HO
GENT

H04 - Array en ArrayList

Table of Contents

1. Doelstellingen

Na het bestuderen van dit hoofdstuk ben je in staat

- een **eendimensionale array** te **declareren** en te **initialiseren** in Java
- met een **eendimensionale array** te **werken** in Java
- een **enhanced for-statement** te **gebruiken** in Java
- een **tweedimensionale array** te **declareren** en te **initialiseren** in Java
- met een **tweedimensionale array** te **werken** in Java
- het **verschil** tussen een ArrayList en een array te **kennen**
- een **ArrayList** te **declareren** en te **initialiseren** in Java
- met een **ArrayList** te **werken** in Java

2. Inleiding

We zijn reeds vertrouwd met Java variabelen die we gebruiken om een waarde, een gegeven, bij te houden. In hoofdstuk 1 maakten we kennis met de **primitieve** datatypes en variabelen waarin we een waarde van dergelijk type kunnen opslaan. Na hoofdstuk 3 weten we nu ook dat een variabele kan gebruikt worden om een referentie naar een **object** bij te houden.

Bij het ontwikkelen van software krijgen we echter dikwijls te maken met groepen, verzamelingen, van gegevens. Denk maar aan een klas die bestaat uit een *groep studenten*, of een webwinkel die een *verzameling aan producten* aanbiedt en die een *aantal klanten* heeft die *bestellingen* plaatsen…

In dit hoofdstuk maken we kennis met datastructuren waarin **meer dan één gegeven** kan bewaard worden. Alle gegevens binnen de verzamelingen die we behandelen zijn **homogeen**: ze zijn allemaal van **hetzelfde type**. Zo gaan bijvoorbeeld alle elementen in een verzameling van hetzelfde primitieve type zijn, of zullen alle elementen referenties zijn naar objecten van éénzelfde klasse.

3. Arrays

> Een array is een referentie naar een *container*, die een verzameling van **gelijk getypeerde gegevens** bevat. De container heeft een **vaste grootte**. De afzonderlijke gegevens in de array noemen we **elementen**. Elk element kan benaderd worden via zijn positie in de container, deze positie noemen we de **index** van het element.

3.1. Declaratie & instantiatie

Om een array te declareren zetten we ná het **datatype** van de elementen rechte haakjes: **[]**

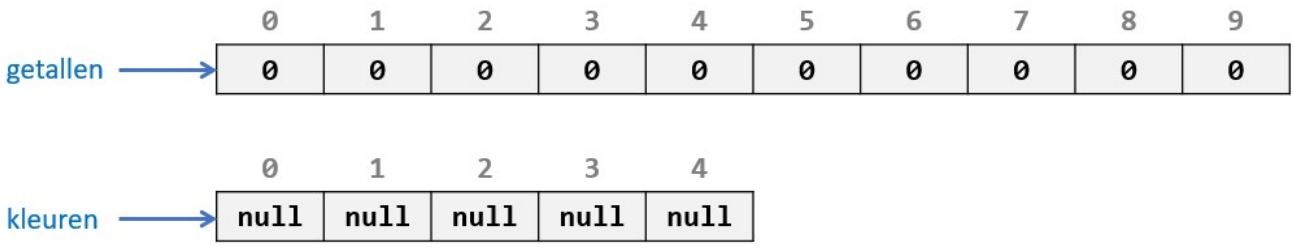
```
    int getal;          // de variabele getal kunnen we gebruiken om 1 integer bij te
houden
```

```
    int[] getallen;     // de variabele getallen kunnen we gebruiken om een container
te maken waarin we meerdere integers kunnen bijhouden
```

Een array is een verwijzing naar een object, een container, die de verzameling aan gegevens zal bevatten. In bovenstaand voorbeeld is de variabele `getallen` dus een **referentie** variabele. Het object die de verzameling met gegevens zal bevatten moeten we instantiëren. We kunnen hiervoor gebruik maken van de `new` operator. Bij deze instantiatie moet je de **grootte** van de verzameling vastleggen. Hiervoor gebruiken we weer rechte haakjes, waartussen we nu het **aantal elementen, die we in de verzameling willen,** specificeren. Merk op dat dit aantal elementen na instantiatie niet meer gewijzigd zal kunnen worden!

```
    int[] getallen;
    getallen = new int[10]; // via de variabele getallen hebben we toegang tot een
verzameling van 10 int variabelen

    String[] kleuren = new String[5]; // via de variabele kleuren hebben we toegang
tot een verzameling van 5 String variabelen
```

Na instantiatie bevat de container direct het gevraagde aantal elementen, i.e. variabelen. Elke variabele werd hierbij geïnitialiseerd op zijn **default waarde**. Uit vorig hoofdstuk herinneren we nog dat de default waarde voor numerieke gegevens `0` is, voor boolse gegevens is die `false` en voor referentie types is de default waarde `null`. Bovenstaande arrays `getallen` en `kleuren` kunnen we als volgt visualiseren.



De elementen in de container zijn **geordend**. Zoals we op bovenstaande afbeelding zien heeft elk element een volgnummer, een positie. Dit is de **index** en deze start steeds vanaf **0**.

"Zero Based Indexing" is de term die gebruikt wordt om aan te geven dat het **eerste element** in een geordende verzameling **index 0** heeft.

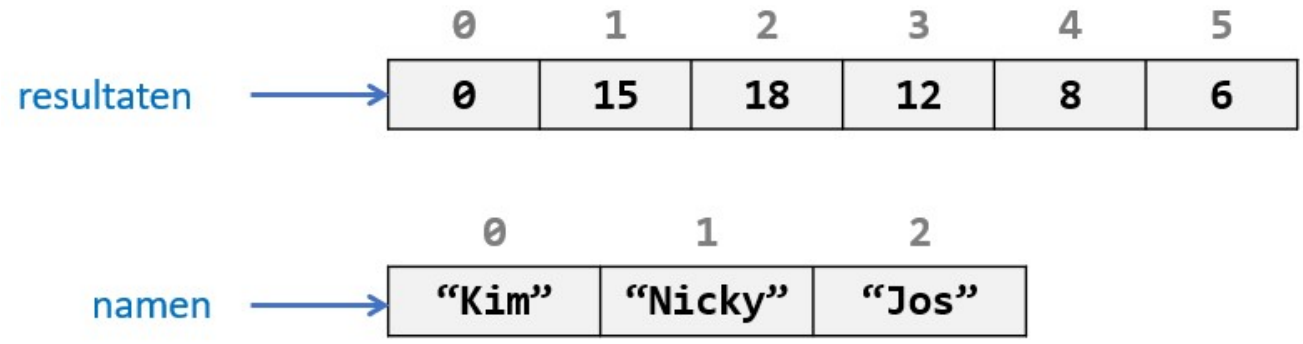
Indien op het moment van instantiatie reeds gekend is welke elementen in de verzameling moeten komen, kunnen we dit ook aangeven in Java. Zo hebben we de mogelijkheid om niet steeds te moeten vertrekken van een array die gevuld is met variabelen die automatisch werden geïnitialiseerd op hun default waarde. Bij dergelijke instantiatie volstaat het om **expliciet de gegevens** die in de verzameling moeten komen, op te sommen tussen **accolades**. De grootte van de array kan worden afgeleid uit het aantal elementen die tussen de accolades staat en moeten we dan ook niet opgeven.

```
int[] resultaten = new int[] {0, 15, 18, 12, 8, 6};
String[] namen = new String[] {"Kim", "Nicky", "Jos"};
```

Bij de expliciete **declaratie én initialisatie** van een array variabele, zoals hierboven, is de `new` operator redundant en *mogen* we deze weglaten.

```
int[] resultaten = {0, 15, 18, 12, 8, 6};
String[] namen = {"Kim", "Nicky", "Jos"};
```

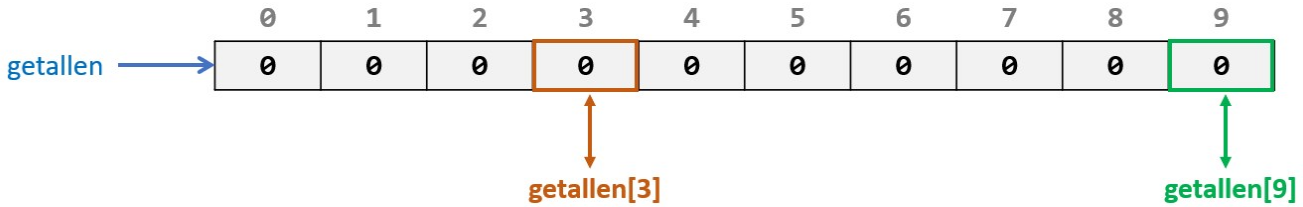
De resulterende arrays:



3.2. Werken met arrays

3.2.1. Index en meer rechte haakjes...

Om één specifiek element uit een array vast te pakken, dit is dus één specifieke variabele uit de verzameling variabelen die in de container zit, maken we gebruik van de **naam van de array variabele**, gevolgd door **rechte haakjes** waartussen we de **index** van het gewenste element plaatsen.



In volgend voorbeeld drukken we de waarde van het vierde element in de array `getallen` af in de console:

```
System.out.printf("Het vierde getal in de array is %d.", getallen[3]);

// Uitvoer:
// Het vierde getal in de array is 0.
```

De index die we tussen de rechte haakjes plaatsen mag ook een expressie zijn met een **int** resultaat.

Analoog aan vorig voorbeeld kunnen we ook schrijven:

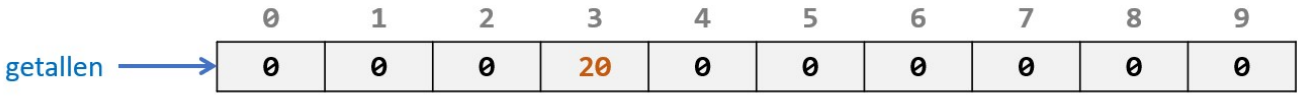
```
    int index = 3;
    System.out.printf("Het %d-de element in de array is %d.", index + 1, getallen
[index]);  ①

    // Uitvoer:
    // Het 4-de element in de array is 0.
```

① het vierde, `index + 1`, element in de array zit op index 3, `getallen[index]`

Merk op dat we dikwijls een offset van 1 moeten gebruiken indien we willen communiceren over de elementen in een array. In natuurlijke taal praten we over het **eerste**, het **tweede**, ... element in een geordende verzameling, terwijl dit in de array elementen zijn met index **0**, index **1**, ...

