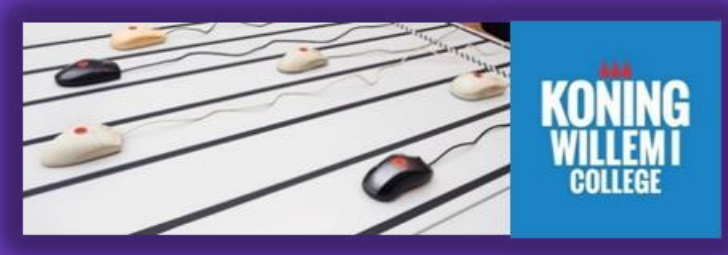
Basis standalone

Realiseren

hoofdstuk

4

Object-Oriented Programming



Algemene informatie

Onderwerp	Object-oriented Programming
Leerdoel(en)	1. Je kunt aangeven wat OOP is. 2. Je kunt een class aanmaken. 3. Je kunt een class initialiseren (via een object). 4. Je kunt een class method aanmaken. 5. Je kunt een class attribute aanmaken. 6. Je kunt het verschil tussen private/ public uitleggen en gebruiken.
Vereiste voorkennis	1. De student heeft de kennis uit de readers voorafgaand aan deze reader (Reader 4) opgedaan.
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Tijdens het schrijven van software ben je vaak bezig met het bouwen van een afspiegeling van de echte wereld. Bijvoorbeeld voor een systeem voor een autogarage is het nodig om een auto in te kunnen voeren. Zo wil je dus per auto de kleur, het aantal stoelen, merk, kenteken, eigenaar gegevens, etc., etc. kunnen invoeren. Je moet dus ergens één definitie (beschrijving) van zo'n auto hebben. Daarna kun je dus vanuit deze definitie één of meerdere auto's in je programma gaan aanmaken. Deze manier van werken noemen Object-Oriented Programming (OOP), in het Nederlands ook wel Object Georiënteerd Programmeren genoemd.

Object georiënteerd programmeren (=OOP) is in de jaren 90 van de vorige eeuw erg in opkomst gekomen. Voor die tijd was het lineair of procedure gericht programmeren, dit zorgde vooral bij grote programma's voor onleesbare en moeilijk onderhoudbare code.

Inhoud

1. Wat is OOP?

In de inleiding heb je al kunnen lezen wat OOP ongeveer is en waarom OOP is ontstaan. Maar wat is het nu precies. OOP is niets anders dan een manier van programmeren.

Bij programmeren geef je doormiddel van code opdrachten aan een computer die uitgevoerd dienen te worden. Bij het programmeren in OOP wordt een applicatie opgebouwd als een verzameling van objecten die allemaal behoren tot een soort of klasse. Zo probeer je de werkelijkheid te weerspiegel via code (denk aan het voorbeeld van de auto).

Waarschijnlijk is nu nog steeds niet helemaal duidelijk wat OOP is. Aan het einde van deze reader zal het een stuk duidelijker zijn. Je hebt dan een eenvoudig voorbeeld gemaakt waardoor je in praktijk hebt kunnen zien wat OOP is.

In de komende hoofdstukken gaan we dieper in op de diverse principes van OOP. Begrippen die hierbij onder andere aan bod komen zijn:

- Object
- Klasse (Class)
- Fields
- Eigenschappen (properties)
- Overerving (inheritance)
- Inkapseling (encapsulation)
- Overloading
- Overriding

Tegenwoordig zijn alle belangrijke programmeertalen object-georiënteerd. Voorbeelden van deze talen zijn:

- C#/ C++
- Python
- Java
- PHP (vanaf versie 5)
- Delphi (Pascal)
- Visual Basic

Voorbeeldne van talen die niet object-georiënteerd zijn, zijn:

- Assembler
- C
- Basic
- (ANSI) Pascal
- (ANSI) Cobol

2. Objecten en classes

Object-georiënteerd programmeren wordt ook wel het programmeren van classes (klassen) genoemd. Dit komt doordat alle functionaliteit die je aan een object wilt geven eerst in een klasse moet programmeren. De termen object en klasse worden vaak door elkaar en voor hetzelfde gebruikt. Dit is echter niet juist. Er is wel degelijk een verschil tussen een object en een klasse. We gaan eerst eens kijken wat objecten en klassen eigenlijk zijn.

