

Enums, arrays en collections

Programming Basics

INHOUD

1	INLEIDING	3
2	ENUMS	4
1.1	Wat zijn enums?	4
1.2	Gebruik van enums	4
1.3	Index van enums	6
1.4	Nut van Enums	8
3	ARRAYS	9
3.1	Wat zijn arrays?	9
3.2	Een Array declareren	10
3.3	Een array instantiëren	10
3.3.1	Declareren, de elementen afzonderlijk instantiëren	10
3.3.2	Verkorte schrijfwijze: direct instantiëren	10
3.3.3	Nog korter	11
3.4	Toepassing	11
3.5	De klasse Array	12
3.6	Arrays met meerdere dimensies	14
4	COLLECTION KLASSEN	17
4.1	List	17
4.2	Dictionary	20
5	SAMENVATTING	24

1 INLEIDING

In programma's werkten we tot hiertoe met variabelen. Die konden wisselende waarden bevatten, maar slecht **één terzelfdertijd**. We konden wel meerdere waarden van dezelfde soort opslaan, maar dan enkel in een control zoals een listbox of een combobox (of m.b.v. meerdere variabelen).

Om dergelijke lijsten/reeksen van waarden in het geheugen op te slaan, hebben we in C# meerdere mogelijkheden. Ze variëren van een eenvoudige, vaststaande lijst met waarden naar complexere objecten.

We kunnen hiervoor gebruik maken van Enums, array's en collections, welke we verder in deze syllabus zullen bespreken.

2 ENUMS

1.1 WAT ZIJN ENUMS?

Enums worden gebruikt om een vaststaande reeks gerelateerde waarden in op te slaan. Je kunt ze bv. gebruiken voor:

- de dagen van de week
- de maanden
- de sterrenbeelden van de horoscoop
- gebruikersstatus
- operaties van een rekenmachine (optellen, aftrekken, delen en vermenigvuldigen)
- ...

De enige manier om de enum aan te passen, is door de broncode te veranderen.

Het moet dus echt wel gaan om zaken die nooit zullen veranderen.

1.2 GEBRUIK VAN ENUMS

Aangezien we nog steeds vier seizoenen hebben en er geen seizoenen zullen wegvallen of bijkomen zullen we even de vier seizoenen gebruiken als voorbeeld voor een enum.

1. Maak een nieuwe WPF applicatie (.NET Core)
2. Voorzie één button op je MainWindow met volgende properties:
 - **Name:** btnShowEnumValue
 - **Content:** Toon Enum waarde
 - **Event:** Click
3. We declareren een enum als volgt:

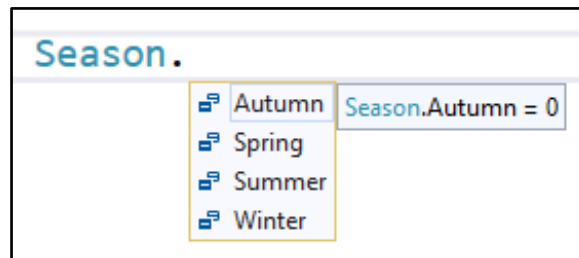
```
enum Season { Autumn, Winter, Spring, Summer }
```

```
21 public partial class MainWindow : Window
22 {
23     0 references
24     public MainWindow()
25     {
26         InitializeComponent();
27     }
28     1 reference
29     enum Season { Autumn, Winter, Spring, Summer }
30     1 reference
31     private void BtnShowEnumValue_Click(object sender, RoutedEventArgs e)
32     {
33     }
```



Momenteel gaan wij onze enumeraties globaal aanmaken in de code behind van onze WPF applicatie. In de toekomst gaan wij enumeraties echter onderbrengen in een class library zodat ze over je volledige applicatie beschikbaar zijn. Later meer hierover dus.

- Om één van de waardes van de enum `Season` op te halen moeten de naam van de enum typen, gevolgd door een punt. Door de intellisense die aanwezig is in Visual Studio worden alle waardes aanwezig in de enum `Season` getoond in een lijst:



- In ons voorbeeld willen we een waarde vanuit de enum `Season` tonen in een `MessageBox`. Dit kunnen we doen met onderstaande code. We kiezen vanuit de enum `Season` de waarde `Autumn` en zetten dit d.m.v. `ToString()` om naar een string die getoond kan worden in een `MessageBox`.

```
private void BtnShowEnumValue_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show(Season.Autumn.ToString());
}
```

- Indien we een andere waarde uit de enum `Season` willen tonen, moeten we `Autumn` vervangen door een andere waarde van de enum `Season`. Bijvoorbeeld `Summer`. Onze code ziet er dan zo uit:

```
private void BtnShowEnumValue_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show(Season.Summer.ToString());
}
```



Wat we hierboven doen (een enum waarde als string afbeelden) zullen we in de toekomst hoogstwaarschijnlijk nooit meer doen. In bovenstaande voorbeelden hadden we eigenlijk net zo goed de tekst “Autumn” of “Summer” kunnen afbeelden.

Enumeraties ga je in de praktijk hoofdzakelijk gebruiken om jou, als programmeur, te verplichten een waarde uit een lijst met vooraf gedefinieerde waarden te kiezen. Bv. wanneer we later met databases gaan werken zal dit zijn nut hebben. Een ander voorbeeld: enums worden ook gebruikt om de keuze van de afbeelding en knoppen te bepalen in een `MessageBox`.

