**Agenda**

* **De Taal C#**
  + Grondbeginselen van C#
    - Generics
      * Constraints
      * Covariance, Contravariance & Invariance
  + Collections in C#
    - Allocatie van collecties
    - Arrays, Two Dimensional Arrays & Jagged Arrays
    - IEnumerable, ICollection, IList
    - SortedList
    - Stack & Queue
    - Dictionary & Hashset

# Generics

Generics zijn bij C# versie 2.0 toegevoegd. Generics maken het mogelijk om type parameters mee te geven aan klassen, methodes en interfaces. Door het gebruik van generics kan runtime casting of boxing worden voorkomen.

Waarom zou je generics gebruiken?

* Code reuse maximaliseren
* Type Safety (Compile time)
* Performance

Generics worden voornamelijk in collectie klassen gebruikt, maar je kan ze overal tegenkomen.

* Generic Type Parameters
* Generic Classes
* Generic Interfaces
* Generic Methods
* Generic Delegates

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>

## Constraints

Bij het declareren van generics kunnen er constraints worden gebruikt om aan te geven waar de opgegeven generics aan zullen moeten voldoen. De volgende constraints kunnen worden meegegeven aan generic definities:

* **where T: struct**

Generic T moet van het type struct zijn

* **where T : class**

Generic T moet van het type class zijn

* **where T : new()**

Generic T moet een parameterloze constructor hebben. De ‘new()’ constraint moet altijd als laatste worden gedefineerd.

* **where T : <base class name>**

Generic T moet erven van de opgegeven base class

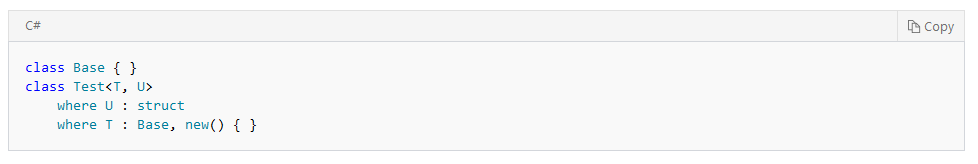
* **where T : <interface name>**

Generic T moet het opgegeven interface implementeren

* **where T : U**

Generic T moet type of erven van het type opgegeven door U.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/constraints-on-type-parameters>



## Covariance, Contravariance & Invariance

Specificatie mag alleen op interfaces & delegates.

* ***Covariance (out keyword)***

Staat je toe om een meer derived type te gebruiken.

IEnumerable<Derived> mag worden meegegeven aan type IEnumerable<Base>

* ***Contravariance(in keyword)***

Staat je toe om een type mee te geven die minder derived is dan het opgegeven type.

IEnumerable<Base> mag worden meegegeven aan type IEnumerable<Derived>

* ***Invariance (standaard)***

Staat je toe om alleen het opgegeven type mee te geven.

https://stackoverflow.com/questions/2662369/covariance-and-contravariance-real-world-example

<https://docs.microsoft.com/en-us/dotnet/standard/generics/covariance-and-contravariance>

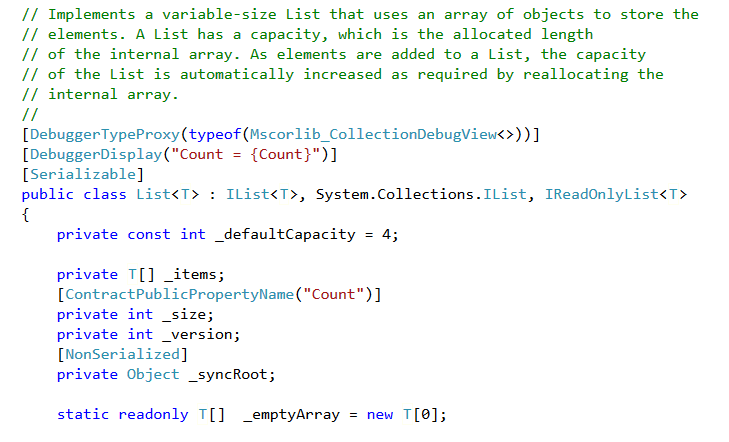
# Collections in C#

Voor veel doelen wil je in C# gebruik maken van verzamelingen van objecten. In c# kan je onderscheid maken tussen arrays en collecties. Arrays kan je het beste gebruiken als je een vast aantal – strongly typed objecten hebt. Arrays bieden snelle access maar moeten wel met een vaste lengte gedeclareerd worden.

