C# Les 9 – properties

Zoals we nu werken lukt alles wel, maar er is een probleem. We hebben geen controle over hoe onze objecten worden ingesteld omdat we *public fields* gebruiken. In ons programma kan een 999-deurs wagen gemaakt worden van het jaar 0. Of een auto met een negatief aantal deuren kan je ook maken… Het zou beter zijn als we wat meer controle hebben over onze data.

```
auto1.NDeuren = 999;
auto1.Bouwjaar = 0;
```

In plaats van velden public te zetten worden ze meestal private zetten. Daardoor zijn de velden enkel instelbaar door methodes in de eigen class en zijn ze niet meer bereikbaar vanuit andere classes (dus ook niet vanuit je Main-methode, want die zit waarschijnlijk in een andere class). Vaak worden private velden aangeduid met een underscore (_) voor de naam.

```
private string _naam;
private float _prijs;
```

Een **public** veld is instelbaar door iedereen.

Een **private** veld is enkel instelbaar door methodes in de class van waar het veld in staat.

We hebben nu enkel de constructor om onze velden in te stellen, daar kunnen we controlleren of er geen vreemde waarden worden ingesteld. Maar het rechstreeks instellen van de velden is nu onmogelijk ☹ (daar gaan we zo meteen iets aan veranderen).

Hieronder een voorbeeld van een class “Drank” met 2 private fields en een check in de constructor of de prijs niet negatief is.

```
public class Drank
```

```
{
    private string _naam;
    private float _prijs;

    public Drank(string naam, float prijs)
    {
        _naam = naam;

        if (prijs < 0)
        {
            _prijs = 0;
        }
        else
        {
            _prijs = prijs;
        }
    }
}
```

(I.p.v. foutieve invoer te gaan corrigeren gaan we in toekomst een error terugsturen.)

Eigenschappen/Properties

We kunnen de velden nu enkel via de constructor instellen. Om de prijs te veranderen van een bestaande drank kunnen we eventueel een nieuwe methode toevoegen aan onze Drank class:

```
public void setPrijs(float prijs)
{
    _prijs = prijs;
}

/*
In de Main:
Drank drank1 = new Drank("Fristy", 1);
drank1.setPrijs(2.5F);
*/
```

Voor de prijs op te vragen zouden we een getPrijs methode kunnen voorzien. Maar de code om een nieuwe prijs in te stellen of op te vragen is langer geworden dan voorheen, met de publieke velden ☹:

```
// Met een public field:
drank1.Prijs = 2.5F;

//Met een eigen methode:
drank1.setPrijs(2.5F);
```

Als je ervaring hebt met Java zal je dergelijke set- en get- methodes zeker herkennen. Anders dan Java en andere OO-talen voorziet C# een manier om toch de korte schrijfwijze van de publieke velden te kunnen gebruiken. Als je **get-** en **set**-blokken in een methode in je class zet kan je de naam van de methode gebruiken zoals een veld. Het woordje **value** in het set-blok krijgt automatisch de juiste waarde mee. **Prijs** wordt nu een **property** (of **eigenschap**) genoemd.

```
public class Drank
{
    private string _naam;
    private float _prijs;

    public Drank(string naam, float prijs)
    {
        _naam = naam;

        if (prijs < 0)
        {
            _prijs = 0;
        }
        else
        {
            _prijs = prijs;
        }
    }

    public float Prijs
    {
        get { return _prijs; }
        set { _prijs = value;}
    }
}

/*
In de Main:
Drank drank1 = new Drank("Fristy", 5);
drank1.Prijs = 2.5F;
*/
```

Je kan trouwens ook enkel een get-blok toevoegen, zo kan je enkel gevens in een object zetten tijdens de creatie (via de constructor) en wordt de instelling vanaf dan *read-only*. Je kan op die manier dus de toegang tot bepaalde gegevens gaan beperken.

```
public float Prijs
{
    get { return _prijs; }
}
```

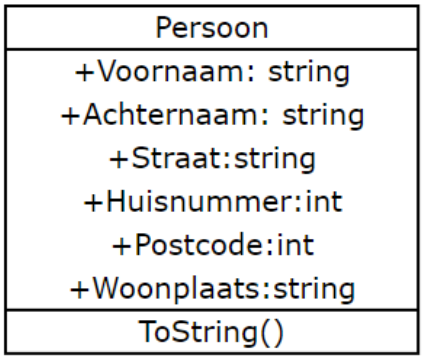
Oefening 1: Property "Naam"

a. Maak, gelijkaardig aan de property Prijs, een property *Naam* aan met een bijhorend get- en set-blok.

b. Kan je er voor zorgen dat er geen negatieve prijs kan ingesteld worden? **TIP**: Je moet hiervoor code in het set-blok van prijs gaan toevoegen.

Oefening 2: Persoon class

a. Maak een Persoon class met de volgende gegevens:



b. Maak een lijst met een aantal Persoon objecten in. Maak een programma waarmee je kan filteren op postcode (of woonplaats). Als je niets ingeeft moet het programma al de personen in de lijst weergeven.

Oefening 3: Refactoring vormen

a. Maak een class Rechthoek met als **property** breedte en hoogte. Zorg ervoor dat een negatief ingegeven breedte of hoogte wordt omgezet naar nul.

   Maak een methode berekenOppervlakte die de oppervlakte berekent en teruggeeft als float.

b. Maak een class Driehoek en Cirkel aan met bijhorende properties en berekenOppervlakte methode.

c. Breidt het programma verder uit met een while-lus zodat de gebruiker oppervlaktes kan blijven berekenen totdat ze beslist te stoppen.