Opnieuw, later komen we hier nog op terug.

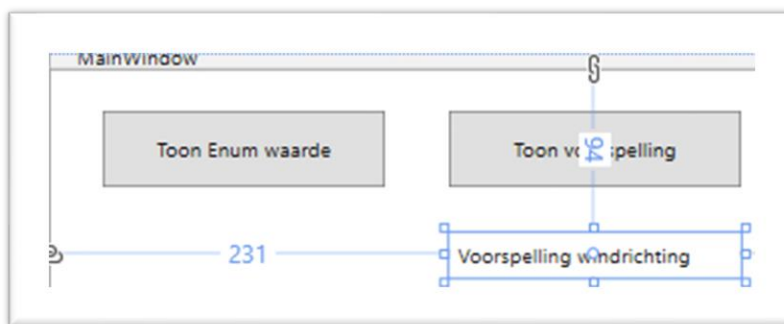
1.3 INDEX VAN ENUMS

Alle elementen in een enum hebben elk een index. Hierdoor is het mogelijk om eventuele bewerkingen uit te voeren met enums.

We tonen dit even voor aan de hand van een klein voorbeeld met een enum van windrichtingen.

7. Plaats een nieuwe button op je venster :

- **Button:** btnShowPrediction
- **Label:** lblPrediction



8. In onze code-behind definiëren we een enum met de windrichtingen:

```
enum WindDirection { East, South, North, West };
```

```
28  enum Season { Autumn, Winter, Spring, Summer }
29  enum WindDirection { East, South, North, West };
30
```

9. We creëren een click-event op onze button en voorzien volgende code:

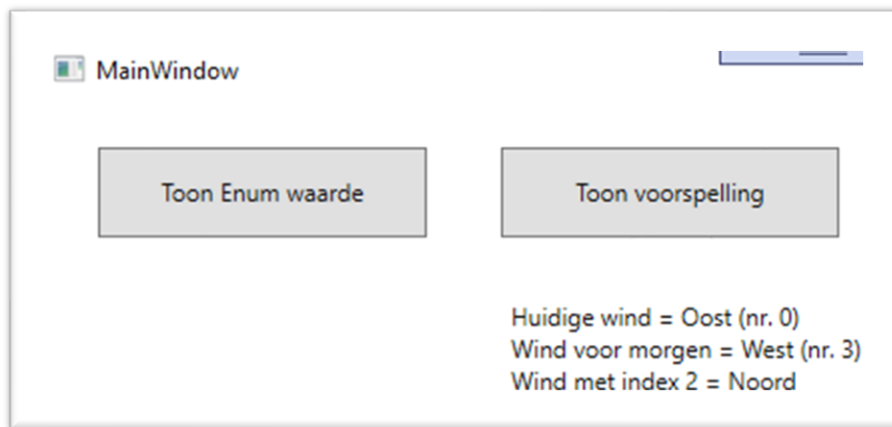
```
private void BtnShowPrediction_Click(object sender, RoutedEventArgs e)
{
    int currentWindIndex = (int)WindDirection.East;
    string currentWind = WindDirection.East.ToString();
    lblPrediction.Content = "Huidige wind = " + currentWind + " (nr. " + currentWindIndex +
    ")\n";

    int tomorrowWindIndex = (int)WindDirection.West;
    string tomorrowWind = WindDirection.West.ToString();
    lblPrediction.Content += "Wind voor morgen = " + tomorrowWind + " (nr. " +
    tomorrowWindIndex + ")\n";

    WindDirection directionTwo = (WindDirection)2;
}
```

```
lblPrediction.Content += "Wind met index 2 = " + directionTwo.ToString();
}
```

10. Na een klik op de knop “Toon voorspelling” zien we het label veranderen.



Met deze applicatie vragen we dus de index van een waarde van een enum op, alsook de waarde zelf.

- Willen we de index weten van een bepaalde waarde van een enum: `(int)EnumNaam.Waarde`

```
int currentWindIndex = (int)WindDirection.East;
```

- Willen we de tekstwaarde opvragen van een enum: `EnumNaam.Waarde.ToString()`

```
string currentWind = WindDirection.East.ToString();
```

- Willen we de waarde weten van een bepaalde index van een enum: `(EnumNaam)index`

```
WindDirection directionTwo = (WindDirection)2;
```

Standaard is de index van de eerste enumeratiewaarde 0, de tweede is dan 1 enz.

Indien je dat wenst, kan je de startwaarde aanpassen (dus anders dan 0).

Pas je enumeratie van de windrichtingen even als volgt aan:

```
enum Season { Autumn, Winter, Spring, Summer }
6 references
enum WindDirection { East = -1, South, North, West };
```

Laat je programma opnieuw lopen en let op het verschil:

Toon voorspelling

Huidige wind = Oost (nr. 0)
Wind voor morgen = West (nr. 3)
Wind met index 2 = Noord



Toon voorspelling

Huidige wind = Oost (nr. -1)
Wind voor morgen = West (nr. 2)
Wind met index 2 = West

1.4 NUT VAN ENUMS

Het nut van Enums is eerder beperkt. Zoals eerder al aangegeven, worden enumeraties vooral interessant wanneer je met klassen begint te werken en zeker wanneer je met meerderen gelijktijdig aan 1 project werkt. Je kan dan op bepaalde plaatsen je collega-programmeurs verplichten om met bepaalde voorgedefinieerde waarden te gaan werken.



