

1.0.0

## **H11: Objecten (al dan niet) aanmaken**

# Constructors

## Constructors



Kennisclip voor deze inhoud

### Werking new operator

Objecten die je aanmaakt komen niet zomaar tot leven. Nieuwe objecten maken we aan met behulp van de `new` operator zoals we al gezien hebben:

```
Auto auto1 = new Auto();
```

De `new` operator doet 2 dingen:

- Hij reserveert ruimte voor dit object in het geheugen
- Hij zorgt ervoor dat initialisatiecode uitvoert om het object klaar te maken voor gebruik

Via de constructor van een klasse kunnen we code meegeven die moet uitgevoerd worden **telkens een nieuw object van dit type wordt aangemaakt**.

Constructors lijken erg op methodes. We zullen vaak zeggen dat het speciale methodes zijn, al bestaat daar wat technische discussie over. De constructor wordt in elk geval aangeroepen op een gelijkaardige manier aan een gewone methode. Daarom zetten we ronde haakjes: `new Auto()`.

### Soorten constructors

Als programmeur van eigen klassen zijn er 3 opties voor je:

- Je gebruikt geen (expliciete) constructors: het leven gaat voort zoals het is. Je kunt objecten aanmaken zoals eerder getoond. Achter de schermen gebruik je wel een zogenaamde **default** constructor.
- Je hebt enkel een **parameterloze** constructor nodig. Je kan nog steeds objecten met `new Auto()` aanmaken, maar je gaat zelf beschrijven wat er moet gebeuren bij de parameterloze constructor.
- Je wenst gebruik te maken van een of meerdere constructoren met parameters. Hierbij zal je dan extra argumenten kunnen meegeven bij de creatie van een object, bijvoorbeeld: `new Auto(25, 25000)`. Dit kan bijvoorbeeld een auto maken met 25l benzine in de tank en 25000km op de teller. De betekenis van de getallen hangt af van hoe je de constructor schrijft.



### Constructors zijn soms gratis, soms niet

Een lege default constructor voor je klasse krijg je standaard wanneer je een nieuwe klasse aanmaakt. Je ziet deze niet en kan deze niet aanpassen. Je kan echter daarom altijd objecten met `new myClass()` aanmaken.

Van zodra je echter beslist om zelf een of meerdere constructors te schrijven zal C# zeggen "Oké, jij je zin, nu doe je alles zelf". De default constructor die je gratis kreeg zal ook niet meer bestaan en heb je die dus nodig dan zal je die dus zelf moeten schrijven!

### Default constructor

De default constructor is een constructor die je zelf niet schrijft. Hij heeft nooit parameters.

- Standaard is hij `public` ([meer info](#))
- Heeft (zoals alle constructoren) als naam de naam van de klasse zelf

Stel dat we een klasse `Auto` hebben:

```
class Auto
{
    private int benzine = 5;
    private int kilometers;
}
```

We schrijven nergens code voor de constructor. Dan nog kunnen we `new Auto()` schrijven, maar dan wordt de auto aangemaakt met 5l in de tank en 0km (de defaultwaarde van `int`) op de teller.

### Parameterloze constructor

We willen telkens een Auto-object wordt aangemaakt dat dit een random aantal kilometers op de teller heeft. Via een parameterloze constructor kunnen we dat oplossen (je kan namelijk alleen expressies gebruiken als initiële waarde).

Eerst schrijven de parameterloze constructor, deze ziet er als volgt uit:

```

class Auto
{
    private int benzine = 5;
    private int kilometers;

    public Auto()
    {
        // zet hier de code die bij initialisatie moet gebeuren
        // d.w.z. wat er moet gebeuren zodra de auto wordt gemaakt
    }
}

```

De constructor moet de naam van de klasse hebben, public zijn en geen returntype definiëren.

Vervolgens voegen we de code toe die we nodig hebben:


```

class Auto
{
    private int benzine = 5;
    private int kilometers;
    private static Random randomGen = new Random();

    public Auto()
    {
        // ook in de constructor kan je this gebruiken
        this.kilometers = randomGen.Next(0,200000);
    }
}

```

Telkens we nu een object zouden aanmaken met `new Auto()` zal deze een willekeurige kilometerstand hebben. Je kan trouwens ook in de constructor een initiële waarde aan `benzine` geven.

 Zelfs als er een letterlijke initiële waarde wordt toegekend, gebeurt dit meestal in de constructor. Het is een kwestie van smaak, maar een constructor dient toch om te initialiseren.

