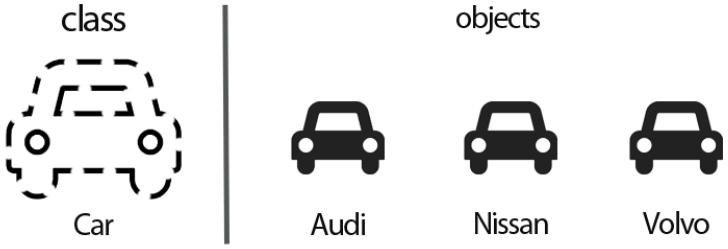
C# Les 8 – Objectgeoriënteerd programmeren

In de vorige lessen hebben we ingezoomd op imperatieve technieken. Je hebt gezien dat je daar al wel wat mee kan doen. Vanaf deze les gaan we ons arsenaal verder uitbreiden met object-geörienteerde (of kortweg OO) technieken en gaan we aan de slag met **classes** en **objecten**. De imperatieve bouwstenen zoals if-statements, lussen en methoden blijven belangrijk. Ze worden in zo goed als elk project samen gebruikt met classes en objecten.

Classes en objecten

Een **object** kan je zien als een soort tuple, maar dan met extra mogelijkheden. Je kan er informatie in bewaren zoals in tuples, maar je kan er ook **methodes** aan toevoegen.

Elk object behoort tot één bepaalde **class**. En om een object te maken hebben we die class nodig: de class werkt als een soort blauwdruk of sjabloon voor het object. De class bepaalt welke informatie en acties (= methodes) een object ter beschikking heeft.



Van één class kan je verschillende objecten gaan maken. Een beetje zoals koekjes maken met een koekjesvorm. De koekjesvorm is de class en de koekjes zijn de objecten. De koekjes hebben dezelfde vorm, maar kunnen wel van elkaar verschillen (zoals onze verschillende wagens).

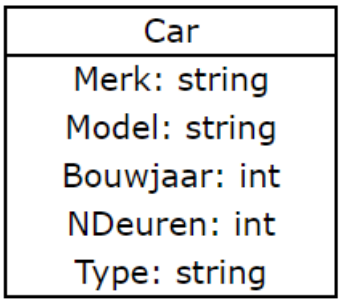
Een class maken doen we **buiten de accolades van class Program** (“Progam” is eigenlijk al een class). Een class om de gegevens van onze auto’s te bewaren zou er zo kunnen uitzien:

```
class Program
{
    static void Main(string[] args)
    {
    }
}

class Car
{
    public string merk;
    public string model;
    public int bouwjaar;
    public int nDeuren;
    public string type;
}
```

Nu hebben we een koekjesvorm (= class) gemaakt. Merk, model, bouwjaar, nDeuren en type zijn de **fields** of **velden**.

Schematisch wordt dit zo voorgesteld in een class-diagram:



Een object maken van een class

Een koekje (= object) maken met onze koekjesvorm doen we met het keyword **new**:

```
class Program
{
    static void Main(string[] args)
    {
        Car auto1 = new Car();
    }
}

class Car
{
    public string Merk;
    public string Model;
    public int Bouwjaar;
```

```
    public int NDeuren;
    public string Type;
}
```

We hebben nu een nieuw Car-object gemaakt. Zoals je ziet is auto1 van het type Car. Car is dus een soort data-type geworden.

De new-regel maakt dus een nieuw object. Een ander woord voor object is een *instance* of *instantie* van de Car-class. Er zitten nog geen gegevens in onze auto1. Dat kunnen we gaan toevoegen onder de new-regel:

```
Car auto1 = new Car();
auto1.Merk = "Volvo";
auto1.Model = "66";
auto1.Bouwjaar = 1975;
auto1.NDeuren = 2;
auto1.Type = "sedan";

Console.WriteLine($"De {auto1.Merk} {auto1.Model} is een {auto1.NDeuren}-
deurs {auto1.Type}. Deze auto werd gebouwd in {auto1.Bouwjaar}.");
```

Methods

We hebben nu een handige constructie gemaakt om informatie in te bewaren. We kunnen ook methodes in onze class zetten. Het schrijven van de zin “De Volvo 66 is een 2-deurs sedan. Deze auto werd gebouwd in 1975.” hoort eigenlijk ook bij de class. Deze kunnen we toevoegen zodat we niet voor elke auto een lange WriteLine(…) moeten gaan schrijven.

```
class Car
{
    public string Merk;
    public string Model;
    public int Bouwjaar;
    public int NDeuren;
    public string Type;

    public void PrintAuto()
    {
        Console.WriteLine(
            $"De {Merk} {Model} is een " +
            $"{NDeuren}-deurs {Type}. " +
            $"Deze auto werd gebouwd in {Bouwjaar}.");
    }
}
```

De PrintAuto methode is bruikbaar voor elk Car object. En als we de zin later willen gaan aanpassen moeten we de aanpassing maar op 1 plaats maken, handig!