De elementen van een array zijn volwaardige variabelen. Alles wat we reeds leerden in vorige hoofdstukken in verband met variabelen kunnen we nu dus ook toepassen op elementen van een array. Om het vierde getal in de array de waarde 20 te geven gebruiken we volgend assignment statement:

```
    getallen[3] = 20;
```

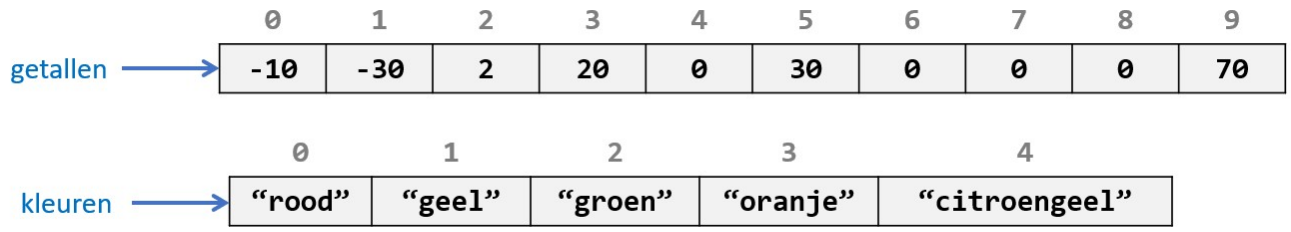


Meer voorbeeldjes:

```
    int[] getallen = new int[10];
    String[] kleuren = new String[5];

    getallen[0] = -10;
    getallen[3] = 20;
    getallen[9] = getallen[3] + 50;

    int i = 5;
    getallen[i] = 30;
    getallen[i % 2] = -30;
    getallen[i / 2] = i / 2;

    kleuren[0] = "rood";
    kleuren[1] = "geel";
    kleuren[2] = "groen";
    kleuren[3] = "oranje";
```

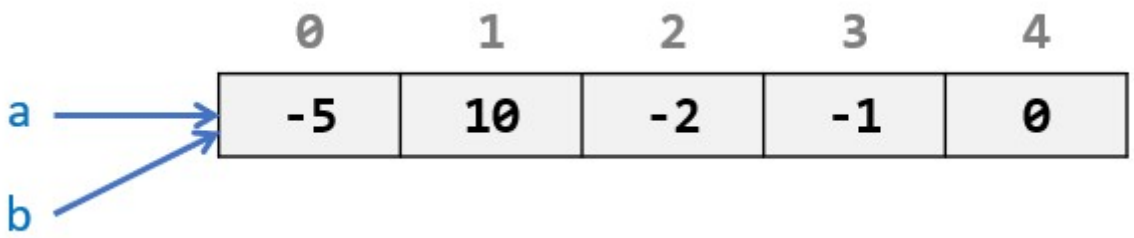
```
kleuren[4] = "citroen" + kleuren[1];
```

Bovenstaande statements leiden tot volgende arrays:

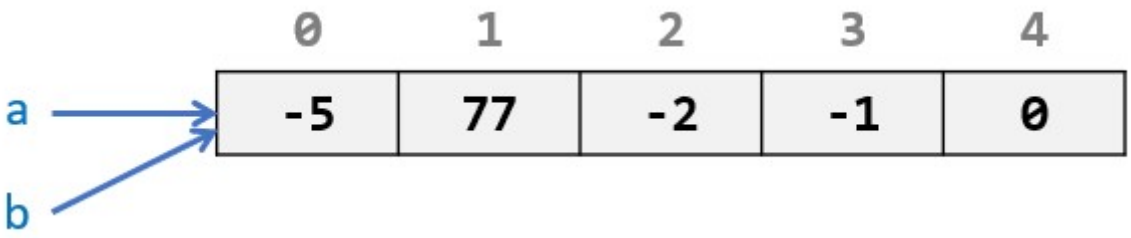


We mogen niet uit het oog verliezen dat de array variabele zelf werkelijk een referentie bevat naar de verzameling en dat het dus ook mogelijk is om met die referentie te werken! Onderstaand voorbeeld zoomt hier op in.

```
int[] a = {-5, 10, -2, -1, 0};
int[] b = a;
```

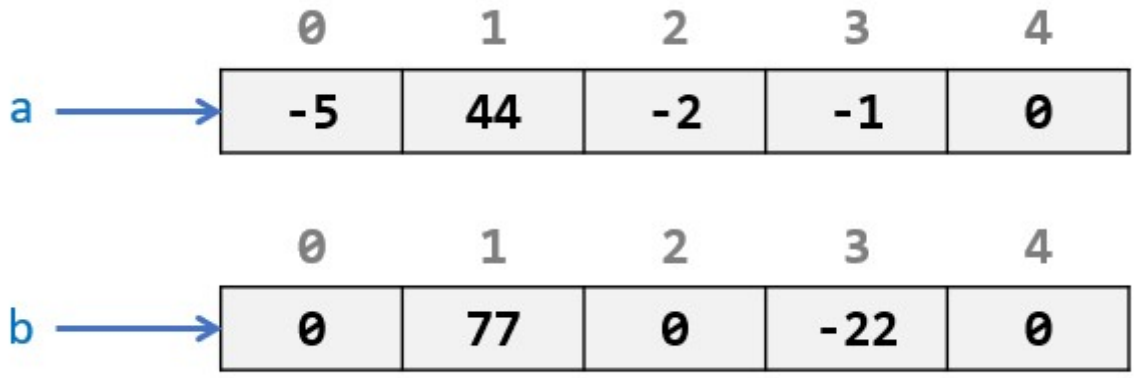



```
b[1] = 88;
System.out.printf("In array a zit op index 1 de waarde %d", a[1]);
System.out.printf("In array b zit op index 1 de waarde %d", b[1]);
a[1] = 77;
System.out.printf("In array a zit op index 1 de waarde %d", a[1]);
System.out.printf("In array b zit op index 1 de waarde %d", b[1]);

// Uitvoer:
// In array a zit op index 1 de waarde 88
// In array b zit op index 1 de waarde 88
// In array a zit op index 1 de waarde 77
// In array b zit op index 1 de waarde 77
```



Verder bouwend hierop:

```
b = new int[5];
b[1] = a[1];
a[1] = 44;
b[3] = -22;
System.out.printf("In array a zit op index 1 de waarde %d", a[1]);
System.out.printf("In array b zit op index 1 de waarde %d", b[1]);

// Uitvoer:
// In array a zit op index 1 de waarde 44
// In array b zit op index 1 de waarde 77
```



Wanneer we met de elementen van een array willen werken maken we constant gebruik van indices. Als programmeur heb je de verantwoordelijkheid om er voor te zorgen dat een gebruikte index **geldig** is. Concreet betekent dit dat de kleinst mogelijke index die we kunnen gebruiken `0` is, en de grootst mogelijke index steeds gelijk is aan het `aantal elementen in de array - 1`. Het aantal elementen dat een array bevat kan je steeds achterhalen via de `length` property.

```
int[] getallen = new int[10];
String[] kleuren = {"rood", "geel", "groen", "rood", "geel"};

System.out.printf("De array getallen bevat %d elementen", getallen.length);
System.out.printf("De array kleuren bevat %d elementen", kleuren.length);

// Uitvoer:
// De array getallen bevat 10 elementen
// De array kleuren bevat 5 elementen
```

Indien in een programma gebruik wordt gemaakt van een ongeldige index dan wordt er een `ArrayIndexOutOfBounds` exception geworpen. Dit moeten we uiteraard steeds proberen te vermijden!

```
String[] kleuren = {"rood", "geel", "groen", "roze", "paars"};
int aantalKleuren = kleuren.length;
kleuren[aantalKleuren] = "wit"; ①
```

① Het programma wordt hier onderbroken: *Exception in thread "main"*

java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5

- Om het **aantal elementen in een array** te bepalen gebruik je de ingebouwde **property** `length`
  - dit is een property, je maakt geen gebruik van ronde haakjes!
  - **geldige indexen voor een array a** liggen in het interval **[0, a.length - 1]**
- Om het **aantal karakters in een String** te bepalen gebruik je de **methode** `length()`.
  - dit is een methode aanroep, je moet gebruik maken van ronde haakjes!

Door gebruik te maken van de `length` property kunnen we er voor zorgen dat we met onze index niet buiten de grenzen van de array te gaan.

Onderstaand voorbeeld illustreert hoe we alle elementen van een array mooi kunnen afdrukken, ongeacht de lengte van de array. In dit voorbeeld wordt de array `getallen` op een willekeurige manier een array van **10**, of een array van **3** `int`-s. In beide gevallen gaat de for-lus de array correct overlopen.

```
    private void toonGetallen() {
        SecureRandom sr = new SecureRandom();
        int[] getallen;
        if (sr.nextBoolean())
            getallen = new int[] { -10, -30, 2, 20, 0, 30, 0, 0, 0, 70 };
        else
            getallen = new int[] { 12, 8, 20 };
        System.out.printf("De array bevat %d elementen:%n", getallen.length);
        for (int index = 0; index < getallen.length; index++) {
            System.out.printf("- op index %d zit element %d.%n", index, getallen[
index]);
        }
    }
```

Als we deze methode een aantal keer aanroepen krijgen we soms dit te zien:

```
 De array bevat 10 elementen:
 - op index 0 zit element -10.
 - op index 1 zit element -30.
 - op index 2 zit element 2.
 - op index 3 zit element 20.
 - op index 4 zit element 0.
 - op index 5 zit element 30.
 - op index 6 zit element 0.
 - op index 7 zit element 0.
```

```
    - op index 8 zit element 0.
    - op index 9 zit element 70.
```

Soms dit:

```
    De array bevat 3 elementen:
    - op index 0 zit element 12.
    - op index 1 zit element 8.
    - op index 2 zit element 20.
```

3.2.2. De klassieke for-lus en de enhanced for-lus

Verzamelingen en lus-structuren gaan hand in hand. Daar we werken met meerdere gegevens zullen we ook dikwijls over meerdere gegevens willen itereren. In bovenstaande methode `toonGetallen` werd gebruik gemaakt van een typische, klassieke for-lus om alle elementen uit de array te overlopen. In veel gevallen zullen we bij de iteratie echter geen nood hebben aan de index van elk element, maar zal enkel het element zelf van belang zijn in de body van de lus. Java voorziet daarom ook in een handige lus-structuur waarbij de typische 'administratie' (i.e. teller initialiseren, teller updaten, teller testen) die we anders steeds nodig hebben om een verzameling te overlopen wegvalt. We noemen dit een **enhanced for-loop**. Bij dergelijk lus hoef je enkel een **variabele** te declareren en te specificeren welke array je wil overlopen. De lus wordt automatisch voor elk element uitgevoerd en daarbij heb je in de body van de lus, via de gedeclareerde variabele, steeds toegang tot een **copy** van het 'huidige' element.