### Constructor met parameter(s)

Soms wil je argumenten aan een object meegeven bij creatie. We willen bijvoorbeeld de inhoud van de tank en de kilometerstand meegeven die het object moet hebben bij het aanmaken. Met andere woorden, stel dat we dit willen schrijven:

```
Auto auto1= new Auto(25,20000);
```

Als we dit met voorgaande klasse , die enkel een parameterloze (default) constructor heeft, uitvoeren, zal de code een fout geven. C# vindt geen constructor die twee ints als parameter heeft.

Net zoals bij [overloading van methoden](#) kunnen we ook constructors overladen, d.w.z. alternatieven met dezelfde naam voorzien maar met een ander stel parameters. De code is gelijkaardig als bij method overloading:

```
class Auto
{
    int benzine;
    int kilometers;


    public Auto(int benzine, int kilometers)
    {
        this.benzine = benzine;
        this.kilometers = kilometers;
    }
}
```

Dat was eenvoudig. **Maar** denk eraan: je hebt een eigen constructor geschreven en dus heeft C# gezegd "ok, je schrijft zelf constructor, trek je plan. Maar de parameterloze versie zal je ook zelf moeten schrijven!" Je kan nu **enkel** je objecten met `new Auto(int benzine, int kilometers)` aanmaken. Schrijf je `new Auto()` dan zal je een error krijgen. Wil je die constructor nog hebben, dan zal je die met de hand moeten schrijven, bijvoorbeeld:

```
class Auto
{
    int benzine;
    int kilometers;

    public Auto(int benzine, int kilometers)
    {
        this.benzine = benzine;
        this.kilometers = kilometers;
    }

    public Auto() {
    }
}
```

 Er is geen grens op het aantal constructoren dat je kan schrijven, als ze maar verschillende parameters hebben.

Wanneer heb ik constructoren nodig?

Tot recent maakten we onze objecten steeds aan met de default constructor. Pas daarna gaven we eventuele properties de juiste waarde. Dat houdt een risico in: er is een periode waarin onze objecten nog niet "af" zijn. In het slechtste geval vergeten we zelfs om de properties in te stellen en krijgen we objecten die misschien ongeldig zijn.

Constructoren helpen dit probleem te voorkomen. Als we één constructor hebben, bijvoorbeeld `Auto(int benzine, int kilometers)`, **moeten** we die gebruiken. We kunnen dus niet vergeten bijvoorbeeld `auto1.Benzine = 25` te schrijven, want we worden gedwongen meteen `new Auto(25, 20000)` te schrijven.

Samengevat: **als er eigenschappen zijn die je meteen bij het aanmaken van een object wil instellen, maak er dan parameters van een constructor voor.**

## Constructor chaining

Als je meerdere overloaded constructoren hebt, hoeft je niet in elke constructor alle code voor objectinitialisatie te schrijven. Het sleutelwoordje `this` biedt ook de mogelijkheid **eerst** een andere constructor aan te roepen en eventueel andere operaties toe te voegen. Dit heet **constructor chaining**. In bovenstaand voorbeeld kan je ook dit schrijven:


```
class Auto
{
    private int benzine;
    private int kilometers;

    public Auto(int benzine, int kilometers)
    {
        this.benzine = benzine;
        this.kilometers = kilometers;
    }

    public Auto() : this(5,5)
    {
        // hier gebeurt niets meer
        // maar : this(5,5) zorgt dat een oproep van deze constructor
        // eerst de andere oproept, met beide waarden 5
    }
}
```

# Spelen met strings

Je kan objecten aanmaken door een omschrijving ervan in tekst om te zetten naar een constructoroproep. We noemen dit proces "deserialisatie". Een object omzetten naar een omschrijving noemen we "serialisatie". Deze omschrijving kan verschillende conventies volgen: XML, JSON, YAML, CSV,...

 We hebben eerder in de cursus geleerd hoe we CSV-bestanden moesten uitlezen. Fris dit zo nodig op, want we gaan nu een stap verder.