De repository met de code uit dit hoofdstuk rond enumeraties kan je hier terugvinden :

git clone <https://github.com/howest-gp-prb/cu-h6-enumeraties.git>

3 ARRAYS

3.1 WAT ZIJN ARRAYS?

Arrays worden soms ook gegevensvelden of tabellen genoemd.

Aan een **variabele** kan je altijd maar **één waarde tegelijk** toewijzen. De variabele *month* kan bv. enkel de naam van één maand bevatten.

Een **array** daarentegen kan verschillende waarden bevatten. Men spreekt daarbij van **verschillende elementen**, waarbij elk element een waarde heeft. De afzonderlijke elementen gedragen zich bijgevolg als variabelen. In plaats van tien variabelen kan je dus ook een array van tien elementen gebruiken. De array *monthsOfTheYear* zou dus twaalf variabelen van het type *string* kunnen bevatten.

De elementen van een Array behoren allen tot hetzelfde type. Zo kan je hebben:

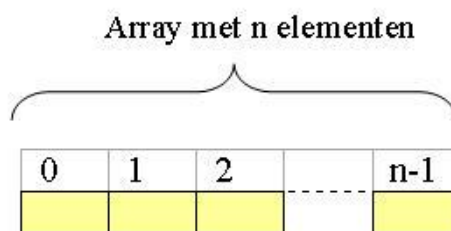
- een array van *ints*
- een array van *strings*
- een array van *Buttons*
- een array van arrays (die op hun beurt van het type *bool* zijn)
- ...

Arrays hebben nog een groot voordeel: de verschillende elementen worden doorlopend genummerd (index), waarbij het **eerste element altijd de index 0** krijgt. Als programmeur kunnen we deze nummering niet aanpassen, zoals bij enums wel het geval is.

Via deze index kun je waarden ophalen of bepalen:

```
string[] month = new string[12];  
month[0] = "januari"; //waarde toekennen  
MessageBox.Show(string.Format("De eerste maand is {0} ", month[0])); //waarde ophalen  
-- of --  
MessageBox.Show($"De eerste maand is {month[0]} "); //waarde ophalen
```

Met een lus (zie volgende hoofdstuk) kan je de elementen van de array doorlopen.



Een Array met n elementen: de elementen hebben een index 0 tot n-1. De indices van een array met 10 elementen lopen dus van 0 tot en met 9.



Verschil tussen arrays en enums

- Bij **enums** gaan we de waardes van de enum niet veranderen, maar gaan we deze **enkel opvragen**.

- Bij **arrays** kunnen we elk element gaan **opvragen en wijzigen**, zolang we binnen **hetzelfde datatype** blijven

3.2 EEN ARRAY DECLAREREN

```
int[] ages;
```

Hier werd een array-variabele met de naam **ages** gedeclareerd; de elementen zullen van het type **int** zijn. De **vierkante haken** na het variabele-type geven dus aan dat de variabele een array van elementen van het gekozen type (in het voorbeeld **int**) zal zijn.

3.3 EEN ARRAY INSTANTIËREN

Een array is een referentietype: een arrayvariabele is dus een variabele die op de stack wordt bewaard en daar enkel het geheugenadres bevat naar het geheugenblok dat op de heap de eigenlijke elementen van de array bevat (zie hoofdstuk over 'value en reference').

Bij de instantiëring van een array wordt aangegeven hoeveel elementen de array zal bevatten. De elementen kunnen nu ook opgevuld worden. Deze elementen moeten van het type zijn dat je hebt aangegeven bij declaratie.

We kunnen op verschillende manieren instantiëren.

3.3.1 DECLAREREN, DE ELEMENTEN AFZONDERLIJK INSTANTIËREN

```
int[] ages = new int[4];  
//De array met de naam ages zal 4 elementen bevatten van het type int.  
//Deze elementen worden genummerd 0 - 3  
  
ages[0] = 18;  
ages[1] = 63;  
ages[2] = 42;  
ages[3] = 7;
```

Opmerking: het aantal elementen kan ook de waarde van een **int**-variabele zijn:

```
int numberOfElements = 4;  
int[] ages = new int[numberOfElements];
```

3.3.2 VERKORTE SCHRIJFWIJZE: DIRECT INSTANTIËREN

Als je al onmiddellijk de beginwaarden van de elementen kent, kan je in C# een verkorte schrijfwijze gebruiken door de elementen tussen accolades te noteren:

```
int[] ages = new int[4] { 18, 63, 42, 7 };
```

3.3.3 NOG KORTER

Het is zelfs toegestaan om het sleutelwoord `new` gevolgd door het type en het aantal elementen gewoon weg te laten:

```
int[] ages = { 18, 63, 42, 7 };
```

3.4 TOEPASSING

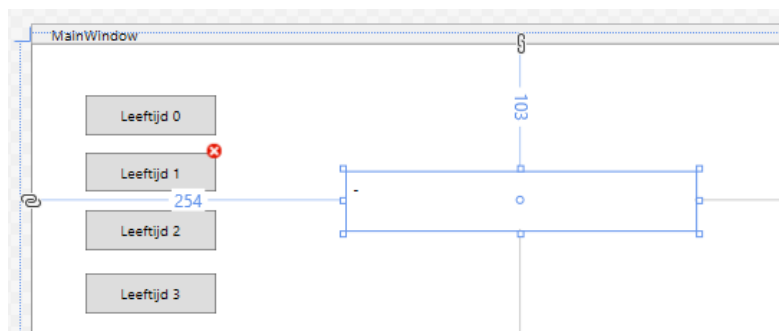
Maak een nieuwe solution aan met daarin een WPF (.Core) project.