Volgende voorbeeldjes waarbij we **som van alle elementen** van een array berekenen verduidelijken dit. In het eerste fragment wordt gebruik gemaakt van een klassieke for lus, in het tweede fragment werd dezelfde functionaliteit uitgewerkt aan de hand van een enhanced for lus:

```
        double[] getallen = { 23.12, 2.56, 88, 97.4 };
        double som = 0;
        for (int index = 0; index < getallen.length; index++) { ①
            som += getallen[index];
        }
        System.out.printf("De som is %.3f", som); // De som is 211,080
```

① De **klassieke for-lus:** merk op hoe we een teller (genaamd `index`) initialiseren op 0, bij elke iteratie met 1 verhogen en bij elke iteratie testen of die kleiner is dan de lengte van de array `getallen`.

```
        double[] getallen = { 23.12, 2.56, 88, 97.4 };
        double som = 0;
        for (double getal : getallen) { ①
            som += getal;
        }
        System.out.printf("De som is %.3f", som); // De som is 211,080
```

① De **enhanced for-lus:** we declareren een variabele `getal` en geven aan, na de `:`, dat we willen itereren over de array `getallen`. De lus wordt voor elk element van de array uitgevoerd. In de body van de lus bevat de variabele `getal` telkens een **copy** van een element van de array `getallen`. Bij de eerste iteratie is dat een copy van het element op index 0, bij de tweede iteratie een copy van het element op index 1, enzoverder…

- De **enhanced for-lus** laat ons toe om op een **eenvoudige, simpele** manier over een array te itereren.
  - waar mogelijk zullen we steeds gebruik maken van deze lus, in OOSD houden wij immers van het **KISS**-principe: **Keep It Short & Simple**
- De **klassieke for-lus** geeft je meer controle
  - in de body van de enhanced for-lus heb je **geen toegang tot de index** van het 'huidig' element, indien je die index nodig hebt in de body zal je toch liever een klassieke for-lus willen gebruiken
    - voorbeeld: we willen enkel met elementen op even indices iets doen, we willen enkel elementen tussen twee gegeven indices verwerken, we hebben de index van een element nodig om de verwerking te doen, …
  - in de body van de enhanced for-lus kan je de **elementen van de array waarover je itereert niet wijzigen**, in de body beschik je in de lus variabele enkel over een **copy** van een element uit de array, niet het element zelf

Volgende voorbeeldjes illustreren dat we via de lus variabele in een enhanced for-lus geen wijzigingen kunnen aanbrengen aan de elementen in de array waarover we itereren. De bedoeling van de code snippets is om in een array van gehele getallen, `getallenmix`, elk negatief getal in 0 te veranderen…

We proberen dit eerst met een **klassieke for-lus**

```
        int[] getallenmix = { -5, 10, -2, -1, 0, 23 };
        for (int index = 0; index < getallenmix.length; index++) {
            if (getallenmix[index] < 0)
                getallenmix[index] = 0; ①
        }

②
        // Resultaat, inhoud van de array getallenmix:
        // - op index 0 zit element 0.
        // - op index 1 zit element 10.
        // - op index 2 zit element 0.
        // - op index 3 zit element 0.
        // - op index 4 zit element 0.
        // - op index 5 zit element 23.
```

① we kennen effectief aan het element op plaats `index` in de array `getallenmix` de waarde `0` toe

② het resultaat is een gewijzigde array...

En nu proberen we hetzelfde doen aan de hand van een **enhanced for-lus**

```
        int[] getallenmix = new int[] { -5, 10, -2, -1, 0, 23 };
        for (int getal : getallenmix) {
            if (getal < 0)
                getal = 0; ①
        }

②
        // Resultaat, inhoud van de array getallenmix:
        // - op index 0 zit element -5.
        // - op index 1 zit element 10.
        // - op index 2 zit element -2.
        // - op index 3 zit element -1.
        // - op index 4 zit element 0.
        // - op index 5 zit element 23.
```

① we kennen aan `getal`, *dit is een **copy** van een element uit de array* `getallenmix`, `0` toe; niet aan het element zelf!

② de array is ongewijzigd!

3.3. Enkele uitgewerkte voorbeelden

3.3.1. Voorbeeld - Histogram

In dit voorbeeld gaan we de verdeling van examenresultaten grafisch weergeven aan de hand van een staafdiagram. We vertrekken van een array die gehele waarden bevat. Elke waarde in die array stelt het *aantal studenten* voor die een resultaat behaalde in een bepaald interval. Op index 0 vinden we het aantal studenten die een resultaat in het interval [0, 9] behaalden, op index 1 vinden we het aantal studenten die een resultaat in het interval [10, 19] behaalden, en dit gaat zo verder tot we uiteindelijk op de laatste index het aantal studenten vinden die het maximum behaalden (de laatste index is een uitzondering daar het niet over een interval maar over 1 enkel resultaat, het maximum, gaat).

Voorbeeld voor array met verdeling:

```
int[] verdeling = {5, 8, 2};
```

Gewenste uitvoer:

```
    Verdeling examenresultaten:
    00-09: *****
    10-19: ********
```

```
      20: **
```

Voorbeeld voor array met verdeling:

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

Gewenste uitvoer:

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

Het programma:

```
public class Staafdiagram {
    public static void main(String[] args) {
        new Staafdiagram().toonStaafdiagram();
    }

    private void toonStaafdiagram() {
        int[] verdeling = { 8, 0, 6, 7, 11, 9, 13, 5, 17, 2, 1 };

        System.out.println("Verdeling examenresultaten:");

        for (int index = 0; index < verdeling.length; index++) { ①
            if (index == verdeling.length - 1) ②
                System.out.printf("%5d: ", index * 10);
            else
                System.out.printf("%02d-%02d: ", index * 10, index * 10 + 9);

            tekenStaafje(verdeling[index]); ③

            System.out.println();
        }
    }

    private void tekenStaafje(int lengte) {
        for (int sterretje = 1; sterretje <= lengte; sterretje++) {
```

```
            System.out.print("*");
        }
    }
}
```

① voor elk element in de array moeten we een lijn van het staafdiagram genereren

② de laatste index (die we kennen via de length property) krijgt een speciale behandeling want deze stelt geen interval voor

③ aan de methode tekenStaafje geven we het aantal sterretjes die moeten afgedrukt worden door: dit is het *aantal studenten* en zit vervat in het element die op de huidige index zit

3.3.2. Voorbeeld - Enquête

In dit voorbeeld gaan we de resultaten van een tevredenheids enquête verwerken. Voor deze enquête gaven een aantal scholieren een score van **1 ("slecht") tot 5 ("super")** op de kwaliteit van de schoolmaaltijden. De scores zitten in een array en de bedoeling van dit programma is dat we alle scores gaan samenvatten.

Voorbeeld scores:

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

Voorbeeld uitvoer:

```
    Score    Aantal
        1         2
        2         3
        3         3
        4         4
        5         7
    Totaal aantal antwoorden: 19
```

Het programma:

```
public class Enquete {
    public static void main(String args[]) {
        new Enquete().verwerkEnquete();
    }

    private void verwerkEnquete() {
        int[] scores = { 5, 2, 5, 4, 3, 5, 2, 1, 5, 5, 1, 4, 3, 3, 4, 5, 5, 4, 2 };
        int[] resultaten = new int[5]; ①

        for (int score : scores) { ②
            resultaten[score - 1]++;
        }
```

```
        System.out.printf("%8s%8s%n", "Score", "Aantal");
        for (int resultaat = 0; resultaat < resultaten.length; resultaat++) {
            System.out.printf("%8d%8d%n", resultaat + 1, resultaten[resultaat]); ③
        }

        System.out.printf("Totaal aantal antwoorden: %d%n", scores.length); ④
    }

}
```

① er zijn 5 mogelijke scores (1 - 5), voor elke mogelijke score zullen we het aantal keer dat de score werd gegeven bijhouden, vandaar dat we een array voor 5 `int`-s instantiëren. Merk op dat de indices in deze array lopen van 0 t.e.m. 4! Alle elementen in deze array staan automatisch op de default waarde 0.

② voor elke gegeven score wordt het juiste element in de array met 1 verhoogd, op index 0 vinden we dus het aantal keer dat score 1 werd gegeven, op index 1 het aantal keer dat score 2 werd gegeven enzoverder…

③ hier moeten we er weer rekening mee houden dat er een verschil van 1 zit tussen de index en de gegeven score

④ het totaal aantal antwoorden is gelijk aan het aantal elementen in de array scores

4. ArrayList

Hoewel een array een heel nuttige datastructuur is waarmee het eenvoudig werken is (square brackets is all you need…) heeft deze structuur ook 1 groot nadeel: de grootte van een array moet bij instantiatie vastgelegd worden en kan nadien niet meer gewijzigd worden.

> Een `ArrayList` bevat net zoals een array een geordende verzameling elementen. Elk element in de verzameling zit ook gepositioneerd op een zero-based **index**. De **grootte van de verzameling is dynamisch**! Het aantal elementen in de verzameling zal at run-time kunnen wijzigen.

Het feit dat dergelijke verzamelingen een dynamische grootte hebben brengt een enorme flexibiliteit met zich mee. De meeste verzamelingen die wij kennen uit de reële wereld hebben immers een dynamische grootte. Het aantal producten dat een webshop aanbiedt gaat variëren doorheen de tijd, het aantal klanten ook, het aantal bestellingen ook…

4.1. Declaratie & instantiatie

Om gebruik te kunnen maken van deze datastructuur in een programma moeten we expliciet het type `ArrayList` **importeren**. `ArrayList` is immers een onderdeel is van het **Collections framework** van Java. In het algemeen is een software framework als het ware een bibliotheek waarin bepaalde klassen, datatypes, reeds uitgewerkt zijn. Wij kunnen deze uitgewerkte klassen gebruiken door die te ontlenen bij het framework. In Java termen zeggen we dat we dergelijke klassen **importeren** om

ze te kunnen gebruiken.

