nine of Diamonds   Ace of Diamonds    Four of Diamonds   Seven of Diamonds
```

4.8. Enhanced for voor arrays

Ook voor arrays is het steeds aangewezen zo veel mogelijk gebruik te maken van een enhanced for. Als we de elementen willen doorlopen, is een enhanced for altijd efficiënter. Voor het wijzigen van de array-elementen dienen we echter een gewone for te gebruiken.

Bijvoorbeeld:
Stel `int[] array = {2,5,7};`

Om de elementen uit deze array te laten zien, gebruiken we beter geen traditionele for-lus:

```
for (int i=0; i<array.length; i++)
    System.out.printf("%4d", array[i]);
```

maar wel een enhanced for:

```
for (int getal : array)
    System.out.printf("%4d", getal);
```

Het resultaat is immers hetzelfde, maar de enhanced for is veel efficiënter.

Ander voorbeeld:
Gegeven:

```
int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
int total = 0;
```

Om nu de som van alle array-elementen te maken en het resultaat van deze optelling te stockeren in de variabele total, zullen we geen traditionele for gebruiken:

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

maar wel een enhanced for:

```
for (int number : array)
    total += number;
```

Voordeel enhanced for: Doordat je niet meer met een index werkt, kan de fout `ArrayIndexOutOfBoundsException` niet meer voorkomen.

5. Tweedimensionale arrays

5.1. Definitie

Een tweedimensionale array

- is net zoals een ArrayList en een eendimensionale array een datastructuur waarin je een **zelfde soort gegevens** kunt opslaan, zoals waarden van een bepaald primitief type of referenties naar instanties van een bepaalde klasse
- bestaat uit twee dimensies: rijen en kolommen
- is eigenlijk een eendimensionale array waarvan elk element zelf ook weer een eendimensionale array is
- bevat elementen die elk twee indexen hebben
  - de beide indexen beginnen op nul
  - bij conventie is de eerste index de rijindex en de tweede dus de kolomindex

Schematische voorstelling van een 3x4-array

	kolom 0	**kolom 1**	**kolom 2**	**kolom 3**
rij 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
rij 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
rij 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Het element **a[2][1]** verwijst naar een element uit de array met **naam a**, **rijindex 2** en **kolomindex 1**.

5.2. Declaratie en creatie van een tweedimensionale array

5.2.1. Initialisatie door opsomming

Voorbeeld:

int[][] b = { { 1, 2 }, { 3, 4 } };

	kolom 0	kolom 1
rij 0	1	2
rij 1	3	4

b[0][0] bevat de waarde 1, b[0][1] bevat de waarde 2
b[1][0] bevat de waarde 3, b[1][1] bevat de waarde 4

De **naam** van de array is **b**.

De array b heeft **4 elementen**.

Er zijn **2 rijen**. Dit merk je aan het aantal { }-paren binnenin de buitenste accolades.

Per rij zijn er telkens **2 kolommen** (dit hoeft niet gelijk te zijn voor elke rij). Dit weten we omdat er tussen de binnenste { }-paren telkens 2 getallen staan.

Met behulp van de naam van de array en de twee indexen krijgt elk element zijn eigen naam, in ons voorbeeld b[0][0], b[0][1], b[1][0] en b[1][1].

Ander voorbeeld:

int[][] b = { { 1, 2 }, { 3, 4, 5 } };

	kolom 0	kolom 1	kolom 2
rij 0	1	2	
rij 1	3	4	5

b[0][0] bevat de waarde 1, b[0][1] bevat de waarde 2, **b[0][2] bestaat niet**
b[1][0] bevat de waarde 3, b[1][1] bevat de waarde 4, b[1][2] bevat de waarde 5

5.2.2. Initialisatie na creatie

Voorbeeld van de declaratie en creatie van een 3 x 4 array:

- Eerst declareren we een referentie die naar een tweedimensionaal array-object kan verwijzen
  `int[][] b;`
- Daarna creëren we een tweedimensionaal array-object en er met de referentie naar verwijzen
  `b = new int[ 3 ][ 4 ];`
- OF we kunnen de twee acties combineren:
  `int[][] b = new int[ 3 ][ 4 ];`

Elk element wordt automatisch geïnitialiseerd!
Dit lukt alleen als het aantal kolommen per rij voor elke rij hetzelfde is.

b is een variabele dat naar een 2 dim. array-object verwijst. M.a.w. b is een **referentie**.

	kolom 0	kolom 1	kolom 2	kolom 3
rij 0	0	0	0	0
rij 1	0	0	0	0
rij 2	0	0	0	0

Voorbeeld waar het aantal kolommen per rij verschillend is:

```
int[][] b;
b = new int[2][];   // 2 rijen
b[0] = new int[5];  // rij 0 heeft 5 kolommen
b[1] = new int[3];  // rij 1 heeft 3 kolommen
```

	kolom 0	kolom 1	kolom 2	kolom 3	kolom 4
rij 0	0	0	0	0	0
rij 1	0	0	0		

elementen b[1][3] en b[1][4] bestaan niet

5.3. Programmavoorbeelden

In de volgende programmavoorbeelden zullen we (een keer met een gewone for en een keer met een enhanced for):

- een tweedimensionale array declareren en initialiseren
- de elementen van de array weergeven op het scherm

Voorbeeld 1 (gewone for):

```
package cui;

public class InitArray2DimVb1
{
    public static void main(String args[])
    {
        int[][] array = {{1, 2, 3},{4, 5}}; ①

        System.out.println("Values in array by row are\n");
        String output = "";

②
        for (int row = 0; row < array.length; row++)
        {
③
            for (int column = 0; column < array[row].length; column++)
                 output += String.format("%d    ", array[row][column]);

            output += "%n";
        }
        System.out.printf(output);
    }
}
```

① Creatie en initialisatie van een lokale variabele die een referentie bevat naar een array.

② Doorlopen van de eerste dimensie (rijen) van de array. Het aantal rijen wordt aangeduid met `array.length`.

③ Doorlopen van de tweede dimensie (kolommen) van de array. Het aantal kolommen per rij wordt bepaald via `array[row].length` waarbij `row` de rij-index voorstelt.

Voorbeeld 2 (enhanced for):

```
package cui;

public class InitArray2DimVb2
{
    public static void main(String args[])
    {
        int[][] array = {{1, 2, 3},{4, 5}};

        System.out.println("Values in array by row are\n");
        String output = "";

①
        for (int[] row: array)
        {
②
            for (int element: row)
                 output += String.format("%d    ", element);

            output += "%n";
        }
        System.out.printf(output);
    }
}
```

① Eerste lus: deel de tweedimensionale array op in rijen. Elke `row` stelt 1 zo'n rij voor.

② Tweede lus: deel de rij op in losse gehele getallen. Elk `element` is een int.

De uitvoer van beide voorbeelden is hetzelfde, namelijk:

```
Values in array by row are

1    2    3
4    5
```