## Objecten deserializeren met CSV

CSV wordt vaak gebruikt om objecten tussen programma's te verplaatsen. Een object inlezen vanaf een CSV bestand is een vorm van deserializeren.

```
// Speler voornaam, familienaam, geboortejaar
string[] lijnen = File.ReadAllLines(@"C:\spelers.csv");
Speler[] spelers = new Speler[lijnen.Length];
for (int i = 0; i < lijnen.Length; i++)
{
    string[] kolomwaarden = lijnen[i].Split(',');
    spelers[i] = new Speler(kolomwaarden[0], kolomwaarden[1], Convert.ToInt32(kolomwaarden[2]))
}
```

## CSV wegschrijven

Je kan tekst uit een bestand lezen, maar uiteraard kan je ook naar een bestand wegschrijven. De 2 eenvoudigste manieren zijn:

- `File.WriteAllText` : deze gebruik je als je 1 enkele string wilt wegschrijven
- `File.WriteAllLines` : deze is de omgekeerde van `ReadAllLines()` en zal een array van strings wegschrijven.

Een voorbeeld:



```
string[] stringArray = new string[]
{
    "cat",
    "dog",
    "arrow"
};

File.WriteAllLines("file.txt", stringArray);
```

**Opgelet met het schrijven naar bestanden: dit zal onherroepelijk het target bestand overschrijven.**

Gebruik `if(File.Exists(pathToFile))` om te controleren of een bestand bestaat of niet. Eventueel kan je dan aan de gebruiker bevestiging vragen of je deze effectief wilt overschrijven.

Wil je CSV-bestand maken dan zal je eerst met `String.Join` of met stringinterpolatie een komma-separated lijst maken, bijvoorbeeld:

```
Speler[] spelers = {
    new Speler("Tim", "Dams", 1981),
    new Speler("Jos", "Stoffels", 1970),
    new Speler("Mounir", "Hamdaoui", 1984)
};

string[] lines = new string[spelers.Length];
for (int i = 0; i < lines.Length; i++)
{
    Speler speler = spelers[i];
    lines[i] = $"{speler.Voornaam},{speler.Familienaam},{speler.Geboortejaar}";
}

System.IO.File.WriteAllLines("spelers.csv", lines);
```

# Oefeningen

## Oefening: H11-Figuren

### Leerdoelen

- werken met klassen en objecten
- gebruik maken van properties om geldige waarden af te dwingen
- gebruik van een constructor

### Functionele analyse

Functioneel is dit programma hetzelfde als H10-figuren.

### Technische analyse

Voorzie je eerdere figuren ( `Rechthoek` en `Driehoek` ) van een constructor met twee parameters, waarvan de tweede telkens de hoogte voorstelt en de eerste de andere afmeting. Zorg dat deze constructor gebruik maakt van de properties, zodat je makkelijker objecten met de juiste afmetingen kan maken en toch dezelfde foutmeldingen krijgt als eerder.

Je zal een extra, parameterloze, constructor moeten toevoegen omdat `DemonstreerFiguren` van eerder moet blijven werken. Schrijf een nieuwe methode `DemonstreerFigurenMetConstructor` om te tonen dat je hetzelfde kan bereiken met de constructor met twee parameters. Deze moet ook opgeroepen kunnen worden van uit je keuzemenu.

### Voorbeeldinteractie(s)

Schrijf `DemonstreerFigurenMetConstructor` zodanig dat je exact onderstaande interactie krijgt:

```
Het is verboden een breedte van -1 in te stellen!  
Het is verboden een hoogte van 0 in te stellen!  
Een rechthoek met een breedte van 2,2m en een hoogte van 1,5m heeft een oppervlakte van 3,3m².  
Een rechthoek met een breedte van 3m en een hoogte van 1m heeft een oppervlakte van 3,0m².  
Een driehoek met een basis van 3m en een hoogte van 1m heeft een oppervlakte van 1,5m².  
Een driehoek met een basis van 2m en een hoogte van 2m heeft een oppervlakte van 2,0m².
```

## Uitbreiding CursusResultaat (SchoolAdmin project)

### Leerdoelen

- makkelijker objecten aanmaken
- gebruik maken van properties om geldige waarden af te dwingen

### Functionele analyse

Functioneel zal je niet veel verschil zien met eerder. Dit is zuiver een aanpassing die de kwaliteit van je code verhoogt.

### Technische analyse

Pas `CursusResultaat` aan zodat het huidige attribuut `Naam` privaat wordt (hernoem het dan ook naar `naam`) en voeg een **read-only** property `Naam` toe om deze informatie toch toegankelijk te houden.

Voor `Resultaat` doe je een gelijkaardige aanpassing, maar de property is niet read-only. Hij kan ingesteld worden, maar enkel op een waarde tussen 0 en 20. Aanpassingen naar een andere waarde worden genegeerd.

Voeg ook een constructor toe met twee parameters. De eerste is voor de naam, de tweede voor het resultaat. Doe ten slotte nodige wijzigingen in andere onderdelen van je code om met deze nieuwe code te werken, zonder iets te veranderen aan het gedrag van je systeem.