Arrays maken geen onderdeel uit van het Collections framework, ze zitten standaard in de Java taal ingebakken.

Bovenaan het bestand met onze code zetten we dan ook volgend `import` statement:

```
    import java.util.ArrayList;
```

Bij de declaratie zullen we, net zoals bij arrays, moeten aangeven wat het type is van de elementen die in de verzameling zullen zitten. Dit gebeurt via de *diamond notation*: het type zetten we tussen *kleiner dan* en *groter dan* tekens: `<type>`.

Hier zie je de declaratie van de variabele `benodigdheden`. We zullen de variabele kunnen gebruiken om een verzameling van `String`-s bij te houden.

```
    ArrayList<String> benodigdheden; // benodigdheden is een referentie naar een
container die meerdere Strings zal kunnen bevatten
```

Na de declaratie volgt de instantiatie. Via instantiatie gaan we de container aanmaken. In tegenstelling tot het instantiëren van een array hoeven we hierbij nu **geen grootte** te specificeren. Het resultaat zal een lege container zijn, een verzameling die geen elementen bevat. Hoewel je de diamond notatie ook bij instantiatie moet gebruiken hoef je het datatype niet te herhalen.

De declaratie en instantiatie van onze `benodigdheden`:

```
    ArrayList<String> benodigdheden;
    benodigdheden = new ArrayList<>();
```

Zoals steeds kan de declaratie en initialisatie ook in 1 stap gebeuren:

```
    ArrayList<String> benodigdheden = new ArrayList<>();
```

Hoewel er op zich niets verkeerd is met deze manier van declareren en instantiëren gaan we hier toch nog een klein beetje aan sleutelen.

Zoals de naam suggereeert biedt het Collections framework onderdak aan een uiteenlopende waaier van collections: datastructuren die toelaten om allerlei soorten verzamelingen bij te houden en te verwerken. `ArrayList` is een voorbeeld van een `List`-datastructuur. Het Collections framework bevat echter ook nog andere `List`-structuren en om een grote mate van flexibiliteit in te bouwen in onze software gaan we bij de declaratie liever gewoon aangeven dat we een `List` willen gebruiken.

Bij declaratie gaan we ons nog niet vastpinnen op een specifieke `List`-structuur

Bij instantiatie gaan we wel expliciet aangeven dat het soort `List` die we effectief willen gebruiken

een `ArrayList` is.

Bij instantiatie gaan we specifiek kiezen voor een `ArrayList`

De vele voordelen van dergelijke manier van werken zullen later in het OLOD OOSD II worden toegelicht maar we kunnen nu alvast gewoon worden aan deze techniek.

Onze verzameling `benodigdheden` gaan we dus als volgt declareren en instantiëren:

```
    List<String> benodigdheden;             // declaratie: gebruik List
    benodigdheden = new ArrayList<>();      // instantiatie: gebruik ArrayList
```

Of in 1 stap:

```
    List<String> benodigdheden = new ArrayList<>();
```

Merk op dat we nu zowel `List` als `ArrayList` moeten importeren...

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

public class BasisArrayList {
    public static void main(String[] args) {
        List<String> benodigdheden = new ArrayList<>();

}
```

4.2. Werken met ArrayLists

4.2.1. Elementen toevoegen - `add`

Om een element toe te voegen aan de een `ArrayList` maken we gebruik van de `add` methode

```
    benodigdheden.add("hamer");
```

Onze verzameling `benodigdheden` bevat nu het element *hamer*! Daar onze *hamer* het eerste, en voorlopige enige, element is in de lijst is, vinden we het terug op **index 0**.

Laten we nog een paar benodigdheden toevoegen:

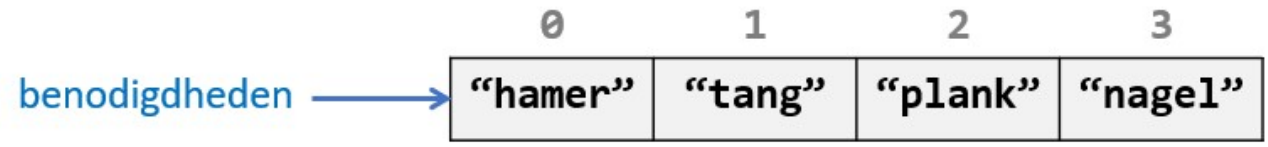
```
benodigdheden.add("plank");
benodigdheden.add("nagel");
```

Onze verzameling bevat nu 3 elementen... Merk op dat de `add` methode een element steeds **achteraan** de lijst toevoegt:

We kunnen ook gebruik maken van een overload van de `add` methode waarbij we de **index** voor het nieuwe element specificeren. Het element die op de index voor het nieuwe element zit, samen met alle elementen die daarop volgen, schuiven dan een plaatsje op naar rechts...

Laten we op index 1 *tang* toevoegen en kijken hoe de verzameling er dan uit ziet:

```
benodigdheden.add(1, "tang");
```



We moeten voorzichtig omspringen met deze overload van de `add` methode want de index moet binnen de grenzen `[0, aantal elementen in lijst]` liggen willen we een `IndexOutOfBoundsException` vermijden!

4.2.2. Aantal elementen - `size` & `isEmpty`

De `size` methode retourneert het aantal elementen in de `ArrayList`. Voor een lege lijst zal deze methode dus 0 retourneren.

```
System.out.printf("We hebben een %d zaken nodig...", benodigdheden.size());

// Uitvoer:
// We hebben een 4 zaken nodig...
```

Om te testen of een lijst leeg is maken we gebruik van de `isEmpty` methode.

```
if (benodigdheden.isEmpty())
    System.out.println("We hebben niets nodig...");
else
```

```
        System.out.printf("We hebben een %d zaken nodig...", benodigdheden.size());
```

4.2.3. Elementen ophalen - `get`

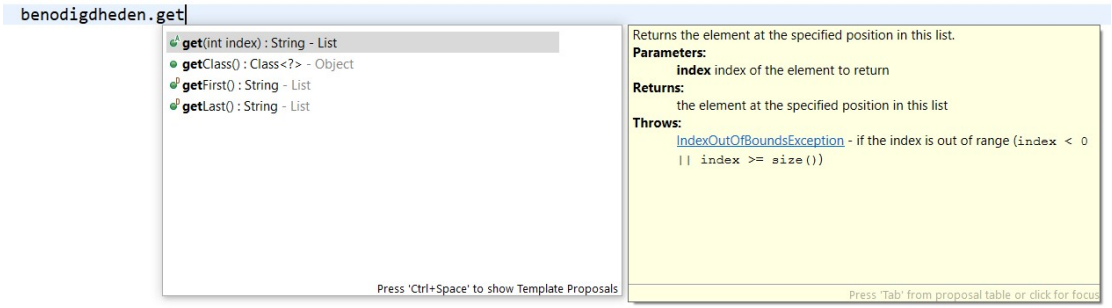
Om een element op te halen uit een `ArrayList` moet je de **index** van het gewenste element doorgeven aan de `get` methode.

```
    String benodigdheid = benodigdheden.get(2); ①
    System.out.println("We hebben een " + benodigdheid + " nodig.");

    // Uitvoer:
    // We hebben een plank nodig.
```

① de variabele `benodigdheid` wordt geïnitialiseerd met de waarde van het element die in de verzameling `benodigdheden` op index 2 zit

Het is belangrijk dat we een **geldige index** doorgeven aan de `get` methode. Bij een ongeldige index zal een `IndexOutOfBoundsException` geworpen worden.

Raadpleeg de **Java API** documentatie om exact te weten hoe je een methode moet aanroepen en wat je exact van een methode mag verwachten!



4.2.4. Elementen wijzigen - `set`

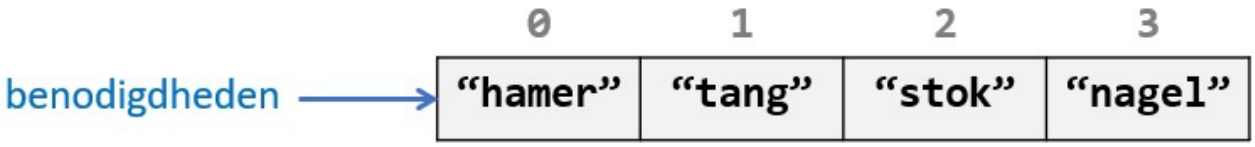
Om een element in de lijst te vervangen kunnen we de `set` methode gebruiken. We moeten de **index** van het element die we wensen te vervangen doorgeven, samen met de nieuwe waarde voor het element.

```
    benodigdheden.set(2, "balk");
```

Als we de Java API documentatie raadplegen zien we dat de `set` methode ook een returnwaarde heeft, namelijk het element die werd vervangen. Alhoewel we de `set` methode dikwijls in

bovenstaande vorm zullen gebruiken, *we zijn meestal enkel geïnteresseerd in het effect van de methode*, kan je eventueel gebruik maken van de returnwaarde.

```
    String oud = benodigdheden.set(2, "stok");
    System.out.printf("We hebben niet langer een %s nodig. We gaan een %s gebruiken.",
oud, benodigdheden.get(2));

    // Uitvoer:
    // We hebben niet langer een balk nodig. We gaan een stok gebruiken.
```



4.2.5. Elementen verwijderen - `remove`

Een laatste cruciale methode, `remove`, laat toe om een element uit een lijst te verwijderen. Op deze manier zal een `ArrayList` dus krimpen.

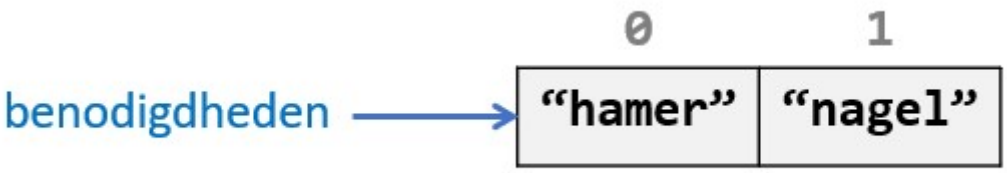
Bij de `remove` methode hebben we de mogelijkheid om de **index** op te geven van het element die we wensen te verwijderen. Zoals je in de Java API kan terugvinden retourneert deze vorm van de remove methode het element dat werd verwijderd uit de lijst.