Collecties bieden een veel flexibelere manier om met verzamelingen van objecten te werken. Een collectie in C# kan dynamisch groeien en krimpen, en de .Net framework class library bied veel functionaliteit voor de collecties in C#.

Collecties zijn klassen, en moeten dus met ‘ new ‘ worden geinitialiseerd.

Eigenlijk heeft het geen zin om na te denken over de implementatie (want dat regelt het framework voor je) maar onderwater bestaan de meeste collecties uit arrays:



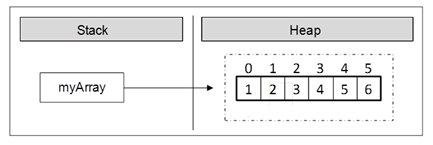
<https://referencesource.microsoft.com/#mscorlib/system/collections/generic/list.cs>

## Allocatie van collecties

Collecties en arrays worden over het algemeen opgeslagen net als reference types. Dit houd in dat de variabele met het adres op de stack komt, en de daadwerkelijke collectie op de heap.

## 

## int[] myArray = { 1, 2, 3, 4, 5, 6 };

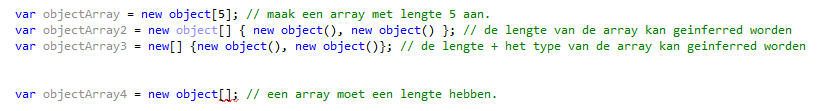


## Arrays, Two dimensional arrays & Jagged Arrays

Binnen C# zijn de volgende type arrays te gebruiken:

* ***Single***

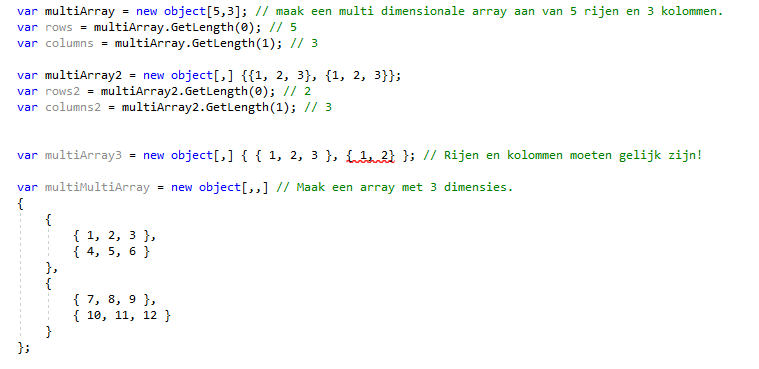
Single arrays zijn de meest voorkomende arrays. Bij arrays moet je altijd de lengte specifiëren tijdens de declaratie:



<https://msdn.microsoft.com/nl-nl/library/9b9dty7d(v=vs.120).aspx>

* ***Multi dimensionaal***

Multi dimensionale array zijn arrays welke over ‘ rijen en kolommen ‘ beschikken. De lengte van de rijen en kolommen moet overeenkomen. Een multi dimensionale array geef je aan met een ‘ ,’ in de array declaratie:

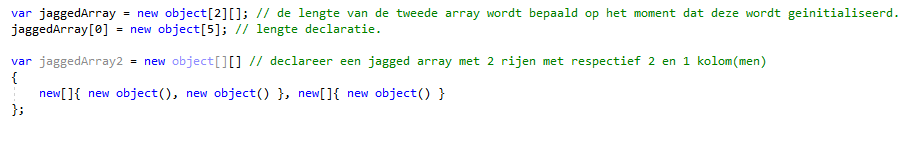


Je kan zo veel dimensies toevoegen aan de array als je zelf wilt.

<https://msdn.microsoft.com/nl-nl/library/2yd9wwz4(v=vs.120).aspx>

* ***Jagged***

Een jagged array is een ‘ array van arrays ‘. In tegenstelling tot de multi dimensionale array kan een jagged array dus per array een variabele array bevatten. Dit houd in dat de rijen en kolommen (en dan voornamelijk de kolommen) niet overeen hoeven te komen.



<https://msdn.microsoft.com/nl-nl/library/2s05feca(v=vs.120).aspx>

https://stackoverflow.com/questions/597720/what-are-the-differences-between-a-multidimensional-array-and-an-array-of-arrays

## IEnumerable vs ICollection Interface vs IList

<https://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx>

Als je binnen het .Net framework je eigen collectie wilt implementeren wordt er vaak gebruik gemaakt van de volgende interfaces:

* ***IEnumerable***



Voor een lijst waardoor je moet itereren. De Enumerator verzorgt het itereren door de collectie.