Binnen onze methode zijn de velden (Merk, Model, Ndeuren, …) van het object zichtbaar. Merk op dat we voor een methode die bij een class hoort (meestal) geen “static” plaatsen.

Onze schematische voorstelling van onze Car class wordt nu zoals hiernaast staat afgebeeld.

Car
Merk: string Model: string Bouwjaar: int NDeuren: int Type: string
void: PrintAuto()

2 auto’s maken in onze Main en de gegevens tonen doen we zo:

```
Car auto1 = new Car();
auto1.Merk = "Volvo";
auto1.Model = "66";
auto1.Bouwjaar = 1975;
auto1.NDeuren = 2;
auto1.Type = "sedan";

Car auto2 = new Car();
auto2.Merk = "Fiat";
auto2.Model = "Panda";
auto2.Bouwjaar = 1981;
auto2.NDeuren = 3;
auto2.Type = "hatchback";

auto1.PrintAuto(); // output: De Volvo 66 is een ...
auto2.PrintAuto(); // output: De Fiat Panda is een ...
```

In het geheugen van de computer zitten nu 2 objecten met onderstaande gegevens. Elk Car object beschikt over de PrintAuto methode die in de class gedefinieerd is. Zoals je in bovenstaand voorbeeld ziet gaat deze methode aan de slag met de gegevens die in het object zitten waarop je deze methode aanroept.

Car auto1
Merk: "Volvo" Model: "66" Bouwjaar: 1975 NDeuren: 2 Type: "sedan"
void: PrintAuto()

Car auto2
Merk: "Fiat" Model: "Panda" Bouwjaar: 1981 NDeuren: 3 Type: "hatchback"
void: PrintAuto()

Een class is een manier om gegevens en bijhorende acties te groeperen. Van een class kunnen verschillende objecten worden gemaakt met het **new** keyword. Een object maken van een class noemt men **instantiëren**. Een object wordt ook wel een **instantie** van een class genoemd.

De gegevens in een object of class worden **velden** genoemd. Je kan ze opvragen of veranderen via de punt.

De methodes zijn acties die je op een object kan uitvoeren. Je kan ze uitvoeren door de punt en ronde haakjes achter de methode-naam te plaatsen.

Oefening 1: Classes en Objecten

Maak voor de gezelschapspellen in onderstaande tabel classes en instantieer 6 objecten met de gegevens. Maak ook een PrintSpel() methode die de gegevens zo toont:

"Het spel Catan kost 41,99 euro en is te spelen vanaf 10 jaar. Er zijn 2 spellen op voorraad en het staat niet in promo."

Naam	Prijs	Promo	Op voorraad	Vanaf leeftijd
Catan	41,99	False	2	10
Carcassonne	27,99	True	3	8
Uno	9,99	False	5	7
Scrabble	39,99	False	3	10
Pietjesbak	14,99	True	2	8

Constructor

Onze code om de gegevens in de objecten te zetten is wel eerder lang, we moeten 6 regels schrijven om de data te bewaren, voor 2 auto's hebben we 12 regels nodig en voor 10 hebben we 120 regels nodig... pfff. Gelukkig kunnen we een speciale methode maken die de data in de velden zet in 1 regel: een **constructor**.

Om een constructor te maken moet je in de class een methode toevoegen die de naam van de class heeft. In ons geval moeten we dus een methode maken met als naam Car. Een constructor is een beetje een apart geval want deze methode heeft geen return-type (dus je mag geen void, string of int voor de methode-signatuur plaatsen):

```
public Car(string merk, string model, int bouwjaar, int nDeuren, string type)
{
    Merk = merk;
    Model = model;
    Bouwjaar = bouwjaar;
    NDeuren = nDeuren;
    Type = type;
}
```

Het creëren van de objecten (ook wel instantiation of instantiatie genoemd) kan nu in 1 regel door weer gebruik te maken van het **new** keyword, maar dit keer met gegevens tussen de ronde haakjes:

```
Car auto1 = new Car("Volvo", "66", 1975, 2, "sedan");
Car auto2 = new Car("Fiat", "Panda", 1975, 2, "sedan");
```

Een **constructor** is een speciale methode die de naam van de class draagt. De constructor maakt nieuwe objecten aan.