```
    String verwijderd = benodigdheden.remove(2);
    System.out.printf("De lijst van benodighdheden bevat niet langer %s...",
verwijderd);

    // Uitvoer:
    // De lijst van benodighdheden bevat niet langer stok...
```

Een andere mogelijkheid, via een overload van de remove methode, is om aan te geven welk element we uit de lijst wensen te verwijderen. Deze overload van de `remove` methode retourneert een boolean die aangeeft of het opgegeven element gevonden en verwijderd werd.

```
    if (benodigdheden.remove("tank"))
        System.out.printf("We hebben een tank verwijderd.");
    if (benodigdheden.remove("tang"))
        System.out.printf("We hebben een tang verwijderd.");

    // Uitvoer:
```

```
    // We hebben een tang verwijderd.
```



4.3. Voorbeeld temperaturen

4.3.1. Introductie

In dit uitgewerkt voorbeeld bekijken we een applicatie die toelaat dat een gebruiker voor een aantal opeenvolgende dagen een temperatuur (uitgedrukt als een kommagetal) kan ingeven. De gebruiker signaleert het einde van de invoer met de ingave van de sentinel waarde `-100`. Nadien worden enkele statistieken in verband met de ingegeven temperaturen getoond. Hierbij zullen we zullen kennis maken met nog enkele handige `List` methodes die we hierboven nog niet behandelden.

Vooraleer we starten moeten we even stilstaan bij het **datatype van de elementen** in een `List`. Dit is dus het type die we in de *diamond* zetten.

Het datatype van de elementen in een `List` **moet een referentie type** zijn. Het is niet mogelijk om elementen van primitieve datatypes in een `List` te stoppen.

4.3.2. Wrapper klassen

In Java heeft **elk primitief type** een overeenkomstige **wrapper class**. Een wrapper class laat toe om **objecten** te maken die een **primitief type 'bevatten'**.

Hoewel het dus **niet mogelijk** is volgende declaratie te schrijven in Java:

```
    List<double> temperaturen; // Dit is niet geldig!!!
```

kunnen we wel gebruik maken van de wrapper klasse voor `double`; deze klasse noemt `Double`:

```
    List<Double> temperaturen; // Dit is OK
```

Hieronder zie je de wrapper klassen voor elk primitieve type.

Primitief type	Wrapper class
byte	Byte

Primitief type	Wrapper class
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Op deze manier kunnen we nu dus eender welke `List` maken. Voor een `List` waarin we elementen van een primitief type zouden wensen bij te houden mogen we gewoon niet vergeten de naam van de wrapper klasse te gebruiken in de *diamond*, en niet de naam van het primitief type. Veel meer hoeven we voorlopig niet te weten over de wrapper klassen want de overstap van de waarde van een primitief type naar een object van zijn wrapper class, en omgekeerd, zal in Java automatisch gebeuren.

We noemen dit mechanisme

- **autoboxing** - *automatische conversie van een primitief type naar een object van zijn corresponderende wrapper class*
- **unboxing** - *automatische conversie van een object van een wrapper class type naar zijn corresponderende primitief type*

4.3.3. Uitwerking

We leggen de focus terug op onze temperatuur applicatie. Laat ons beginnen met de gewenste uitvoer van het programma te bekijken.

Voorbeeld uitvoer

```
Geef temperatuur in voor dag 1 > 13,5
Geef temperatuur in voor dag 2 > 12,6
Geef temperatuur in voor dag 3 > 11,5
Geef temperatuur in voor dag 4 > 11,8
Geef temperatuur in voor dag 5 > 12
Geef temperatuur in voor dag 6 > 12,3
Geef temperatuur in voor dag 7 > 12,3
Geef temperatuur in voor dag 8 > -100 ①

Overzicht temperaturen per dag voor 7 dagen ②
Dag 1: 13,5 graden
Dag 2: 12,6 graden
Dag 3: 11,5 graden
Dag 4: 11,8 graden
```

```
Dag 5: 12,0 graden
Dag 6: 12,3 graden
Dag 7: 12,3 graden

Er werden geen vriestemperaturen opgemeten. ③
Dag 1 was de warmste dag met 13,5 graden. ④
Werd er een temperatuur van 20,0 graden opgemeten? nee ⑤
```

① de sentinel waarde is -100

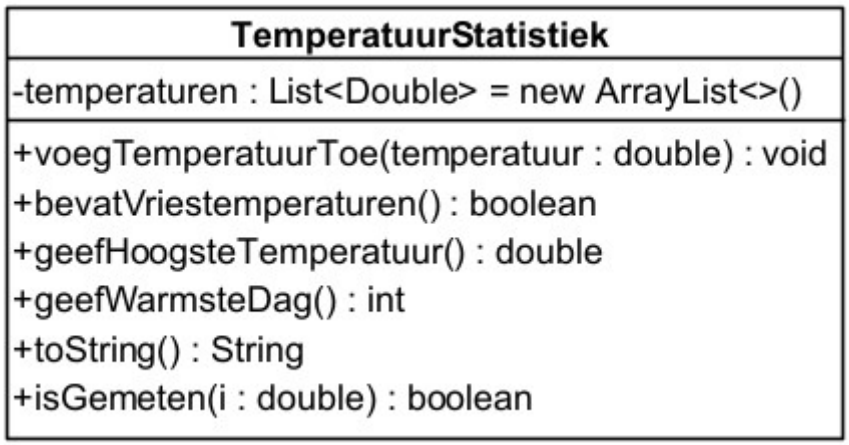
② een overzicht per dag wordt afgedrukt

③ er wordt vermeld of er al dan niet vriestemperaturen werden opgemeten (dit zijn temperaturen kleiner dan of gelijk aan 0)

④ er wordt vermeld welke dag de warmste dag was en welke temperatuur toen werd bereikt

⑤ er wordt vermeld indien in de metingen een temperatuur met waarde 20 voorkomt

De domeinklasse TemperatuurStatistiek



Declaratie van de verzameling `temperaturen`

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

public class TemperatuurStatistiek {

    private List<Double> temperaturen = new ArrayList<>(); ①
```

① de klasse TemperatuurStatistiek heeft 1 field `temperaturen`; dit field bevat de verzameling temperaturen en wordt automatisch geïnitialiseerd op een lege lijst

Een temperatuur toevoegen aan `temperaturen`

```
    public void voegTemperatuurToe(double temperatuur) {
        temperaturen.add(temperatuur); ①
    }
```

① merk op dat we ons niets hoeven aan te trekken van het feit dat de lijst **objecten van de wrapper klasse** `Double` bevat, Java zal via het mechanisme van **autoboxing** de `double temperatuur` automatisch converteren naar een instantie van `Double` bij aanroep van de `add` methode

Bepalen of er temperaturen kleiner dan of gelijk aan 0 zitten in `temperaturen`

```
    public boolean bevatVriestemperaturen() { ①
        for (double t : temperaturen) { ②
            if (t <= 0)
                return true;
        }
        return false;
    }
```

① ook om de elementen van een `List` te overlopen kunnen we gebruik maken van de **enhanced for loop**, zodra we een temperatuur vinden die kleiner dan of gelijk aan 0 is kunnen we beslissen dat er vriestemperaturen werden opgemeten; merk op dat de lus variabele `t` van het primitief type `double` is, unboxing gebeurt automatisch!

De hoogste temperatuur in `temperaturen` vinden

```
    public double geefHoogsteTemperatuur() { ①
        double warmsteTemperatuur = temperaturen.get(0); ②
        for (double t : temperaturen) {
            if (t > warmsteTemperatuur)
                warmsteTemperatuur = t;
        }
        return warmsteTemperatuur;
    }
```

① deze methode retourneert de hoogste temperatuur uit de lijst `temperaturen`; we maken geen gebruik van de wrapper klasse, de lus variabele en het return type is primitief: `double`

② indien je reeds gebruik maakt van JDK 21 kan je in plaats van `.get(0)` gebruik maken van het meer verbose `.getFirst()`...

Het volgnummer van de warmste dag bepalen

```
    public int geefWarmsteDag() {
        return temperaturen.indexOf(geefHoogsteTemperatuur()) + 1; ①
    }
```

① in de Java API docs staat helder uitgelegd wat de `indexOf` methode doet; merk op dat op **index 0** de temperatuur zit die ingegeven werd voor **dag 1**...

Bouwen van een `String` met het dag aan dag overzicht van `temperaturen`

```
    public String toString() {
        int aantalDagen = temperaturen.size(); ①
        String resultaat = String.format("Overzicht temperaturen per dag voor %d
dag%s%n", aantalDagen,
                aantalDagen > 1 ? "en" : "");
        for (int i = 0; i < aantalDagen; i++) { ②
            resultaat += String.format("Dag %d: %.1f graden%n", i + 1, temperaturen
.get(i));
        }
        return resultaat;
    }
```

① het aantal elementen in de lijst is het aantal dagen waarvoor temperaturen werden ingegeven

② binnen een klassieke for lus gaan we voor elke dag, `i+1`, de ingegeven temperatuur, `.get(i)`, ophalen en toevoegen aan de `String resultaat`; *merk weerom op dat we ons niets hoeven aan te trekken van het feit dat de lijst* ***objecten van de wrapper klasse*** **`Double`** *bevat, Java zal via het mechanisme van* ***unboxing*** *deze instantie van* `Double` *automatisch converteren naar een primitieve `double`*

