Enkele sollicitatievragen

Wat is het verschil tussen .NET Core en een .NET Framework ?

https://www.mijnhostingpartner.nl/blog/tips/het-verschil-tussen-net-core-en-een-net-framework/

https://dotnettutorials.net/course/asp-net-core-tutorials/

Wat is Inversion Of Control IoC?

https://www.tutorialsteacher.com/ioc/introduction

https://www.tutorialsteacher.com/ioc/inversion-of-control

Wat is Depencency Injection?

https://www.tutorialsteacher.com/ioc/dependency-injection

Hoe werkt de ASP.NET Core Dependency Injection (Core3.x)

https://dotnettutorials.net/lesson/asp-net-core-dependency-injection/

Welke 2 belangrijke methoden vind je in Startup.cs in een ASP.NET Core MVC Web of Api app en waarvoor dienen deze?

https://www.tutorialsteacher.com/core/aspnet-core-startup

Wat is de IoC built-in container in ASP.NET Core?

https://www.tutorialsteacher.com/core/internals-of-builtin-ioc-container-in-aspnet-core

Hoe werkt de Request-pipeline in ASP.NET Core?

https://dotnettutorials.net/lesson/asp-net-core-request-processing-pipeline/

https://dotnettutorials.net/lesson/asp-net-core-middleware-components/

Wat is Kestrel?

https://dotnettutorials.net/lesson/kestrel-web-server-asp-net-core/

Wat is het verschil tussen InProcess en OutOfProcess Hosting?

https://dotnettutorials.net/lesson/asp-net-core-inprocess-hosting/

https://dotnettutorials.net/lesson/asp-net-core-outofprocess-hosting/

Wat is een extension method?

https://www.tutorialsteacher.com/csharp/csharp-extension-method

Wat is het verschil tussen Value en reference types?

In .NET hebben we categorieën van datatypes. Enerzijds de *value types*, anderzijds de *reference types*.

Voorbeelden van value types
zijn `int`, `double`, `decimal`, `char`, `bool`, `DateTime` en alle enumatietypes.
Deze zijn allen met een *structure* (`struct`) gedefinieerd.

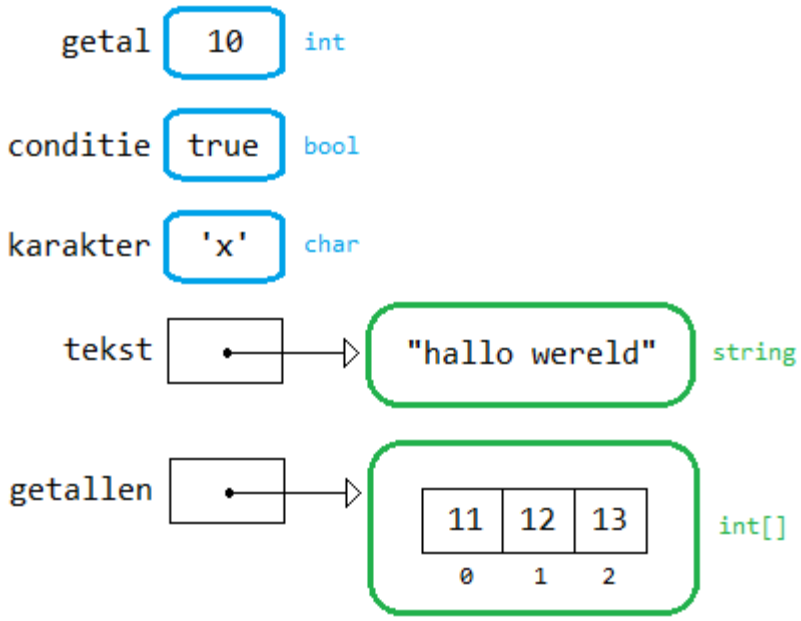
Voorbeelden van reference types zijn `string`, `object`, `Array` en alle afgeleide tabeltypes als `int[]`, `bool[,]`, `string[,,]`.
Of om het eenvoudig te maken, alle datatypes die gedefinieerd zijn aan de hand van een klasse (`class`).

```
using System;
class ValueEnReferenceTypesVoorbeeld
{
    static void Main()
    {
        //Value types:
        int getal = 10;
        bool conditie = true;
        char karakter = 'x';

        //Reference types:
        string tekst = "hallo wereld";
        int[] getallen = { 11, 21, 31 };
    }
}
```

C# broncode voorbeeld 183

Een belangrijk verschil is dat dataholders, variabelen bijvoorbeeld, van een value type rechtstreeks geassocieerd zijn met de instantie.
Dataholders van een reference type bevatten dan eerder een verwijzing, een *referentie*, naar de instantie.
Je zou het als volgt kunnen visualiseren...



Afbeelding 7

De instanties zijn hier in een kleur weergegeven. Blauw voor value type instanties. Groen voor objecten van een reference type.
De directe inhoud van variabelen als `tekst` en `getallen` is de verwijzing naar de plaats in het geheugen waar de reference type instantie zich bevindt.

Value type instanties zitten op de *stack* gealloceerd.
Dit kan je bekijken als een eenvoudig register waarin de dataholder in scope worden bijgehouden, met daarbij hun datatype en de éénen en nullen die deze waarde representeren.

Naam	Datatype	Bits
getal	int	00000000000000000000000000001010
conditie	bool	00000000000000000000000000000001
karakter	char	00000000000000000000000001111000

Afbeelding 8

Reference type instanties zitten op de *heap* gealloceerd.
Op de heap zitten objecten waar naartoe met *adressen* (*referenties*) wordt verwezen. Bijvoorbeeld...