- Maak een event aan wanneer `MainWindow` geladen is
- In dit event declareren we een array met leeftijden
- In dit event tonen we een `MessageBox` die de derde waarde/leeftijd uit onze array `ages` zal tonen.

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        int[] ages = { 18, 63, 42, 7 };
        MessageBox.Show("De derde leeftijd is: " + ages[2].ToString());
    }
}
```

Plaats 4 buttons op je venster (`btnAge0`, `btnAge1`, `btnAge2` en `btnAge3`) en een label (`lblAge`):



Wanneer op `btnAge0` wordt geklikt, verschijnt in `lblAge` de melding “Leeftijd met index 0 = 18”, `btnAge1` de melding “Leeftijd met index 1 = 63”, enz.

De code om de melding af te beelden, stoppen we in een afzonderlijke methode. Omdat we onze array nu op verschillende plaatsen zullen nodig hebben, gaan we de declaratie globaal maken:



We maken een methode aan om het bericht in het label te plaatsen:

```
void ShowAge(int index)
{
    lblAge.Content = $"Leeftijd met index {index} = {ages[index]}";
}
```

En in de 4 afzonderlijke click event-handlers van onze knoppen roepen we deze methode dan telkens op:

```
private void BtnAge0_Click(object sender, RoutedEventArgs e)
{
    ShowAge(0);
}

private void BtnAge1_Click(object sender, RoutedEventArgs e)
{
    ShowAge(1);
}

private void BtnAge2_Click(object sender, RoutedEventArgs e)
{
    ShowAge(2);
}

private void BtnAge3_Click(object sender, RoutedEventArgs e)
{
    ShowAge(3);
}
```

3.5 DE KLASSE ARRAY

`System.Array` is een klasse binnen .Net.

Deze klasse bevat een aantal methoden (`Copy`, `Reverse`, ...) en eigenschappen (`Length`, ...), die we bij Enums niet vinden.

Arrays hebben echter ook een **belangrijk nadeel**: ze zijn fixed-length: om elementen toe te voegen of te verwijderen, ben je dus verplicht een nieuwe `Array` te maken!

Het alternatief is om een array aan te maken die zeker genoeg elementen bevat, maar dan ga je niet zuinig om met het geheugen van de computer.

Arrays zijn dus pas echt interessant als het aantal elementen in een reeks nooit of heel zelden verandert. Als oplossing hiervoor kunnen we Collections zoals lijsten gebruiken, die verder in dit hoofdstuk worden toegelicht.

Plaats nog een button (btnSailors) en een label (lblSailors) op je venster. Maak voor deze button een click event-handler aan en neen onderstaande code over:

```
private void btnKaaprenVaarders_Click(object sender, RoutedEventArgs e)
{
    string[] sailors = { "Jan", "Piet", "Joris" };
    string[] sailorsCopy = new string[3];
    sailors.CopyTo(sailorsCopy, 0);
    sailors = new string[4];
    sailorsCopy.CopyTo(sailors, 0);
    sailors[3] = "Korneel";
    string sailorText = sailors[0] + ", " + sailors[1] + ", " + sailors[2] + " en " +
sailors[3] + "\nDie hebben baarden, zij varen mee.";

    MessageBox.Show(sailorText);
}
```

Laten we even elke instructie afzonderlijk bekijken :

- `string[] sailors = { "Jan", "Piet", "Joris" };`
We maken een array van strings aan en we vullen deze meteen met 3 waarden: denk er aan, omdat we onze array op deze manier declareren (en initialiseren) bestaat deze array uit 3 elementen, waar we per definitie nu niets meer kunnen aan toevoegen.
- `string[] sailorsCopy = new string[3];`
We maken een nieuwe array van strings aan. Deze keer vullen we hem niet onmiddellijk met waarden maar geven wel aan dat deze array ook uit exact 3 elementen zal bestaan.
- `sailors.CopyTo(sailorsCopy, 0);`
We gebruiken de `CopyTo` methode van de `Array` klasse om de inhoud van de ene array (`sailors`) te kopiëren naar de andere array (`sailorsCopy`). Het 1° argument van deze methode is dus de naam van de doel-array, het 2° argument geeft aan vanaf welke plaats in de doel-array er moet geschreven worden (hier dus vanaf het eerste element).
- `sailors = new string[4];`
We herdefiniëren ("her-instantiëren") de array `sailors`: hierdoor gaat de huidige inhoud van deze array verloren.
- `sailorsCopy.CopyTo(sailors, 0);`
We kopiëren terug de inhoud van `sailorsCopy` naar `sailors`: denk er aan, het 4° (en laatste / met index 3) element van `sailors` zal hier niet gevuld worden, we kopiëren immers maar 3 waarden.
- `sailors[3] = "Korneel";`
We vullen het laatste (index 3) element met een waarde
- `string sailorText = sailors[0] + ", " + sailors[1] + ", " + sailors[2] + " en " + sailors[3] + "\nDie hebben baarden, zij varen mee.";`
we concateneren de verschillende waarden van de array tot 1 string.
- `MessageBox.Show(sailorText);`
En we beelden deze string af in een messagebox.

3.6 ARRAYS MET MEERDERE DIMENSIES

Tot nu toe hadden we het over arrays met 1 dimensie. Stel dat we de dagen van de week wensen bij te houden dan zouden we iets als volgt kunnen maken:

```
string[] daysOfTheWeek = new string[7];
daysOfTheWeek[0] = "maandag";
daysOfTheWeek[1] = "dinsdag";
daysOfTheWeek[2] = "woensdag";
daysOfTheWeek[3] = "donderdag";
daysOfTheWeek[4] = "vrijdag";
daysOfTheWeek[5] = "zaterdag";
daysOfTheWeek[6] = "zondag";
```

Stel nu dat we ook zouden willen bijhouden wanneer we elke dag beginnen en eindigen met werken. Een oplossing zou kunnen zijn om hiervoor nog 2 extra arrays aan te maken

```
string[] startHours = new string[7];
string[] stopHours = new string[7];
startHours[0] = "09:00";
stopHours[0] = "18:00";
startHours[1] = "08:00";
stopHours[1] = "19:00";
...
```

We kunnen dit echter anders oplossen. We kunnen immers arrays maken met meerdere dimensies (in theorie tot 32, in de praktijk doorgaans nooit meer dan 3).

Ter info :

- Een 1-dimensionele array kan je vergelijken met een kolom uit een spreadsheet
- Een 2-dimensionele array kan je vergelijken met een werkblad uit een spreadsheet (een tabel)
- Een 3-dimensionele array kan je vergelijken met een verzameling van werkbladen in een spreadsheet

Onze 2-dimensionele array (werkdagen, startuur en einduur) gaan wij als volgt declareren :

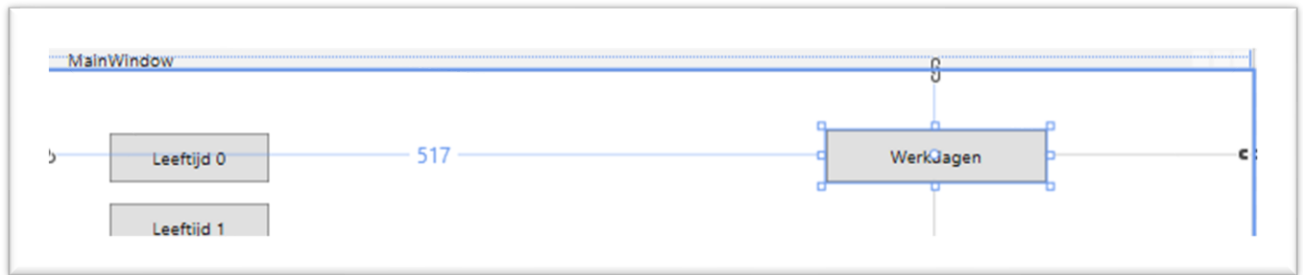
```
string[,] daysOfTheWeek = new string[7,3];
```

Opnieuw, stel je een 2-dimensionele array voor als een tabel. `string[7,3]` geeft aan dat wij een “tabel” maken van 7 rijen en 3 kolommen (2 dimensies dus).

Ter info : stel dat we een 3 dimensionele array wensen aan te maken dan zou onze declaratie er bijvoorbeeld als volgt moeten uitzien:

```
int[,,,] manyNumbers = new int[20,10,4];
```

Plaats op je venster nog een laatste knop : btnWorkingDays



Maak de click event-handler voor deze knop aan en neem onderstaande code over :

```
string[,] daysOfTheWeek = new string[7,3];
daysOfTheWeek[0,0] = "maandag";
daysOfTheWeek[1,0] = "dinsdag";
daysOfTheWeek[2,0] = "woensdag";
daysOfTheWeek[3,0] = "donderdag";
daysOfTheWeek[4,0] = "vrijdag";
daysOfTheWeek[5,0] = "zaterdag";
daysOfTheWeek[6,0] = "zondag";

daysOfTheWeek[0, 1] = "09:00";
daysOfTheWeek[1, 1] = "08:00";
daysOfTheWeek[2, 1] = "08:00";
daysOfTheWeek[3, 1] = "08:00";
daysOfTheWeek[4, 1] = "08:00";
daysOfTheWeek[5, 1] = "";
daysOfTheWeek[6, 1] = "";

daysOfTheWeek[0, 2] = "18:00";
daysOfTheWeek[1, 2] = "17:00";
daysOfTheWeek[2, 2] = "12:00";
daysOfTheWeek[3, 2] = "17:00";
daysOfTheWeek[4, 2] = "16:00";
daysOfTheWeek[5, 2] = "";
daysOfTheWeek[6, 2] = "";
```

Vervolgens zouden wij in onze event-handler nu ook de inhoud van deze 2-dimensionele array willen afbeelden. Om dit te doen gebruiken we lussen (iets wat we pas volgend hoofdstuk bespreken). Neem onderstaande code over onderaan je event-handler :

