C# Les 11 – Kortere schrijfwijzes

Met velden en properties (= eigenschappen) hebben we nu eigenlijk twee wat gelijkaardige mechanismes gezien. Eigenschappen zijn krachtiger omdat we daar gegevens mee kunnen afschermen. Het vervelende aan properties is dat je meer schrijfwerk hebt. In dit document bekijken we hoe we properties korter kunnen schrijven door de velden automatisch te laten genereren achter de schermen. We bekijken ook enkel alternatieve schrijfwijzen.

Laten we de twee technieken zoals we ze nu hebben behandeld nog eens naasteen plaatsen. Dit is een class met 2 publieke velden:

```
public class HuisDier
{
    public string Naam;
    public float Gewicht;
}

// Gebruik (bv. in Program):
HuisDier Patient1 = new HuisDier();

Patient1.Naam = "Fiffy";
Patient1.Gewicht = 9.5F;
```

Merk op dat we het speciale woordje *new* gebruiken om een nieuw HuisDier object te maken. Er is echter geen constructor voorzien. Elke class heeft echter een ***default constructor*** die een nieuw object aanmaakt als er geen constructor wordt gegeven door de programmeur. Een eigen constructor is dus niet altijd nodig.

Met de velden kan je een werkende applicatie maken, maar het kan zijn dat er onzinnige dingen in onze data komen te staan zoals een lege string als naam of een negatief gewicht.

We kunnen dit herschrijven naar properties, maar onze code wordt plots 13 regels lang i.p.v. 3… ☹

```
public class HuisDier
{
    private string _naam;
    private float _gewicht;

    public string Naam
    {
        get
        {
            return _naam;
```

```
        }
        set
        {
            _naam = value;
        }
    }

    public float Gewicht
    {
        get
        {
            return _gewicht;
        }
        set
        {
            _gewicht = value;
        }
    }
}
```

De code die je telkens moet herschrijven om een applicatie aan de gang te krijgen wordt *boilerplate* code genoemd. Om een simpele property zonder validatie te maken in C# had je vroeger telkens bovenstaande boilerplate code-regels nodig.

Auto-implemented properties

Gelukkig hebben ze hier wat op gevonden: a*uto-implemented properties*. Als je geen extra controles nodig hebt op je data kan je bovenstaand code ook zo schrijven:

```
public class HuisDier
{
    public string Naam { get; set; }
    public float Gewicht { get; set; }
}
```

Merk op dat

- De velden verdwenen zijn, deze worden achter de schermen automatisch gegenereerd en zijn nu niet meer bereikbaar.
- Het get- en set-blok hebben geen accolades meer.

Deze code doet bijna hetzelfde als het vorige codevoorbeeld (het schermt de velden ook nog af voor de objecten van de class zelf). De code is 10 regels korter geworden!

Als je toch validatie nodig hebt (validatie = controleren of de aan bepaalde voorwaarde voldoet, zoals geen lege naam of een negatief gewicht) heb je toch weer de langere schrijfwijze nodig. Een auto-implemented setter kan je ook *private* maken, hierdoor kan de property enkel veranderd worden door methodes in de class zelf. Dat wil dus zeggen dat je van buiten de class de naam niet meer kan wijzigen (zie voorbeeld hieronder). Als je geen extra methode voorziet kan je de naam enkel instellen via de constructor.

```
public class HuisDier
{
    public string Naam { get; private set; }
    public float Gewicht { get; set; }

    public HuisDier(string naam, float gewicht)
    {
        Naam = naam;
        Gewicht = gewicht;
    }
}

// Gebruik (bv. in Program):

HuisDier Patient1 = new HuisDier("James", 23.7F);
Patient1.Naam = "Fiffy"; // Dit werkt niet meer! Naam is readonly.
```

Soms is het handig om gegevens readonly te maken. Denk bijvoorbeeld aan een winkelwagentje met een int *TotaalProducten* dat enkel veranderd mag worden als er een product wordt toegevoegd of als er wordt afgerekend. De eigenschap instelbaar maken kan leiden tot inconsistente gegevens. Bijvoorbeeld een winkelwagen met 0 producten in maar met TotaalProducten = 5. Of misschien nog erger (voor de winkelier), een winkelwagen met 99 producten maar een TotaalProducten = 0 (moet de klant dan 0 euro betalen?).