De applicatie

```
    public static void main(String[] args) {
        new TemperatuurApp().leesTemperaturenEnToonStatistieken();
    }

    private void leesTemperaturenEnToonStatistieken() {
        TemperatuurStatistiek ts = new TemperatuurStatistiek(); ①

        // Inlezen van temperaturen tot sentinel -100
        int dag = 1;
        double temperatuur = geefTemperatuur(dag++);
        while (temperatuur != -100) {
            ts.voegTemperatuurToe(temperatuur); ②
            temperatuur = geefTemperatuur(dag++);
        }

        // Bevragen van ts en alle resultaten tonen
        System.out.println(ts.toString());
        System.out.printf("%nEr werden %svriestemperaturen opgemeten.%n", ts
.bevatVriestemperaturen() ? "" : "geen ");
        System.out.printf("Dag %d was de warmste dag met %.1f graden.%n", ts
.geefWarmsteDag(),
                ts.geefHoogsteTemperatuur());
        double gezochteTemperatuur = 20;
        System.out.printf("Werd er een temperatuur van %.1f graden opgemeten? %s%n",
gezochteTemperatuur,
                ts.isGemeten(gezochteTemperatuur) ? "ja" : "nee",
gezochteTemperatuur);
    }
```

```
    private double geefTemperatuur(int dag) { ③
        System.out.printf("Geef temperatuur in voor dag %d > ", dag);
        return invoer.nextDouble();
    }
```

① er wordt een instantie van `TemperatuurStatistiek ts` aangemaakt, vanuit deze methode kunnen we nu communiceren met `ts`

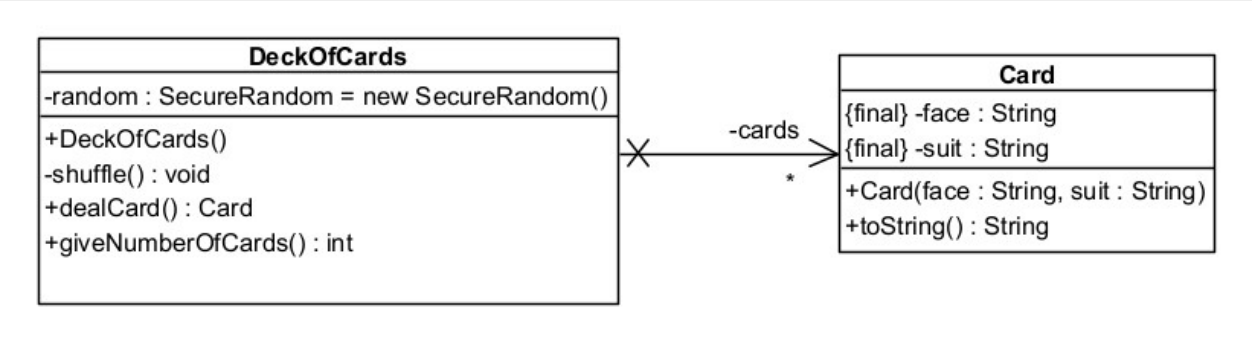
② elke ingelezen temperatuur wordt toegevoegd aan onze `ts`

③ hulpmethode voor het inlezen van de temperatuur met vermelding van de dag

4.4. Voorbeeld: pak speelkaarten maken en uitdelen

In deze case study gaan we een pak van 52 speelkaarten maken en twee spelers elk 8 kaarten uit het pak geven. De kaarten die in de handen van de spelers zitten worden afgedrukt in de console.

Het DCD van de domeinklassen in dit voorbeeld ziet er als volgt uit:



De *pijl* tussen `DeckOfCards` en `Card` noemen we een *associatie* en betekent dat in Java `DeckOfCards` een verzameling `Card`-objecten bevat, dit zal zich vertalen in een private instance variable `ArrayList<Card> deck` in de klasse `DeckOfCards`. In een volgend hoofdstuk worden *UML associaties* uitgebreid behandeld…

- `Card` stelt één speelkaart voor. Een kaart heeft
  - `face`, 1 van volgende waarden: "Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack","Queen", "King"
  - `suit`, 1 van volgende waarden: "Hearts", "Diamonds", "Clubs", "Spades"
- `DeckOfCards` stelt een pak kaarten voor. Wanneer een `DeckOfCards` wordt aangemaakt bevat deze 52 `Card`-s die in willekeurige volgorde zitten. Telkens een `Card` uitgedeeld wordt via de deal methode verdwijnt deze uit de het pak kaarten. Daar het aantal kaarten in een `DeckOfCards` zal variëren doorheen de tijd zullen we de kaarten bijhouden in een `List`.

4.4.1. De klasse Card

```
package domein;
```

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

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

②
    @Override
    public String toString() {
        return String.format("%s of %s", face, suit);
    }
}
```

① declaratie van 2 final fields: de `face` en `suit` van een `Card` zijn na de creatie van de `Card` onveranderlijk.

② de methode toString retourneert de tekstweergave van een Card, bv. "queen of spades", dit is dus `face` of `suit`. In een later hoofdstuk wordt verklaard waarom er boven deze methode de annotatie `@Override` staat, dit zal hier verder niet van belang zijn.

4.4.2. De klasse DeckOfCards:

```
package domein;

import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;

public class DeckOfCards {
    private List<Card> cards; ①
    private SecureRandom random = new SecureRandom();

    public DeckOfCards() {
②
        String[] faces = { "Ace", "Two", "Three", "Four", "Five", "Six", "Seven",
"Eight", "Nine", "Ten", "Jack",
                "Queen", "King" };
        String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };

        cards = new ArrayList<>(); ③

④
        for (String face : faces)
            for (String suit : suits)
                cards.add(new Card(face, suit));
```

```
        shuffle();
    }

    private void shuffle() { ⑤
        for (int index = 0; index < cards.size(); index++) { ⑥
            int indexToSwap = random.nextInt(cards.size()); ⑦
⑧
            Card temp = cards.get(index);
            cards.set(index, cards.get(indexToSwap));
            cards.set(indexToSwap, temp);
        }
    }

    public int giveNumberOfCards() { ⑨
        return cards.size();
    }

    public Card dealCard() { ⑩
        if (!cards.isEmpty()) {
            return cards.remove(0); ⑪
        }
        return null; ⑫

    }
}
```

① het field `cards` is een `List<Card>`

② we declareren twee hulp-arrays; van deze arrays zullen we handig gebruik maken om alle 52 kaarten te maken

- de array `faces` bevat alle 13 mogelijke waarden.
- de array `suits` bevat alle 4 mogelijke kleuren.

③ instantiatie van de instance variable `cards` levert een lege lijst op

④ de `cards` worden gemaakt door alle mogelijke combinaties van de waarden uit de arrays `faces` en `suits` te nemen. Nadien worden de kaarten door elkaar geschud via een aanroep naar de methode `shuffle`

⑤ de methode `shuffle` gebruikt een algoritme dat met één doorloop van de array in een lus de kaarten door elkaar schudt. Elke kaart wordt verwisseld met een kaart op een willekeurige gekozen plaats

⑥ we doorlopen alle kaarten van de stapel

⑦ `indexToSwap` wordt een willekeurig gekozen getal uit het interval `[0, deck.size()[`

⑧ om de kaarten op posities `index` en `indexToSwap` te wisselen wordt gebruik gemaakt van een hulp variabele `temp`

⑨ deze methode retourneert het aantal kaarten in dit `DeckOfCards`, dit is het aantal elementen in de instance variable `cards`

⑩ deze methode verwijdert een kaart uit `cards` en retourneert de verwijderde kaart

⑪ verwijdert en retourneert het eerste element van een `List`, JDK 21 kent hiervoor `.removeFirst()`

⑫ indien er geen kaarten meer zijn (`isEmpty`) wordt `null` geretourneerd

4.4.3. De applicatie klasse

```
package cui;

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

import domein.Card;
import domein.DeckOfCards;

public class DeckOfCardsApplication {

    public static void main(String args[]) {
        new DeckOfCardsApplication().makeDeckAndGiveCards();
    }

    private void makeDeckAndGiveCards() {
        DeckOfCards deck = new DeckOfCards(); ①
        System.out.printf("Before dealing: the fresh deck of cards contains %d
cards.%n", deck.giveNumberOfCards());

②
        List<Card> handPlayer1 = new ArrayList<>();
        List<Card> handPlayer2 = new ArrayList<>();

③
        for (int i = 0; i < 8; i++) {
            handPlayer1.add(deck.dealCard());
            handPlayer2.add(deck.dealCard());
        }

        showHand("Player 1", handPlayer1);
        showHand("Player 2", handPlayer2);

        System.out.printf("%nAfter dealing: the deck of cards now contains %d
cards.%n", deck.giveNumberOfCards());
    }

    private void showHand(String playerName, List<Card> cards) {
        System.out.printf("%nHand of %S%n", playerName);
        for (Card card : cards) {
            System.out.printf("- %s%n", card);
        }
    }
```

```
}
```

① een object van de klasse `DeckOfCards` wordt aangemaakt, we zullen via de publieke methodes in de klasse `DeckOfCards` kunnen communiceren met dit object

② elke speler gaat een aantal kaarten krijgen die bijgehouden worden in een `List<Card>`

③ geef elke speler om beurt een kaart, elke speler zal uiteindelijk 8 kaarten hebben

4.4.4. Voorbeeld uitvoer

```
Before dealing: the fresh deck of cards contains 52 cards.

Hand of PLAYER 1
- Four of Clubs
- Five of Hearts
- Nine of Spades
- Nine of Hearts
- Four of Spades
- Eight of Hearts
- Two of Clubs
- Jack of Spades

Hand of PLAYER 2
- Ace of Hearts
- Ten of Clubs
- Three of Clubs
- Two of Hearts
- Eight of Diamonds
- Nine of Clubs
- Jack of Hearts
- Six of Hearts

After dealing: the deck of cards now contains 36 cards.
```

5. Tweedimensionale arrays

5.1. Definitie

In het begin van dit hoofdstuk werden ééndimensionale arrays, wat wij gewoonweg arrays hebben genoemd, besproken. Daar de **elementen** van een array **referenties** kunnen bevatten is het dus ook mogelijk om in de elementen van een array referenties naar andere arrays te stoppen. Zo komen we tot multidimensionale structuren.

Multidimensionale arrays zijn geneste arrays. Het zijn arrays die **elementen** hebben die op hun beurt **weer arrays** zijn. Zo is een **tweedimensionale array** een array van arrays. Een driedimensionele array is een array van twee-dimensionele arrays. En zo kunnen we verder

gaan...

In dit deeltje gaan we ons beperken tot tweedimensionale arrays. Een **tweedimensionale array** kan je makkelijk voorstellen als een tabel. De twee dimensies vertalen zich dan in **rijen** en **kolommen**.

5.2. Declaratie en initialisatie

Om een tweedimensionale array te declareren maken we gebruik van dubbele vierkante haakjes.