```
string text = "";
for (int row = 0; row < 7; row++)
{
    text += $"Op {daysOfTheWeek[row, 0]} werk ik "
        + $"van {daysOfTheWeek[row, 1]} "
        + $"tot {daysOfTheWeek[row, 2]} \n";
}
MessageBox.Show(text);
```

Wat we hierboven precies doen bespreken we dus in een volgend hoofdstuk. Het mag echter duidelijk zijn dat arrays (en collections) hand in hand zullen gaan met lussen (iteraties).



De repository met de code rond arrays uit dit hoofdstuk kan je hier terugvinden :

git clone <https://github.com/howest-gp-prb/cu-h6-arrays.git>

4 COLLECTION KLASSEN

.Net bevat naast Arrays nog andere klassen om reeksen elementen te hanteren. We noemen ze de Collection klassen. Je vindt ze in de namespace `System.Collections.Generic`.

Om ze te gebruiken, dien je de namespace te vermelden in de code of (bij voorkeur) een `using` statement te voorzien bovenaan.

```
using System.Collections.Generic;
...
List<string> persons;
```

Indien we geen gebruik maken een `using` statement moeten we onze declaratie voluit schrijven:

```
System.Collections.Generic.List<string> persons;
```

In de volgende onderdelen van deze cursus geven we wat uitleg over enkele van deze collections.

4.1 LIST



Lists zal je doorheen je opleiding voortdurend gaan gebruiken. Hier krijg je slechts een korte inleiding.

Een oplossing voor het probleem met de vaste lengte van arrays en enums vinden bij de List-klasse.

Een list bevat namelijk de methode `Add`.

Maak een nieuwe solution aan met daarin 1 WPF project (.Net Core).

Plaats op je venster 1 knop "btnSailors" en maak de click event-handler voor deze knop aan.

In deze event-handler neem je onderstaande code over:

```
private void btnKaaprenVaarders_Click(object sender, RoutedEventArgs e)
{
    string sailorText;
    List<string> sailors = new List<string>();
    sailors.Add("Jan");
    sailors.Add("Piet");
    sailors.Add("Joris");
    sailors.Add("Korneel");

    sailorText = sailors[0] + ", " + sailors[1] + ", " + sailors[2] + " en " + sailors[3]
        + "\nDie hebben baarden, zij varen mee.";
    MessageBox.Show(sailorText, "Niet gesorteerd");

    sailors.Sort();
    sailorText = sailors[0] + ", " + sailors[1] + ", " + sailors[2] + " en " + sailors[3]
        + "\nDie hebben baarden, zij varen mee.";
    MessageBox.Show(sailorText, "Gesorteerd");
}
```

In het voorbeeld hierboven zien we hoe eenvoudig het is om elementen aan een list toe te voegen. Dit lijkt zeer sterk op de manier om items aan een list- of combobox toe te voegen.

Een ander voordeel van lists is dat je er meer methoden op kan toepassen zoals `Remove`, `Contains`, `Find`, `Insert`, `Reverse`, `Sort` en nog véél andere methodes.

Nu nog kort even de code hierboven toelichten:

- `List<string> sailors = new List<string>();`
 - `List<string>` geeft aan dat we een nieuwe List wensen aan te maken en dat we hierin waarden van het type `string` in willen bewaren.
 - `sailors` is uiteraard de naam van je nieuwe List
 - `new List<string>()` net zoals bij de array dien je een instantie aan te maken
- `sailors.Add("Jan");`
`sailors.Add("Piet");`
`sailors.Add("Joris");`
`sailors.Add("Korneel");`

M.b.v. de `Add` methode kan je elementen toevoegen aan een list. Zoals gemeld is er bij een List geen bovengrens: je kan zoveel elementen als nodig toevoegen aan een List.

- `sailorText = sailors[0] + ", " + sailors[1] + ", " + sailors[2] ...`

Elk element uit de List kan je benaderen via zijn index. Dit gebeurt op dezelfde manier als bij arrays (eerste element = 0).

Indien we niet hadden geweten uit hoeveel elementen onze List bestond, dan hadden we dit kunnen opvragen via de `Count`-eigenschap. Bv: `int numberOfElements = sailors.Count;`

- `sailors.Sort();`

M.b.v. de `Sort` methode kan je de elementen binnen een list sorteren (hier dus alfabetisch)



Meer informatie over de List klasse (met alle methoden, properties ...)

- [List<T> klasse](#)

Plaats nog een listbox op je venster (`lstSailors`) en een label (`lblPosition`) en maak de `SelectionChanged` event-handler aan voor de listbox.

Voeg als laatste instructie in de click event-handler van je button onderstaande toe :