Naam	Datatype	Adres
getallen	int[]	0x9FCB3
tekst	string	0x8DE2A

Afbeelding 9

Representaties van de reference types zitten zelf niet in een register als voorgaande, maar worden hier verondersteld aanwezig te zijn op bepaalde geheugenadressen, bepaalde referenties, die met de dataholders zijn geassocieerd.

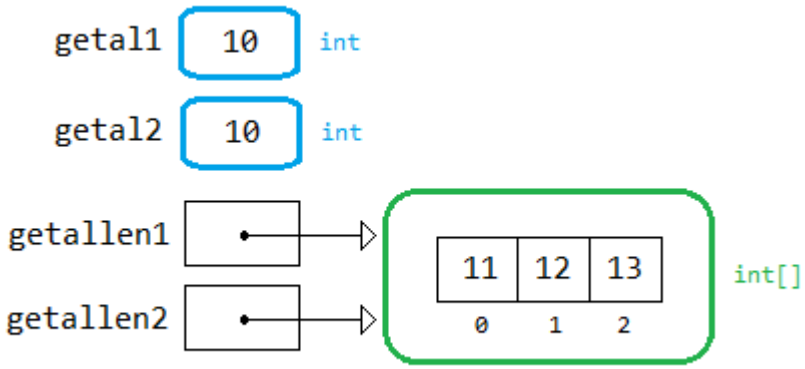
8.2.3.1. Toekenningen

Bij een assignatie wordt telkens een kopie van de directe inhoud aan de target toegekend.
Kennen we in volgend voorbeeld `getal1` toe aan `getal2`, dan zal een kopie van de `int` instantie met `getal2` worden geassocieerd.
Wordt `getallen1` toegekend aan `getallen2`, dan zal enkel een kopie van de verwijzing naar de `int[]` instantie met `getallen2` worden geassocieerd.

```
using System;
class OndiepeKopieBijToekenningenVoorbeeld1
{
    static void Main()
    {
        int getal1 = 10;
        int getal2 = getal1;

        int[] getallen1 = { 11, 12, 13 };
        int[] getallen2 = getallen1;
    }
}
```

C# broncode voorbeeld 184



Afbeelding 10

Wijzigen we daarna de inhoud van het orgineel (`getal1` en het tweede element van `getallen1`), dan zie je hoe `getal2` zijn waarde behoudt. Deze werkt immers met een kopie van de volledige instantie. De kopie is onaangetast.

```
using System;
class OndiepeKopieBijToekenningenVoorbeeld2
{
    static void Main()
    {
        int getal1 = 10;
        int getal2 = getal1;

        int[] getallen1 = { 11, 12, 13 };
        int[] getallen2 = getallen1;

        getal1 *= 2;
        getallen1[1] *= 2;

        Console.WriteLine($"getal1: {getal1}");
        Console.WriteLine($"getal2: {getal2}");

        Console.WriteLine($"getallen1:
{getallen1[0]}.{getallen1[1]}.{getallen1[2]}");
        Console.WriteLine($"getallen2:
{getallen2[0]}.{getallen2[1]}.{getallen2[2]}");

        Console.ReadLine();
    }
}
```

```
getal1: 20
getal2: 10
getallen1: 11.24.13
getallen2: 11.24.13
```

Console Application uitvoer 215

8.2.3.2. Meerdere verwijzingen naar hetzelfde object

Enkel wanneer je met reference type instanties werkt, kan je meerdere verwijzingen naar dergelijke objecten bijhouden.
Dit heeft als voordeel dat bij het doorspelen van informatie, niet met een kopie van de volledige instantie wordt gewerkt, maar op het orgineel kan worden ingewerkt.
Zo zal in het volgende voorbeeld zowel de variabele `aanTeVullenLoonlijst`, als de parameter `loonlijst`, met een kopie van de referentie werken die in de variabelen `loonlijstJohn` of `loonlijstJane` te vinden was.

```
using System;
class MeerdereVerwijzingenNaarZelfdeObjectVoorbeeld
{
    static void Main()
    {
        decimal[] loonlijstJohn = new decimal[12];
        decimal[] loonlijstJane = new decimal[12];
        do
        {
            Print(loonlijstJohn);
            Print(loonlijstJane);

            Console.Write("Toevoegen loon voor (John/Jane)?:
");
            string naam = Console.ReadLine();

            decimal[] aanTeVullenLoonlijst = null;
            if (naam == "John") { aanTeVullenLoonlijst =
loonlijstJohn; }
            else if (naam == "Jane") { aanTeVullenLoonlijst
= loonlijstJane; }
```

```
            Console.Write("Maand (1-12)?: ");
            int maandIndex = int.Parse(Console.ReadLine()) -
1;

            Console.Write("Loon?: ");
            decimal loon
= decimal.Parse(Console.ReadLine());
            Console.WriteLine();

            aanTeVullenLoonlijst[maandIndex] = loon;
        } while (true);
    }
    static void Print(Decimal[] loonlijst)
    {
        foreach (Decimal loon in loonlijst)
        {
            Console.Write(loon + " ");
        }
        Console.WriteLine();
    }
}
```

C# broncode voorbeeld 186

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
Toevoegen loon voor (John/Jane)?: John
Maand (1-12)?: 1
Loon?: 3180

3180 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
Toevoegen loon voor (John/Jane)?: John
Maand (1-12)?: 2
Loon?: 3185

3180 3185 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
Toevoegen loon voor (John/Jane)?: Jane
Maand (1-12)?: 1
Loon?: 1840

3180 3185 0 0 0 0 0 0 0 0 0 0
```

```
1840 0 0 0 0 0 0 0 0 0 0 0
Toevoegen loon voor (John/Jane)?:
```

Console Application uitvoer