Meer info over auto-implemented properties met voorbeelden vind je hier:

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-implement-a-lightweight-class-with-auto-implemented-properties

Object initializers

Met een object initializer kan je een object meteen waardes meegeven zonder de constructor te gebruiken:

```
class Program
{
    static void Main(string[] args)
    {
        HuisDier Patient1 = new HuisDier { Naam = "Fiffy", Gewicht = 9.5F };
    }
}

public class HuisDier
{
    public string Naam { get; set; }
    public float Gewicht { get; set; }
}
```

Deze schrijfwijze kan je op elke plek waar je een object wil maken gebruiken, ook in een List:

```
var Dieren = new List<HuisDier>
{
    new HuisDier { Naam = "Fiffy", Gewicht = 9.5F },
    new HuisDier { Naam = "Blacky", Gewicht = 27.3F },
    new HuisDier { Naam = "Flup", Gewicht = 3F }
};
```

Object initializers zijn soms handig om snel wat data bijeen te zetten om je programma te testen en als je geen constructor wil maken. Om met een object initializer te werken mogen de properties niet readonly staan (= set mag niet private staan). Anders heb je toch weer een constructor of een ander methode nodig.

Oefening: Quiz

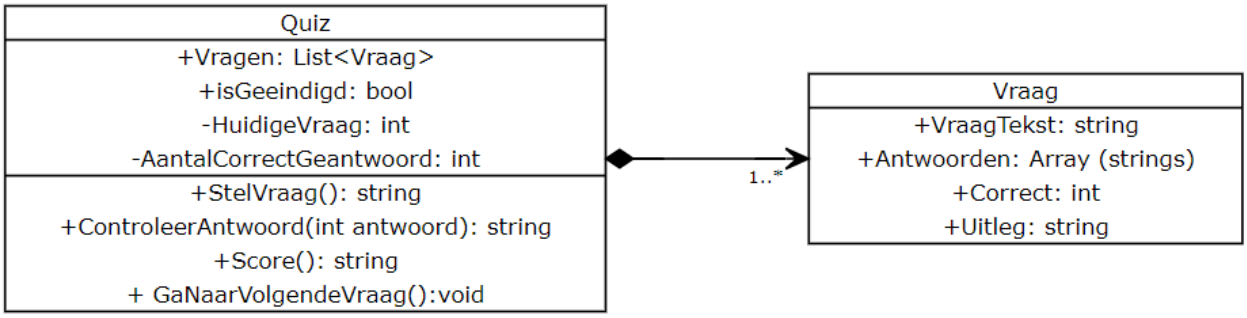
Maak een quiz-programma dat multiple choice vragen vraagt die je moet beantwoorden. Het programma heeft 2 classes: een Quiz class en een Vraag class.

De Quiz class:

Een object Quiz bestaat uit een lijst van Vraag objecten en houdt bij welke vraag in de lijst aan de beurt is (HuidigeVraag) en hoeveel vragen er correct beantwoord zijn (AantalCorrectGeantwoord). Voorzie ook een boolean *isGeeindigd* die op true wordt gezet als de laatste vraag is beantwoord. Bekijk ook eens de methodes in onderstaand class-diagram en hoe ze worden aangeroepen in de Main-methode.

De Vraag class:

Een vraag object bevat de vraag (VraagTekst), de verschillende antwoorden in een array van strings (waarvan er maar 1 correct is), een int die bijhoudt welke vraag er correct is en een uitleg die wordt getoond na het beantwoorden van de vraag.