```
lstSailors.ItemsSource = sailors;
```

Een `ListBox` in je WPF applicatie is eigenlijk niets anders dan een grafische weergave van een collection. M.b.v. de `ItemSource` eigenschap kunnen we probleemloos een collection (onze List) koppelen aan de betrokken `ListBox`.

Omdat onze List uit strings bestaat is het geen enkel probleem om deze onmiddellijk aan de ListBox te koppelen: wanneer we later onze eigen objecten zullen willen afbeelden in een ListBox zullen we trouwens steeds aandacht moeten besteden aan wat we precies willen afbeelden in onze ListBox.

Probeer je code even uit:

Tenslotte willen we, wanneer we een item selecteren in onze listbox in onze List (sailors) gaan opzoeken op de hoeveelste plaats het element gevonden werd.

Het probleem is terug dat we onze List (sailors) aangemaakt hebben binnen de Click event-handler van onze knop waardoor de scope van onze List dus beperkt is tot deze event-handler.

Eerst en vooral gaan we dus het bereik van onze List groter maken door hem globaal aan te maken.

Pas dus eerst je code als volgt aan:



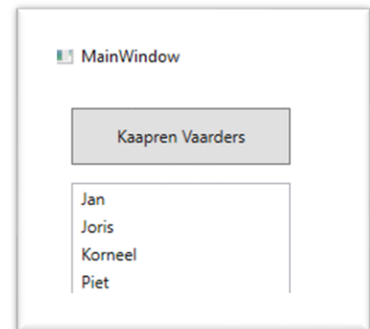
```
16 public MainWindow()  
17 {  
18     InitializeComponent();  
19 }  
20  
21 private void BtnSailors_Click(object sender, RoutedEventArgs e)  
22 {  
23     string sailorText;  
24     List<string> sailors = new List<string>();  
25     sailors.Add("Jan");  
26     sailors.Add("Piet");  
27     sailors.Add("Joris");  
28     sailors.Add("Korneel");
```

Dit wordt:



```
13 List<string> sailors;  
14  
15 public MainWindow()  
16 {  
17     InitializeComponent();  
18 }  
19  
20 private void BtnSailors_Click(object sender, RoutedEventArgs e)  
21 {  
22     string sailorText;  
23     sailors = new List<string>();  
24     sailors.Add("Jan");  
25     sailors.Add("Piet");  
26     sailors.Add("Joris");  
27     sailors.Add("Korneel");
```

Test eerst even je code uit, maar alles zou netjes moeten blijven werken zoals voordien.



Neem nu in de SelectionChanged event-handler van lstSailors onderstaande code over :

```
private void LstSailors_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    lblPosition.Content = "";
    if(lstSailors.SelectedItem != null)
    {
        string selectedSailor = (string)lstSailors.SelectedItem;
        int position = sailors.IndexOf(selectedSailor);
        lblPosition.Content = $"{selectedSailor} gevonden op positie {position}";
    }
}
```

Toelichting:

- We maken lblPosition leeg.
- We kijken even na of er wel degelijk iets geselecteerd werd in onze listbox.
Je kan dit op een aantal manieren doen, dit is er een van.
We hadden bv ook kunnen schrijven :

```
if(lstSailors.SelectedIndex > -1)
```

- Vervolgens vullen we de variabele selectedSailor met de waarde die geselecteerd werd in de listbox. **Opgepast:** SelectedItem is van het type object. We moeten hier verplicht deze waarde casten naar het type string. Uiteraard hadden we ook kunnen schrijven:
lstSailors.SelectedItem.ToString()
- Met de IndexOf methode zoeken we de plaats op waar de waarde zich in onze originele List bevindt.
- En tenslotte geven we in lblPosition de nodige feedback.

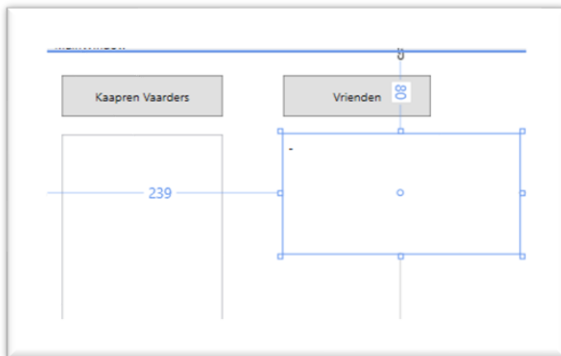
Hier gebruikten we dus een van de standaard voorziene methodes van een List: IndexOf om de index van een element in de lijst op te zoeken. Zo zijn er nog talloze andere methoden die goed van pas zullen komen.

Merk op dat we hier eigenlijk niet eens zouden moeten zoeken naar de index van het element in de lijst sailors, aangezien die sowieso zal overeenkomen met de geselecteerde index in de listbox lstSailors. We konden de index dus ook gewoon opvragen via lstSailors.SelectedIndex. Bovenstaande is een demonstratie van de methode IndexOf, die in andere situaties zeker nog van pas zal komen.

4.2 DICTIONARY

Een Dictionary is een collection waarin paren van key/value (sleutel/waarde) bewaard kunnen worden. In tegenstelling tot andere collections kan een element van een Dictionary niet alleen opgezocht worden aan de hand van een index (int), maar ook (en voornamelijk!) van een sleutel (waarvan je het type zelf kan kiezen). Je kan dit een beetje vergelijken met de primary key van een tabel in een relationele database.

Plaats op je venster nog een knop btnFriends en een label lblFriends. Neem in de Click event-handler van de knop onderstaande code over :



