Dictaat C# collecties

versie juni 2019

1. Collecties

C# kent, net als de meeste andere programmeertalen, naast klassen nog meer complexe datatypes. Twee voorbeelden daarvan zijn array's en lists. Beide types zijn bedoeld om meerdere waardes van een type te kunnen bewaren. Bijvoorbeeld een verzameling getallen, een lijst van strings of een aantal boten. Soms weet je van te voren niet hoeveel waardes je precies moet onthouden en soms zijn het er gewoon te veel om allemaal op te slaan als losse variabelen. Dit zijn allemaal goede redenen om collecties te gebruiken.

We kijken eerst naar array's. Dit is het basistype om meerdere waardes van een en hetzelfde type te kunnen bewaren. Dit datatype werkt ook in de niet object-georiënteerde programmeertalen, zoals bijvoorbeeld C.

1.1 Array's

Bekijk het volgende voorbeeldje:

```
double[] getallen = new double[10];
getallen[0] = 2.5;
MessageBox.Show("Getal nummer 1 = " + getallen[0]);
```

Hier wordt een array van doubles aangemaakt. Om precies te zien een array van precies 10 doubles. Op de tweede regel code wordt het eerste getal in de rij gelijk gemaakt aan 2,5. Met de [blokhaken] kun je elementen een nieuwe waarde geven. Ook kun je getallen opvragen met de [blokhaken]. Dit zie je op de regel daaronder, in de messagebox. Simpel toch? Wat voor getallenreeks wordt er in onderstaand stukje code dan opgeslagen in de array?

```
int[] getallen = new int[10];
for (int i = 0; i < getallen.Length; i++)
{
   getallen[i] = (i + 1) * 5;
}
```

Zoals je ziet heeft een array altijd een vaste grootte. In beide voorbeelden hierboven wordt plek gereserveerd voor 10 getallen (doubles of integers). Met de Length eigenschap van een array (ook te zien in bovenstaande stukje code) kun je opvragen hoeveel elementen je maximaal in de array kunt opslaan. Uiteraard kun je ook andere zaken dan integers en doubles opslaan met een array. Zo kun je bijvoorbeeld ook een array van strings of je eigen objecten aanmaken. Probeer eens uit! Een array kan wel altijd maar 1 soort (type) variabele opslaan.

Net als bij klassen moet je een array aanmaken met new. Wanneer je dat niet doet krijg je daar van Visual Studio een melding over. Als je niet- array probeert te benaderen tijdens de programmauitvoering zond

Let op: We beginnen te tellen bij 0. Het eerste element in een array staat op positie 0. Het getal op *index* 5 is het zesde getal in de reeks.

Breinbrekers: Wist je dat een string eigenlijk een char[] array is? En wist je dat je ook een array van array's (van array's (van array's…)) kunt maken (2D- en 3D-array's)? Leuke uitdagingen om mee te spelen!

1.2 Lists

Het nadeel van een array is dat je van te voren moet specificeren hoeveel plekken je wilt reserveren voor je data. Echter, niet altijd is dit van te voren bekend. Wanneer je een gebruiker zelf elementen laat toevoegen in het systeem weet je niet hoeveel er toegevoegd zullen worden. Daarom is de List bedacht: de meegroeiende array. Wanneer je meer plek nodig hebt zal de List automagisch groeien in maximale grootte.

```
int[] getallen = new int[10];
for (int i = 0; i < 10; i++)
{
   getallen[i] = i * 10;
}

Getallen[10] = 10 * 10;
```

In bovenstaand stukje code worden een array van maximaal 10 integers aangemaakt. Vervolgens worden de tafel van 10 getal voor getal aan de array toegevoegd. Na de for-loop wordt het laatste element aan de array toegevoegd, 10 x 10. Echter, deze zal niet meer passen. Er kunnen maximaal 10 integers in de array en 0 x 10 tot en met 9 x 10 zijn al toegevoegd. Dat zijn er dus al 10 in totaal. Het getal 100 zal niet meer passen en geeft dan een *IndexOutOfRangeException* foutmelding. In onderstaand voorbeeld wordt een alternatief met een List gegeven.

```
List<int> getallen = new List<int>();
for (int i = 0; i < 10; i++)
{
   getallen.Add(i * 10);
}

getallen.Add(10 * 10);
```

Dit stuk code zal niet meer de aangeven dat er geen plek meer is voor het getal 100. De List groeit mee met de vraag. Als er meer plek nodig is zal de List dus plek vrijmaken. Een array kan dit niet uit zichzelf!