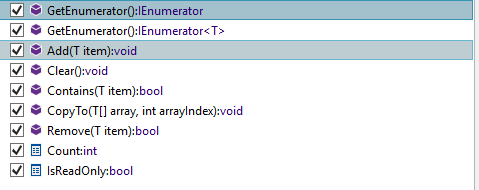
Als je dit interface implementeert moet je de volgende members implementeren:



* ***ICollection***

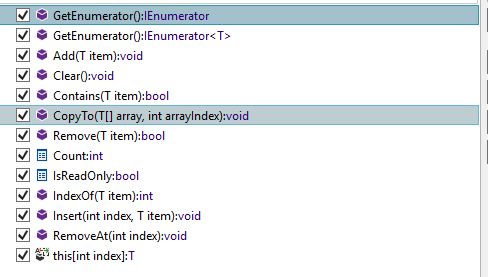


Voor een lijst waardoor je moet kunnen itereren, en welke aanpasbaar moet zijn. Als je dit interface implementeert moet je de volgende members implementeren:



* ***IList*** 

Voor een lijst waardoor je moet enumeraten, welke aanpasbaar moet zijn en welke sorteerbaar moet zijn. Als je dit interface implementeerd moet je de volgende members implementeren:

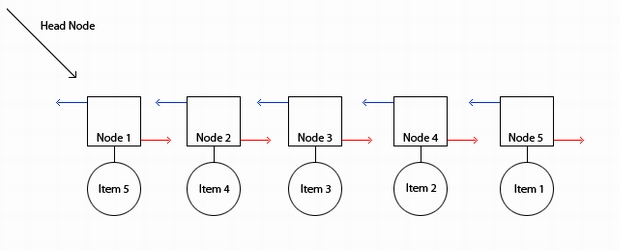


<https://stackoverflow.com/questions/271710/collectiont-versus-listt-what-should-you-use-on-your-interfaces>

## LinkedList<T>

* ***LinkedList***

Over het algemeen is een linked list een lijst waarbij elk element in de list een referentie heeft naar de volgende node. Omdat de C# implementatie van de LinkedList<T> een doubly linked list is, bevat het naast een referentie naar de volgende node ook een referentie naar de vorige node.



De LinkedList<T> bestaat uit LinkedListNode<T>’s. De nodes bevatten de waardes van het gekoppelde object, een referentie naar de volgende en vorige node en een referentie naar de lijst waarin ze zijn opgeslagen.

## Stack<T> & Queue<T>

* ***Stack***

De Stack<T> is een standaard .Net framework klasse welke opereert zoals ook de stack van de CLR ‘ werkt ‘. Dit betekend dat er op de stack waardes geplaatst kunnen worden met .Push() en waardes van bovenaf van de stack kunnen worden gehaald met .Pop() De bovenste waarde van de stack kan worden bekeken met .Peek()

Omdat er altijd met de bovenste waarde van de stack wordt gewerkt hanteerd de stack een **LIFO** volgorde. Dit betekend **Last In First Out.**

* ***Queue***

De Queue<T> in het .Net framework wordt gebruikt zoals je je van een ‘rij’ kan voorstellen. Middels de functie .Enqueue() kan er een item aan het einde van de queue worden toegevoegd. Met .Dequeue kan het item dat vooraan de rij staat uit de collectie worden gehaald. Met .Peek() kan het item vooraan in de rij worden bekeken. In tegenstelling tot de stack hanteert de queue dus een **FIFO** volgorder – **First In First Out.**

## Dictionary<Tkey, TValue> & HashSet<T>

Dictionaries en Hashsets worden vaak gebruikt omdat het indexen van een hashset een o(1) operatie is. Dit houd in dat de operatie om een element te vinden binnen een Hashset vele malen sneller is dan bijv. bij een gewonen List<T> waar het vinden van een element vaak een o(n) operatie is. (of door .net optimalisaties o (log n).

Een hashset werkt door gebruikt te maken van de .GetHashCode op het object. Alle objecten implementeren de .GetHashCode functie.



De hashcode wordt vervolgens gebruikt om de waarde in ‘buckets’ op te slaan. Omdat de waardes in de hashset / dictionary uniek zijn, is het vinden van de index na het verkrijgen van de hash een enkele operatie.