```
private void BtnFriends_Click(object sender, RoutedEventArgs e)
{
    Dictionary<string, int> friends = new Dictionary<string, int>();

    friends.Add("Emmanuel", 40);
    friends.Add("Angela", 42);
    friends.Add("Donald", 25);

    string output = "";
    output += "De leeftijd van Emmanuel is " + friends["Emmanuel"] + "\n";
    output += "De tweede vriend(in) in de rij is " + friends.Keys.ElementAt(1) ;
    lblFriends.Content = output;
}
```

Om de methode ElementAt te kunnen gebruiken, moet je bovenaan using System.Linq; toevoegen. Je zal later meer leren over Linq, o.a. in de module Programming Expert.

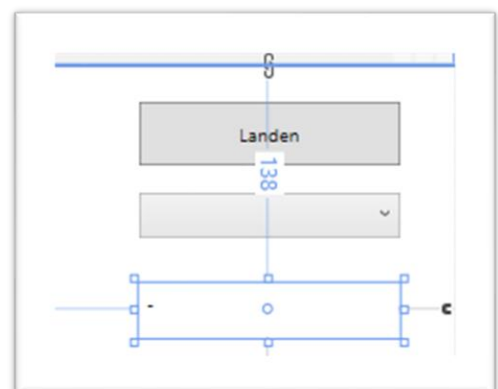
De dictionary is zeer handig als je snel zaken wil opzoeken a.d.h.v. een sleutel van een zelf te kiezen type (bv. string in het voorbeeld hierboven: de naam van de vriend).

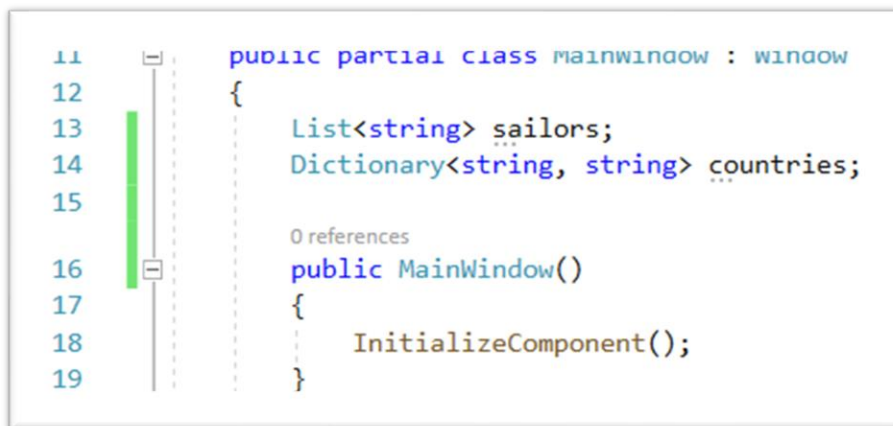
Nog een laatste voorbeeld dat wat praktischer is dan het voorbeeld van onze vrienden.

Plaats nog een button (btnCountries + click event-handler), een combobox (cmbCountries + SelectionChanged event-handler) en een label (lblCapital) op je venster:

Boven de constructor van je MainWindow, declareer je eerst een dictionary countries:

```
Dictionary<string, string> countries;
```





In de event-handler zelf neem je onderstaande code over:

```
private void BtnCountries_Click(object sender, RoutedEventArgs e)
{
    countries = new Dictionary<string, string>();

    countries.Add("Belgium", "Brussels");
    countries.Add("Germany", "Berlin");
    countries.Add("France", "Paris");
    countries.Add("Spain", "Madrid");
    countries.Add("Italy", "Rome");
    countries.Add("Norway", "Oslo");
    countries.Add("UK", "London");
    countries.Add("Ireland", "Dublin");

    cmbCountries.ItemsSource = countries.Keys;
}
```

We maken een instantie aan van de dictionary countries en voegen vervolgens de landen (= keys) en hun hoofdsteden toe (= values). **Belangrijk:** de keys MOETEN uniek zijn, values hoeven niet uniek te zijn.

Tenslotte vullen we de ItemSource property van onze combobox niet met onze volledige dictionary, maar enkel met de keys uit onze dictionary (de landen).


In de SelectionChanged event-handler van onze combobox neem je onderstaande code over :

```
private void CmbCountries_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    lblCapital.Content = "";
    if(cmbCountries.SelectedItem != null)
    {
        string selectedCountry = (string)cmbCountries.SelectedItem;
        lblCapital.Content = countries[selectedCountry];
    }
}
```



Code repository

De volledige broncode van dit voorbeeld is te vinden op

 git clone <https://github.com/howest-gp-prb/cu-h6-collections.git>

Er bestaan in C# ook nog andere collections zoals een *Stack* of *Queue*. Later in je opleiding zal je hiermee misschien nog in aanraking komen. Voorlopig houden we het op List en Dictionary, waarvan we voornamelijk de Lists heel uitvoerig gaan gebruiken.

5 SAMENVATTING

	Index	Lengte	Waarden	Referentie
<i>Enum</i>	Aanpasbaar	Vast	Vast	Naam van de waarde Index
<i>Array</i>	Vast	Vast	Variabel	Index
<i>List</i>	Vast	Variabel	Variabel	Index
<i>Dictionary</i>	Vast	Variabel	Variabel	Index + veldnaam (sleutel)