Oefening 2: Speelgoed constructor

Maak een constructor voor de speelgoedspellen en instantieer de 6 objecten via de constructor.

Lijsten van objecten

Vaak heb je lijsten nodig om een programma te maken. Denk aan een schoenwinkel met allerlei producten, een autodeelsysteem voor lijsten van auto's en klanten of een website die een lijst van posts bijhoudt (zoals Facebook of Instagram). Al deze gegevens kan je bijhouden in **lijsten van objecten**.

Als je wil werken met lijsten mag je deze regel niet vergeten bovenaan je document:

```
using System.Collections.Generic;
```

Een lijst van objecten maken we op dezelfde manier als voorheen. We vermelden nu de naam van de class als data-type tussen het kleiner-dan-/groter-dan-teken:

```
var producten = new List<Product>
{
    new Product("Catan", 41.99F, false, 2, 10),
    new Product("Carcassonne", 27.99F, true, 3, 8),
    new Product("Uno", 9.99F, false, 5, 7),
    new Product("Scrabble", 39.99F, false, 3, 10),
    new Product("pietjesbak", 14.99F, true, 2, 8)
```

```
};
```

Een variabele beschikbaar maken voor heel de Progam class

We gebruiken een static veld om de producten-lijst beschikbaar te maken in heel de Program class. Doe dit enkel voor belangrijke variabelen die essentieel zijn voor je programma. Teveel van deze velden zorgt voor wanordelijke code.

In het geval van de speelgoedspellen is het handig om de lijst van producten in een static veld te gaan zetten, ze zijn de core-business van ons programma en kunnen zo eenvoudiger gemanipuleerd worden door de verschillende methodes:

```
public class Program
{
    static public List<Product> producten;

    static void Main(string[] args)
    {
        // ...
    }
}

public class Product
{
    public string Naam;
    public float Prijs;
    public bool Promo;
    public int OpVoorraad;
    public int VanafLeeftijd;

    public Product(string naam, float prijs, bool promo, int opVoorraad, int
vanafLeeftijd)
    {
        // ...
    }
}
```

In bovenstaande code zie je dat er nu een static veld is bijgemaakt bij de class Program. Dit staat op hetzelfde niveau als de velden van de Product class (zoals de velden Naam, Prijs, Promo, …). Maar waarom staat er bij het veld van Program static voor?

Static

De velden van de class Product worden gecreëerd als we een nieuw product-object maken (met het **new** keyword), ze horen dus bij een object. Ze bevatten voor elk object andere data (bijvoorbeeld “Catan”, “Scrabble”, …).

Een static veld daarentegen hoort bij een class, hiervoor hoef je geen object te maken (dus we hoeven geen “new Program()” te typen) om deze variabele te gaan gebruiken. De variabele bestaat al zonder een object te moeten aanmaken. Omdat een static veld bij een class hoort zit er dus ook maar 1 van in het geheugen van de computer. Een static veld kan dus geen verschillende data per object bevatten. (De vorige lessen hebben we steeds met static velden gewerkt.)

In ons programma heeft dat als gevolg dat we maar één lijst van producten kunnen maken. Maar dat is ons geval net wat we willen. Één lijst is voldoende want er kunnen meerdere producten in één lijst.

De lijst staat nu dus als veld buiten de accolades van de Main methode en is nu beschikbaar over heel de Program class. Dus het veld is ook zichtbaar in de andere methodes van Program.

Een static veld hoort bij een class en wordt maar één keer in het geheugen bewaard. Een niet static veld daarentegen bestaat pas als je een object aanmaakt met het new keyword en bevat per object andere data. **Een niet static veld hoort dus bij een object**.