2.1 Object

Een object is een instantie van een klasse. Voorbeeld: Je eigen auto is een object van de klasse auto. Maar de auto van je vriend(in) is dat ook. En toch zijn ze beide verschillend. Van één klasse maak je dus meerdere objecten. Dit noemen we **initializing** (instanties maken).

2.2 Classes (klassen)

Een klasse is een sjabloon (of tekening) waarvan objecten gemaakt kunnen worden. De klasse definieert hoe de objecten er uit gaan zien aan de hand van eigenschappen (fields en properties). Het gedrag van de objecten wordt vastgelegd in methoden (functies in klassieke programmeertalen).

Je kan dus zeggen dat de klasse een bouwtekening is en een object is het gebouwde huis.

Een derde begrip dat hierbij meteen aan de orde komt is inkapseling (**encapsulation**). Dit begrip heeft betrekking op de attributen en methoden. Het wil namelijk zeggen dat bij OOP de attributen van een object niet rechtstreeks opgevraagd of gewijzigd kunnen worden, maar dat dit altijd via de methodes gebeurt.

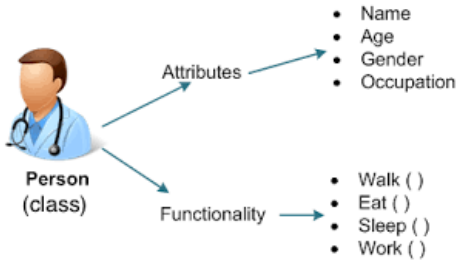
Dit wordt gedaan om ervoor te zorgen dat het object zelf de controle houdt over de waarde die in een eigenschap wordt weggeschreven.

Stel:

Het object **auto** heeft een eigenschap **snelheid**.

Als je dan rechtstreeks het attribuut zou kunnen wijzigen zou je deze auto elke snelheid kunnen geven. Bijvoorbeeld ook een negatieve waarde. Door het via een methode te laten verlopen, kun je controleren of de waarde wel aan voorwaarden voldoet (bijvoorbeeld groter dan 0 en kleiner dan 190km).

Diagram van een class genaamd *Person:*



2.3 Objecten maken

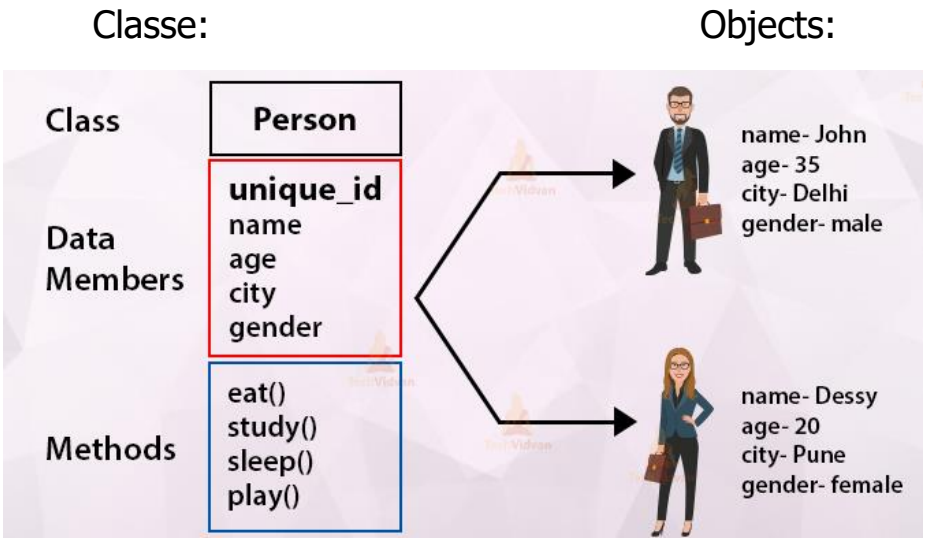
Om in programmacode iets te doen met een klasse moet er eerste een object van gemaakt worden. In de taal C# ziet het commando om een object te maken van een klasse er als volgt uit:

```
<klasse> <object> = new <constructor>;
```

Voorbeeldcode voor het aanmaken van 3 verschillende objecten, allen van de *class* Auto:

```
Auto auto1 = new Auto();
Auto auto2 = new Auto();
Auto auto3 = new Auto();
```

*Hierboven zie je hoe 3 keer een nieuwe **Auto** wordt **geïnitialiseerd** en **gedeclareerd.***



2.4 Constructor

Bij `<klasse>` wordt de naam van de klasse genoemd waar je een object van wil maken. Vervolgens is `<object>` de naam van het object zoals je dat in je programmacode wil gebruiken.