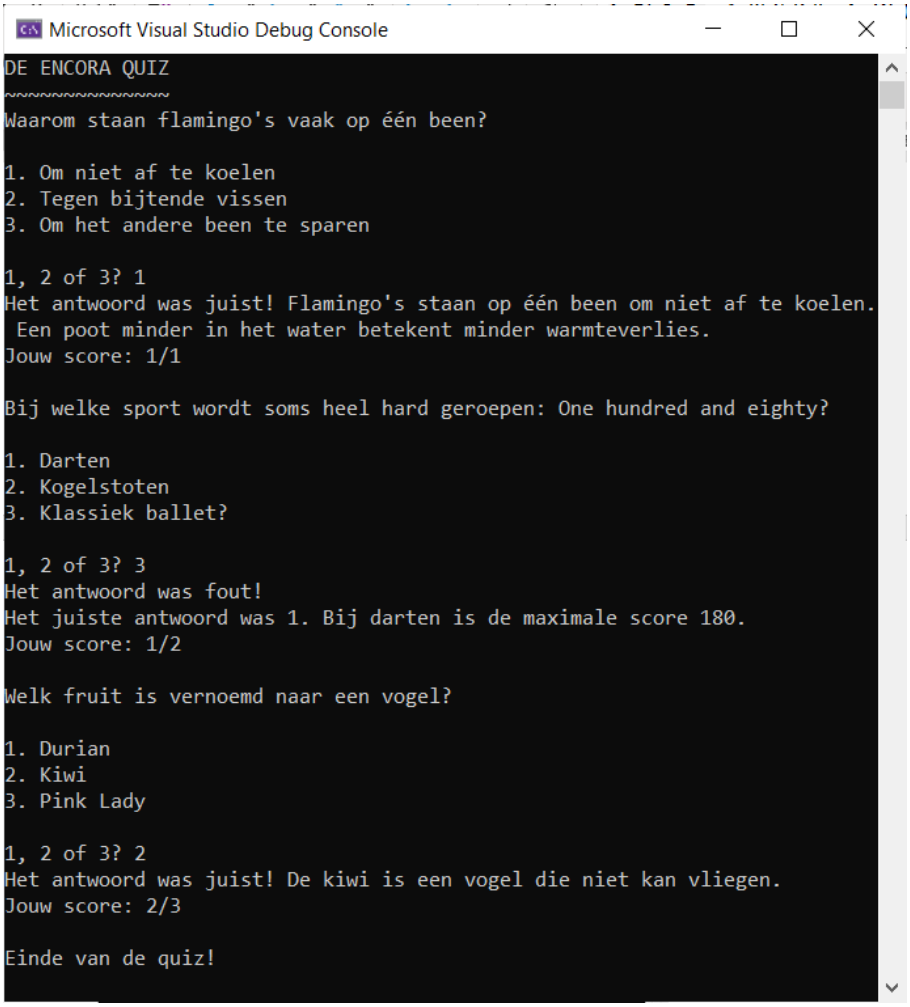



```
public class QuizProgramma
{
    static void Main(string[] args)
    {
        Quiz quiz = new Quiz();

        while (!quiz.isGeeindigd)
        {
            Console.Write(quiz.StelVraag());
            string strAntwoord = Console.ReadLine();
            int intAntwoord;

            while(!int.TryParse(strAntwoord, out intAntwoord))
            {
                Console.Write("Geef een getal in: ");
                strAntwoord = Console.ReadLine();
            }

            Console.WriteLine(quiz.ControleerAntwoord(intAntwoord));
            Console.WriteLine(quiz.Score());
            quiz.GaNaarVolgendeVraag();
            Console.ReadLine();
        }

        Console.WriteLine("Einde van de quiz!");
    }
}
```

Quiz-vragen vind je op het internet, bijvoorbeeld hier: https://raadsels.nu/grappige-quizvragen/

SCREENSHOTS:


C:\Users\so19079\source\repos\ConsoleAppLes...
DE ENCORA QUIZ
~~~~~~~~~~~~~~~~
Waarom staan flamingo's vaak op één been?
1. Om niet af te koelen
2. Tegen bijtende vissen
3. Om het andere been te sparen
1, 2 of 3?


Microsoft Visual Studio Debug Console
DE ENCORA QUIZ
~~~~~~~~~~~~~~~~
Waarom staan flamingo's vaak op één been?
1. Om niet af te koelen
2. Tegen bijtende vissen
3. Om het andere been te sparen
1, 2 of 3? 1
Het antwoord was juist! Flamingo's staan op één been om niet af te koelen.
Een poot minder in het water betekent minder warmteverlies.
Jouw score: 1/1
Bij welke sport wordt soms heel hard geroepen: One hundred and eighty?
1. Darten
2. Kogelstoten
3. Klassiek ballet?
1, 2 of 3? 3
Het antwoord was fout!
Het juiste antwoord was 1. Bij darten is de maximale score 180.
Jouw score: 1/2
Welk fruit is vernoemd naar een vogel?
1. Durian
2. Kiwi
3. Pink Lady
1, 2 of 3? 2
Het antwoord was juist! De kiwi is een vogel die niet kan vliegen.
Jouw score: 2/3
Einde van de quiz!