## Uitbreiding Cursus (SchoolAdmin project)

### Leerdoelen

- informatie op klassenniveau bijhouden
- meer toepassingen van de constructor

### Functionele analyse

We wensen cursussen automatisch te nummeren (zoals in DigitAP ook elke cursus een nummer heeft).

### Technische analyse

Voorzie eerst de klasse `Cursus` van een read-only property `Id` van type `int`. Pas ook de klasse `Cursus` aan zodanig dat het volgende beschikbare nummer voor een cursus wordt bijgehouden in een variabele `maxId`. De eerste cursus zal nummer 1 moeten krijgen. Zorg er ten slotte voor dat elke nieuwe cursus automatisch dit eerste beschikbare nummer krijgt en dat nummer stijgt voor elke cursus die wordt aangemaakt.

Neem dit nummer ook op in de methode `ToonOverzicht` van `cursus`, zodanig dat het cursusnummer tussen haakjes verschijnt, naast de titel van de cursus.

## Verdere uitbreiding Cursus (SchoolAdmin project)

## Leerdoelen

- properties en access control
- meer toepassingen van de constructor

## Functionele analyse

We willen het aantal studiepunten per cursus bijhouden. We willen niet dat het veel extra werk is om dit aantal in te geven en we weten dat de meeste cursussen 3 studiepunten tellen.

## Technische analyse

Voorzie eerst de klasse `Cursus` van een property `Studiepunten` van type `byte`. Je hoeft hierbij geen speciale controles te doen en mag gewoon het algemene patroon volgen, maar maak de setter `private`.

Pas de constructor aan, zodat hij ook het aantal studiepunten aanvaardt als (derde) parameter. Zorg er met behulp van chaining ook voor dat ook calls met een of twee argumenten geldig blijven, waarbij het aantal studiepunten van vanzelf wordt ingesteld op 3.

Zet ook het aantal studiepunten mee in het overzicht dat je krijgt via `ToonOverzicht`. Zet tussen haakjes naast het nummer voor de cursus, gevolgd door `"stp"`.

Pas ten slotte, in `DemonstreerCursussen`, je code aan zodat het vak webtechnologie wordt aangemaakt met ruimte voor 5 studenten en 6 studiepunten telt, terwijl databanken ruimte krijgt voor 7 studenten en 5 studiepunten telt.

## Student uit tekst lezen

### Leerdoelen

- werken met strings
- werken met arrays

### Functionele analyse

Voor de administratie is het handig snel en efficiënt nieuwe studenten te kunnen registreren. Zorg ervoor dat een gebruiker één regel tekst kan intypen met alle gegevens over een student, zonder veel verdere interactie.

### Technische analyse

Schrijf een methode `StudentUitTekstFormaat(string csvWaarde)` die een object van de klasse `Student` teruggeeft. Deze methode mag veronderstellen dat `csvWaarde` eerst de naam van de student bevat, gevolgd door een puntkomma, gevolgd door de geboortedag, puntkomma, geboortemaand, puntkomma, geboortjaar. Alle elementen van de geboortedatum worden voorgesteld als getallen, volgens de afspraken die je ook toepast om datums te noteren in het Belgische formaat. Het kan zijn dat er ook informatie is om de student meteen te registreren voor een of meerdere cursusresultaten. In dat geval staat er na het geboortjaar nog een puntkomma, dan de naam van de cursus, dan het behaalde cijfer. Per cursus herhaalt deze groep van twee elementen zich.

Schrijf daarna een methode `DemonstreerStudentUitTekstFormaat()`. Deze vraagt om de tekstvoorstelling van één student in te typen, maakt de student aan en toont dan het overzicht voor deze student. Neem deze methode ook op als optie in je keuzemenu voor `SchoolAdmin`.

⚠ Deze methode vereist geen bestaande `Student`. Ze heeft wel te maken met de klasse `Student`.

i De student hoeft niet opgenomen te worden in de array `Studenten` van een `Cursus`-object. We verbeteren dit later nog.

## Voorbeeldinteractie

Geef de tekstvoorstelling van 1 student in CSV-formaat:

```
>Bart Van Steen;04;03;1998;Boekhouden;14;Macro-economie;8;Frans, deel 2;18
```

Bart Van Steen, 22 jaar

Cijferrapport:

\*\*\*\*\*

Boekhouden: 14

Macro-economie: 8

Frans, deel 2: 18

Gemiddelde 13,3