```
int[][] matrix;
```

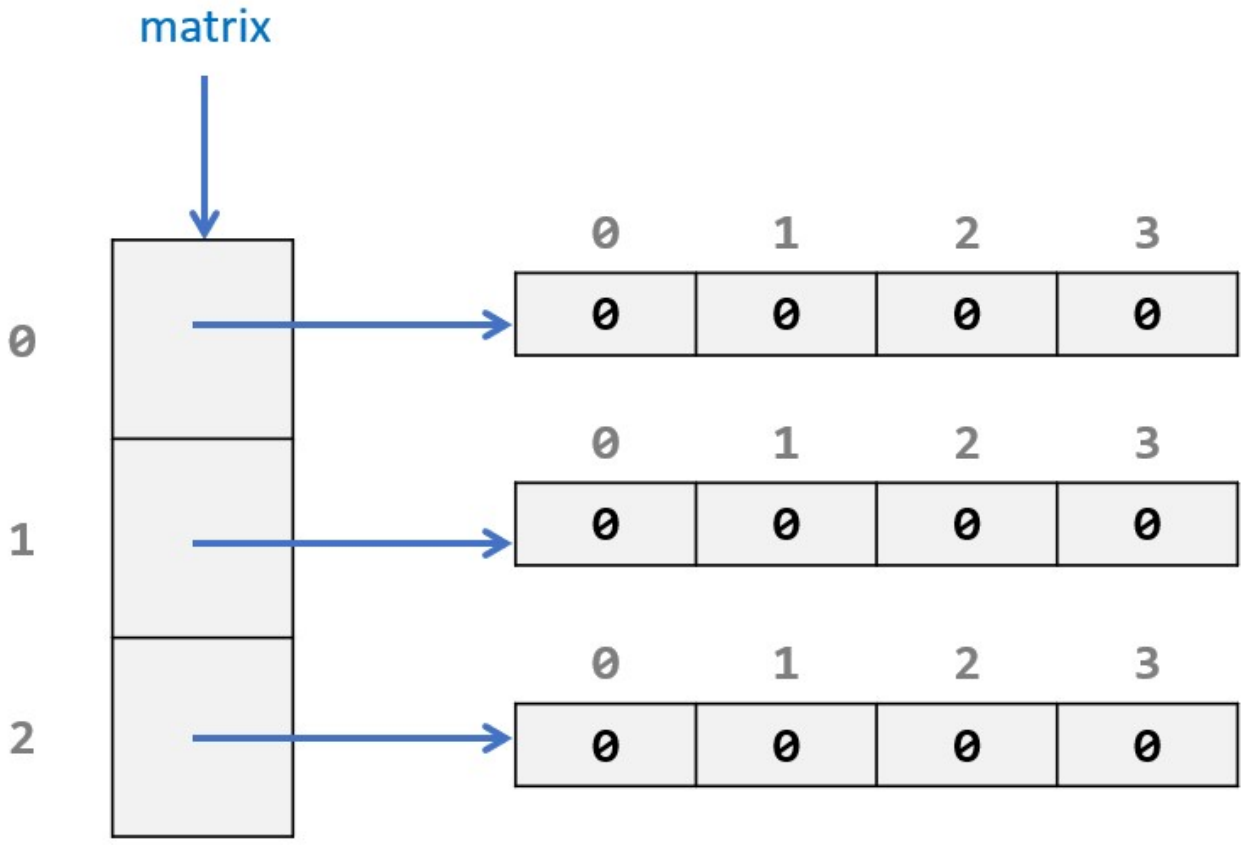
Tijdens instantiatie kunnen we nu een invulling geven aan beide dimensies.

Het is een conventie om in een tweedimensionale array te verwijzen naar de **eerste dimensie** als ***rijen*** en de **tweede dimensie** als ***kolommen***

Volgende initialisatie maakt een matrix van 3 rijen en 4 kolommen.

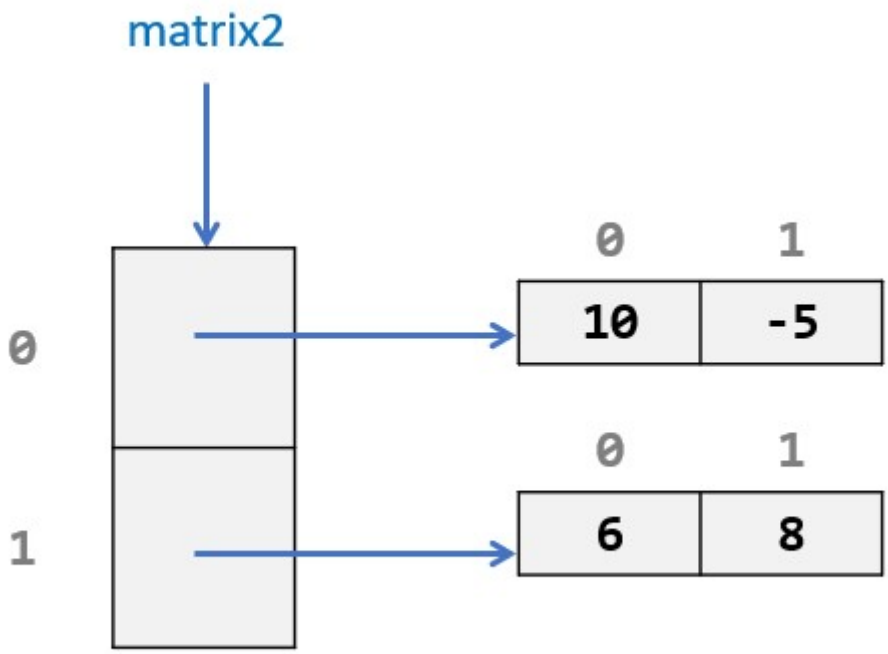
```
int[][] matrix;
matrix = new int[3][4];
```

Na deze instantiatie bevatten alle elementen van onze tweedimensionale structuur de default waarde 0. Het resultaat kunnen we als volgt visualiseren.



Net zoals we expliciet waarden konden opgeven bij declaratie en initialisatie van een ééndimensionale arrays kunnen we dit ook doen voor tweedimensionale arrays.

```
int[][] matrix2 = { { 10, -5 }, { 6, 8 } };
```



Zonder expliciet de referenties te tonen ziet onze `matrix2` er als volgt uit.

	kolom 0	**kolom 1**
rij 0	10	-5
rij 1	6	8

5.3. Werken met tweedimensionale arrays

Via de naam van de variabelen en een rij- en kolom index kunnen we weer aan de verschillende elementen in de verzameling. Hiervoor zetten we de rij- en kolom index apart tussen rechte haakjes, na de naam van de variabele.

Hieronder zie je hoe de elementen in `matrix` benoemd zijn:

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

Laten we kijken naar het effect van enkele statements die werken met onze net geïnitialiseerde `matrix` (in onderstaande afbeelding werden geen referenties getekend en zie je enkel de inhoud van de 3 x 4 elementen):

	0	**1**	**2**	**3**
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

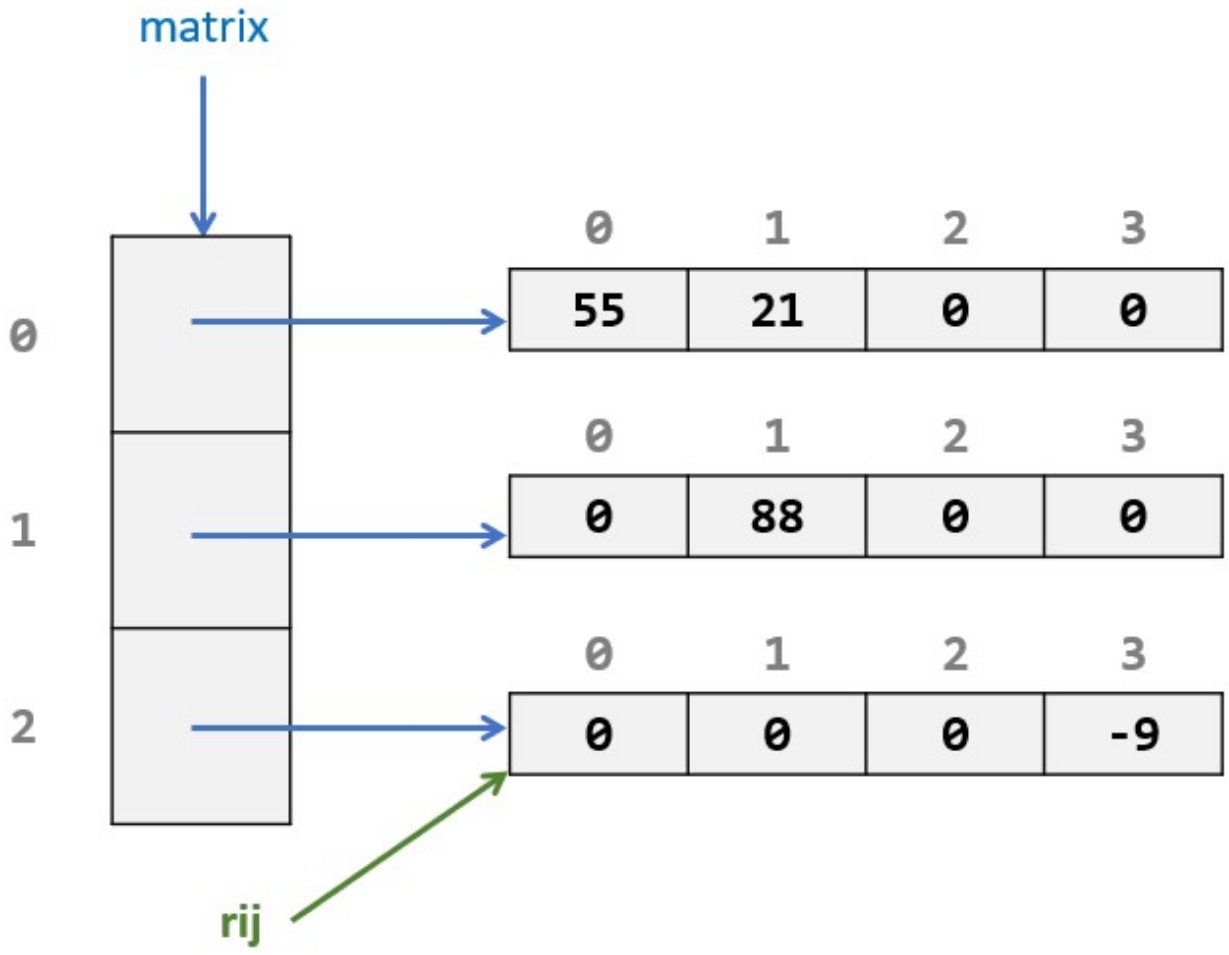
```
matrix[0][1] = 21; // 0-de rij, 1-ste kolom
matrix[0][0] = 55; // 0-de rij, 0-de kolom
matrix[1][1] = 88; // 1-ste rij, 1-ste kolom
matrix[2][3] = -9; // 2-de rij, 3-de kolom
```

Resultaat:

	0	**1**	**2**	**3**
0	55	21	0	0
1	0	88	0	0
2	0	0	0	-9

Merk op dat de variabele `matrix` uiteindelijk **3** elementen bevat: drie arrays die op hun beurt elk **4** `int`-elementen bevatten. Op volgende manier kunnen we bijvoorbeeld de derde rij van onze tabel (dit is een array van `int`-s) vastpakken in de variabele `rij`:

```
int[] rij= matrix[2];
```




```
int[] rij= matrix[2];
for (int kolom = 0; kolom < rij.length; kolom++){
    rij[kolom] = 44;
}
```

In de derde rij van `matrix` kregen alle elementen de waarde 44:

	0	1	2	3
0	55	21	0	0
1	0	88	0	0
2	44	44	44	44

5.4. Jagged arrays

Het is ook mogelijk om tweedimensionale arrays te maken waarbij niet elke rij evenveel kolommen telt. We noemen dit *jagged* arrays.

In onderstaand voorbeeld maken we een tweedimensionale array waarbij de eerste rij 5 kolommen, en de tweede rij 10 kolommen bevat.

```
int[][] jagged1 = new int[2][]; ①

jagged1[0] = new int[5]; ②
jagged1[1] = new int[10]; ③
```

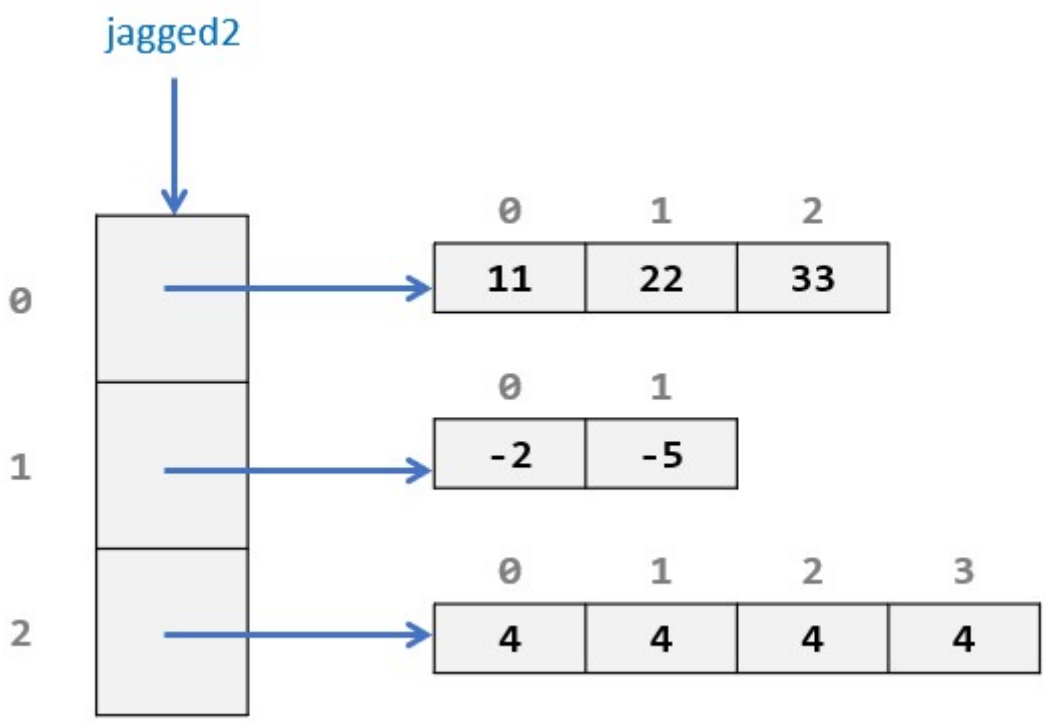
① de array `jagged1` krijgt **2 rijen**; de kolom dimensie kreeg geen waarde, wat meteen ook betekent dat de rijen ingesteld werden op hun default waarde `null`

② de eerste rij wordt een array van **5** `int`-s, we hebben dus **5 kolommen voor de eerste rij**

③ de tweede rij wordt een array van **10** `int`-s, we hebben dus **10 kolommen voor de tweede rij**

We kunnen jagged arrays ook op een expliciete manier initialiseren:

```
int[][] jagged2 = { { 11, 22, 33 }, { -2, -5 }, { 4, 4, 4, 4 } };
```



Bij dergelijk arrays moeten we heel voorzichtig omspringen met de indices.

```
int el1 = jagged2[0][3]  // el1 krijgt de waarde 33
int el2 = jagged2[1][3]  // ArrayIndexOutOfBoundsException!!!
```

5.5. Nog enkele voorbeeldjes

5.5.1. Afdrukken tweedimensionale array

In onderstaand fragment wordt de inhoud van een tweedimensionale array rij per rij afgedrukt:

```
package cui;

public class TweedimensionaleArraysVoorbeeld1 {
    public static void main(String args[]) {
        new TweedimensionaleArraysVoorbeeld1().maakEnToonEenTweedimensionaleArray();
    }

    private void maakEnToonEenTweedimensionaleArray() {

        int[][] jagged2 = { { 11, 22, 33 }, { -2, -5 }, { 4, 4, 4, 4 } };

        System.out.println("De waarden in de array rij per rij:\n");
        String uitvoer = "";
        for (int[] rij : jagged2) { ①
            for (int element : rij) ②
                uitvoer += String.format("%8d", element);
            uitvoer += "%n";
        }
        System.out.printf(uitvoer);
    }
}
```

① buitenste lus: de tweedimensionale array is opgebouwd uit rijen, de variabele rij zal telkens een rij van de tweedimensionale array jagged2 bevatten, het type van de lus variabele is dus `int[]`

② geneste lus: binnen een rij hebben we de `int`-s, de losse gehele getallen, de lus variabele is van het type `int`

```
De waarden in de array rij per rij:

11    22    33
-2    -5
4     4     4     4
```

5.5.2. Opvullen tweedimensionale array

In volgend voorbeeld wordt een array geïnitialiseerd. Elk element krijgt een waarde die gelijk is aan de som van zijn rij- en kolom index.

```
package cui;

public class TweedimensionaleArraysVoorbeeld2 {
    public static void main(String args[]) {
        new TweedimensionaleArraysVoorbeeld2().vulTweedimensionaleArrayOp();
    }

    private void vulTweedimensionaleArrayOp() {
        int[][] jagged1 = new int[4][];
```

```
        jagged1[0] = new int[5];
        jagged1[1] = new int[5];
        jagged1[2] = new int[7];
        jagged1[3] = new int[2];

        for (int rij = 0; rij < jagged1.length; rij++) ①
            for (int kolom = 0; kolom < jagged1[rij].length; kolom++) ②
                jagged1[rij][kolom] = rij + kolom;
    }
}
```

① buitenste lus: de variabele rij wordt gebruikt om te itereren over de rijen, de eindconditie is bepaald door het aantal rijen in de tweedimensionale array: `jagged1.length`

② geneste lus: de variabele kolom wordt gebruikt om te itereren over de elementen van 1 rij, let goed op de eindconditie, deze is nu bepaald door het aantal kolommen in de huidige rij: `jagged1[rij].length`

Indien we de array `jagged1` afdrukken volgens het eerste voorbeeld krijgen we volgende uitvoer

```
De waarden in de array rij per rij:

      0     1     2     3     4
      1     2     3     4     5
      2     3     4     5     6     7     8
      3     4
```

6. Extra's

6.1. Declaratie van een array

Bij de declaratie van een array mag je er ook voor kiezen om de rechte haakjes na de array variabele te zetten in plaats van na het type.

```
    // int[] getallen = new int[10]

    int getallen[] = new int[10];
```

Declaraties in deze vorm komen minder voor maar kunnen eventueel een voordeel opleveren daar je nu arrays en gewone variabelen van eenzelfde type in 1 keer kunt declareren en eventueel instantiëren.

```
    int eenGetal = 88, getallen[] = new int[10];
```

6.2. Give me a break...

Via het `break` statement kan je *vroegtijdig* uit een lus springen. Je kan dit statement gebruiken binnen eender welke lus structuur in Java. Van zodra een `break` statement wordt uitgevoerd stopt de iteratie en wordt er met de statements net na de lus verdergegaan.

In volgend voorbeeld wordt een `ArrayList<Integer>` overlopen om de index te vinden van het eerste getal groter dan 10.

```
package cui;

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

public class ArrayListBreakDemo {
    public static void main(String[] args) {
        new ArrayListBreakDemo().toonBreakStatement();
    }

    private void toonBreakStatement() {
        List<Integer> integerList = new ArrayList<>();
        integerList.add(5);
        integerList.add(8);
        integerList.add(12);
        integerList.add(20);
        integerList.add(3);

        int indexEersteGetalGroterDan10 = -1;
        for (int i = 0; i < integerList.size(); i++) {
            if (integerList.get(i) > 10) { ①
                indexEersteGetalGroterDan10 = i;
                break; ②
            }
        }
        if (indexEersteGetalGroterDan10 != -1)
            System.out.printf("Het eerste element groter dan 10 zit op index %d",
indexEersteGetalGroterDan10);
        else
            System.out.println("De lijst bevat geen element groter dan 10");
    }
}
```

① zodra een getal groter dan 10 gevonden wordt zal de index van dit getal toegekend worden aan de variabele indexEersteGetalGroterDan10 en hoeven we niet meer verder te zoeken

② de lus wordt niet meer verdere uitgevoerd, het programma gaat verder met het if-statement na de lus en drukt het resultaat af

And now, time for a real a break, before jumping to chapter 5 ;-)