Voorbeeld met auto’s

Hieronder een gelijkaardig voorbeeld met een lijst van auto’s. De lijst van auto’s (Autos) staat buiten de Main methode en is dus ook beschikbaar in de methode VoegAutoToe. De laatste regel van de methode VoegAutoToe creëert een nieuw Auto object met het new keyword en voegt deze auto toe aan de lijst Autos.

```
using System;
using System.Collections.Generic;

public class AutoApp
{
    static public List<Auto> Autos;

    static void Main(string[] args)
    {
        Autos = new List<Auto>
        {
            new Auto("Volvo", "66", 1975, 2, "sedan"),
            new Auto("Fiat", "Panda", 1981, 3, "hatchback")
        };

        VoegAutoToe();
    }

    static void VoegAutoToe()
    {
```

```
        Console.Write("Naam: ");
        string naam = Console.ReadLine();

        Console.Write("Model: ");
        string model = Console.ReadLine();

        Console.Write("Bouwjaar: ");
        string strBouwjaar = Console.ReadLine();

        int bouwjaar;

        while (!int.TryParse(strBouwjaar, out bouwjaar))
        {
            Console.Write("Foutieve invoer. Bouwjaar: ");
            strBouwjaar = Console.ReadLine();
        }

        Console.Write("Aantal deuren: ");
        string strNDeuren = Console.ReadLine();

        int nDeuren;

        while (!int.TryParse(strNDeuren, out nDeuren))
        {
            Console.Write("Foutieve invoer. Aantal deuren: ");
            strNDeuren = Console.ReadLine();
        }

        Console.Write("Type: ");
        string type = Console.ReadLine();

        Autos.Add(new Auto(naam, model, bouwjaar, nDeuren, type));
    }
}

public class Auto
{
    public string Merk;
    public string Model;
    public int Bouwjaar;
    public int NDeuren;
    public string Type;

    public Auto(string merk, string model, int bouwjaar, int nDeuren,
                string type)
    {
        this.Merk = merk;
        this.Model = model;
        this.Bouwjaar = bouwjaar;
        this.NDeuren = nDeuren;
```

```
        this.Type = type;
    }

    public void PrintAuto()
    {
        Console.WriteLine($"{Merk} {Model}, {NDeuren}-deurs" +
            $"{Type}, bouwjaar {Bouwjaar}.");
    }
}
```

Oefening 3: "Klant" Class

Maak een nieuwe class "Klant" waarin je de gegevens van een klant kan bewaren:

- Voornaam
- Achternaam
- Geboortedatum (van het type DateTime)
- Woonplaats

Deze code zou moeten werken:

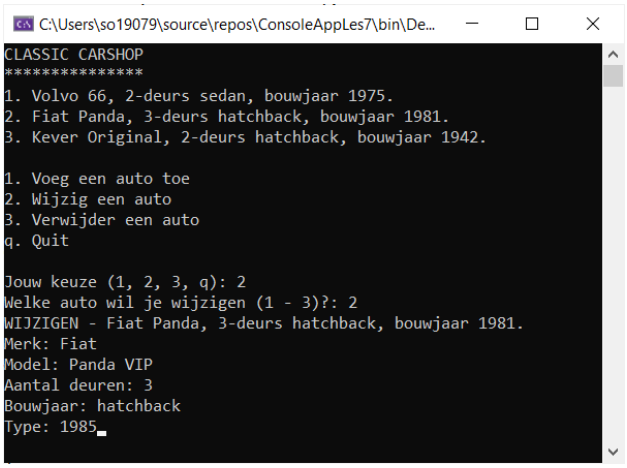
```
Klant nieuweKlant = new Klant("Luc", "De Vos", new DateTime(1962, 7, 12), "Gent");
```

Oefening 4: Toon auto's

A. Voeg aan het programma een nieuwe methode "ToonAutos" toe die de lijst met auto's van onze autoshop toont door voor elke auto in de lijst PrintAuto aan te roepen. Zorg dat er een oplopend nummer wordt geprint voor elke auto.

B. Maak in de AutoApp een methode "WijzigAuto" waarmee je de gegevens van een auto kan veranderen door het nummer in te geven.

C. Zorg dat je programma kan herhalen door een while-lus toe te voegen. Voeg een methode toe om een auto te verwijderen.


C:\Users\so19079\source\repos\ConsoleAppLes7\bin\De...
CLASSIC CARSHOP
****
1. Volvo 66, 2-deurs sedan, bouwjaar 1975.
2. Fiat Panda, 3-deurs hatchback, bouwjaar 1981.
3. Kever Original, 2-deurs hatchback, bouwjaar 1942.
1. Voeg een auto toe
2. Wijzig een auto
3. Verwijder een auto
q. Quit
Jouw keuze (1, 2, 3, q): 2
Welke auto wil je wijzigen (1 - 3)?: 2
WIJZIGEN - Fiat Panda, 3-deurs hatchback, bouwjaar 1981.
Merk: Fiat
Model: Panda VIP
Aantal deuren: 3
Bouwjaar: hatchback
Type: 1985


66