Hieronder zie je een voorbeeld met een List van strings. In plaats van de for-loop wordt de foreach gebruik. Dit is een nieuw soort herhalingsstructuur waarin er om de loop al een element uit de collectie wordt gehaald om er mee verder te werken. Dit scheelt dus weer code! Dit werkt overigens ook met array's.

```
string s = "";
List<string> woorden = new List<string>();
woorden.Add("Hallo");
woorden.Add("allemaal!");

foreach (var woord in woorden)
{
   s = s + woord + " ";
}
MessageBox.Show(s);
```

Waarschijnlijk zie je het al, maar de tekst "Hallo allemaal!" zal verschijnen in een MessageBox. Bij elke iteratie (elke keer dat C# door de collectie loopt) pakt hij het volgende element uit de verzameling. Deze stopt hij in de variabele woord. Deze variable bevat altijd het element wat op dat moment actueel is in de herhaling. Hij pakt dus eest element met index 0, daarna die met index 1, index 2, en zo voorts. Je hoeft dus niet zelf meer de index bij te houden en ook het element wordt automatisch voor je uit de collectie gepakt.

1.3 List methodes

Het leuke van alles opslaan in een List is dat je er gemakkelijk elementen in terug kunt vinden. Daar gebruiken we twee methodes voor, de Contains en IndexOf. Met de Contains methode kun je controleren of een element voorkomt in de List. Deze geeft dus `true` of `false` terug. De IndexOf methode geeft de index terug van het element dat je zoekt. Als het element niet is gevonden zal -1 terug worden gegeven.

```
List<string> talen = new List<string>();

talen.Add("Java");
...

if (talen.Contains("C#"))
{
   MessageBox.Show("C# komt voor in de lijst!");
}
```

In bovenstaand voorbeeld wordt het gebruik van de Contains methode laten zien. Aan de talen List worden verschillende strings toegevoegd. Allereerst wordt de string “Java” aan de List toegevoegd. Op de drie puntjes kunnen nog andere talen zijn toegevoegd.

Als je weet dat een bepaald element in de List voorkomt kun je met de IndexOf methode kijken op welke index het element staat. Bekijk onderstaand voorbeeld.

```
List<string> talen = new List<string>();

talen.Add("Java");
...

int index = talen.IndexOf("C#");
MessageBox.Show("C# staat op index " + index);
```

Wanneer C# niet in de lijst zou staat zou de IndexOf methode aangeven dat de index gelijk is aan -1. Dat is een code die wordt gebruikt wanneer het element waarvan de index van wordt opgevraagd niet wordt gevonden in de List. Dat kun je natuurlijk ook voorkomen door eerst met Contains te kijken of het element voorkomt in de List. Probeer het eens zelf uit en kijk of je de twee kunt combineren!

Tip: De Contains en IndexOf methodes werken ook voor strings!

1.4 Voorbeelden

Voorbeeld	Uitleg

`public List<int> GeefTafelVan(int tafelVan)`
`{`
`  List<int> getallen = new List<int>();`
 	
`  for (int i = 0; i <= 10; i++)`
`  {`
`  getallen.Add(i * tafelVan);`
`  }`
 	
`  return getallen;`
`}`
	In de code hier links zie je een methode die de tafel van een getal teruggeeft, in de vorm van een List van integers. Je zou deze als volgt kunnen gebruiken:
	
	`List<int> tafelVanVijf = GeefTafelVan(5);`
	
	`foreach (var getal in tafelVanVijf)`
	`{`
	`  Console.Out.WriteLine(getal);`
	`}`

`char[] woord = { 'H','a','l','l','o','!' };`
`for (int i = 0; i < woord.Length; i++)`
`{`
`  Console.Out.Write(woord[i]);`
`}`
	Hier links zie je een stukje code om een array van characters op te slaan. Met een for-loop kun je de gegevens ophalen en afdrukken naar de console. Probeer het zelf eens uit!

Console.Out.Write(Environment.NewLine);	

List<double> inworp = new List<double>();
inworp.Add(.5);
inworp.Add(2.0);
inworp.Add(.5);
 	
if (inworp.Contains(.5))
{
MessageBox.Show("Er is minstens 1 " +
"muntstuk van vijftig cent ingeworpen!");
}
	Aan de linkerkant zie je een stukje code met een lijst van prijzen, opgeslagen als lijst van doubles. Wanneer een gebruiker een muntstuk van 50 cent heeft ingeworpen zal er een MessageBox worden weergegeven.

List<double> inworp = new List<double>();
 	
if (inworp.IndexOf(.5) == -1)
{
MessageBox.Show("Er is geen muntstuk " +
"van vijftig cent ingeworpen!");
}
	Bijna hetzelfde voorbeeld als hierboven. Hier wordt de IndexOf methode gebruikt om de index van het eerste vijftig cent munstuk in de lijst op te vragen. Als deze -1 teruggeeft is er geen muntstuk van vijftig cent gevonden.