



// 01

## Een array van objecten

`new` maakt enkel de **array**, nog **geen objecten**: alles is `null`.

```
Student[] mijnKlas = new Student[20];  
mijnKlas[0] = new Student();  
mijnKlas[2] = new Student();
```

|         |      |         |      |     |      |
|---------|------|---------|------|-----|------|
| Student | null | Student | null | ... | null |
| 0       | 1    | 2       | 3    |     | 19   |

Alles vullen: een for-loop met `mijnKlas[i] = new Student();`

// LET OP

Een element benaderen dat nog `null` is, geeft een `NullPointerException`. Maak het object eerst aan.

// 02

## Objecten in de array

Index en **dot-operator**: `mijnKlas[3].Naam = "Duke Peekaboo";`  
Met de **array initializer syntax** maak je de objecten meteen mee aan. De lengte bepaalt de compiler.

```
Student[] mijnKlas = new Student[]  
{  
    new Student(),  
    jos // een bestaand object mag ook  
};
```

`jos` en `mijnKlas[1]` zijn nu **hetzelfde** object. Na `jos = null;` blijft het bestaan via de array.

// 03

## List: een array die meegroeit

`List<>` is een **generieke** klasse: tussen `< >` staat het type van de elementen. Geen begingrootte nodig, de lijst **groeit mee**.

```
List<string> personages = new List<string>();  
personages.Add("Reinhardt");  
personages.Add("Mercy");  
personages[1] = "Torbjorn"; // zoals een array  
for (int i = 0; i < personages.Count; i++)  
{  
    Console.WriteLine(personages[i]);  
}
```

// LET OP

Een `List` heeft `Count`, een array (en een `string`) heeft `Length`.

// 04

## Wat kan een List nog?

| METHODE                   | DOET                                                 |
|---------------------------|------------------------------------------------------|
| <code>Insert(i, x)</code> | voegt <code>x</code> in op plaats <code>i</code>     |
| <code>Remove(x)</code>    | verwijdert <code>x</code>                            |
| <code>RemoveAt(i)</code>  | verwijdert het element op index <code>i</code>       |
| <code>Clear()</code>      | maakt de lijst leeg, <code>Count</code> wordt 0      |
| <code>IndexOf(x)</code>   | index van <code>x</code> , of <code>-1</code>        |
| <code>Contains(x)</code>  | <code>true</code> als <code>x</code> in de lijst zit |
| <code>Sort()</code>       | sorteert, nog niet bij eigen klassen                 |

// LET OP

`IndexOf` en `Contains` vergelijken objecten op **referentie**: twee aparte objecten met dezelfde inhoud gelden als verschillend.

**Array of List?** Vast aantal (12 maanden): array. Komt er iets bij of valt er iets weg (winkelmandje): `List`.

// 05

## foreach

Overloopt **alle** elementen van een array of `List`, zonder teller. De **iteration variable** krijgt elk element om de beurt. Ze is **read-only**: bij objecten kan je de referentie niet vervangen, het object zelf mag je wel aanpassen.

```
foreach (Student eenStudent in deKlas)  
{  
    Console.WriteLine(eenStudent.Naam);  
    eenStudent.Geboortejaar++; // mag  
    eenStudent = new Student(); // fout  
}
```

// LET OP

Elementen toevoegen of verwijderen tijdens een `foreach` compileert, maar crasht: `InvalidOperationException`, *Collection was modified*. Verwijderen doe je met een for die **van achter naar voor** telt.

// 06

## var

De compiler leidt het type af uit wat rechts staat. Daarna ligt het vast: C# blijft **statically typed**.

```
var getal = 5; // int  
var ikke = new Leerkracht(); // Leerkracht  
var klas = new List<Student>(); // List<Student>
```

// 07

## Queue en Stack

**Queue**: first in, first out (FIFO), zoals de rij aan de kassa.

**Stack**: last in, first out (LIFO), zoals een stapel papier.

```
Queue<string> rij = new Queue<string>();  
rij.Enqueue("eerste"); // achteraan bij  
rij.Enqueue("tweede");  
Console.WriteLine(rij.Dequeue()); // vooraan af  
Stack<string> stapel = new Stack<string>();  
stapel.Push("eerste"); // bovenop  
stapel.Push("tweede");  
Console.WriteLine(stapel.Pop()); // bovenste af
```

UITVOER  
eerste  
tweede

// 08

## Dictionary

Elk element is een **key** met een **value**. De key is uniek. Tussen `< >` eerst het type van de key, dan dat van de value.

```
var klanten = new Dictionary<int, string>();  
klanten.Add(123, "Tim Dams");  
klanten.Add(6463, "James Bond");  
Console.WriteLine(klanten[123]);  
klanten.Remove(6463);
```

UITVOER  
Tim Dams

Opzoeken via de key gaat **razendsnel**. In een `foreach (var item in klanten)` lees je `item.Key` en `item.Value`. Als key neem je voorlopig `int` of `string`.