Bij `<constructor>` vul je de naam in de van de constructor die uitgevoerd moet worden om het object aan te maken. Deze naam van de constructor is altijd gelijk aan de naam van de klasse gevolgd door een open haak (en een sluit haak). Eventueel kunnen tussen de haakjes één of meerdere parameters staan.

Een constructor is een bijzondere methode van een klasse. Deze methode wordt uitgevoerd op het moment dat je van een klasse een object maakt.

Je kunt een constructor op twee manieren herkennen.

1. De constructor heeft dezelfde naam als de klasse.
2. Een constructor is de enige methode die geen waarden teruggeeft.

Bij het beschrijven van methodes komen we hier nog op terug. Een klasse kan meerdere constructors hebben. Ze moeten dan wel verschillen in het aantal of het datatype van de parameters.

Voorbeeld van de *constructor* voor de klasse **Auto:**

```
public Auto()
{
    // Zodra een new Auto() is aangemaakt,
    // Zal deze code uitgevoerd worden
    Console.WriteLine("Nieuwe auto aangemaakt.");
}
```

2.5 Fields

Een object (zoals een auto) bevat bepaalde waardes zoals een snelheid, kleur, aantal deuren, etc. Deze waardes kunnen voor ieder object anders zijn (elke auto heeft een kleur, maar niet alle auto's hebben dezelfde kleur). Deze waardes beschrijf je in de *class* en noemen we Fields of Properties (het verschil hiertussen leer je later). Hieronder een voorbeeld van enkele Fields binnen de *class* Auto:

```
class Auto
{
    public string Color;
    public int Speed;
    public int DoorAmount;
    public bool MotorRunning;
}
```

De Fields Color. Speed, DoorAmount en MotorRunning staan beschreven in de class ***Auto***

Het is mogelijk om sommige waardes altijd een **intial value** te geven (standaard waarde die alle objecten in eerste instantie hebben). Zo kun je bijvoorbeeld bepalen, dat de motor van een Auto in eerste instantie altijd uit staat. Zie onderstaand voorbeeld:

```
public string Color;
public int Speed = 0;
public int DoorAmount = 4;
public bool MotorRunning = false;
```

initial values *van bepaalde Fields*

Deze initial values worden aangemaakt, zodra het object Auto aangemaakt wordt (via het *new* statement).

```
public bool MotorRunning = false;
                           (field) bool Auto.MotorRunning
```

De icoontje, dat aangeeft dat MotorRunning een Field is.

2.6 Encapsulering via getters / setters

We hebben hierboven geleerd hoe we via Fields bepaalde waardes aan een class kunnen toevoegen, die daarna binnen het object veranderd/ uitgelezen kunnen worden. Hier zitten echter enkele gevaren aan. Het is bijvoorbeeld NIET mogelijk om de invoer van de waardes te bepalen (het is bijvoorbeeld technisch gezien mogelijk om een Auto met 99 deuren te maken). Of met een negatieve waarde:

```
Auto auto1 = new Auto();

// Dit moet natuurlijk niet mogen:
auto1.DoorAmount = -10;
```

Ongeldige waardes moeten we zien te voorkomen

We willen daarom een controle mechanische bouwen, die de invoer van waardes kan checken en daarna deze pas daadwerkelijk kan opslaan. Hiervoor gebruiken we **getters en setters.** Bij het woord **Get** moet je denken aan het uitlezen en bij het woord **Set** bedoelen we het veranderen van de waarde.

We dienen hiervoor eerst het huidige field **Private** te maken. Let er ook op, dat de eerste letter van de variabele een lage letter dient te worden (Naming Conventions):

```
private int doorAmount = 4;
```

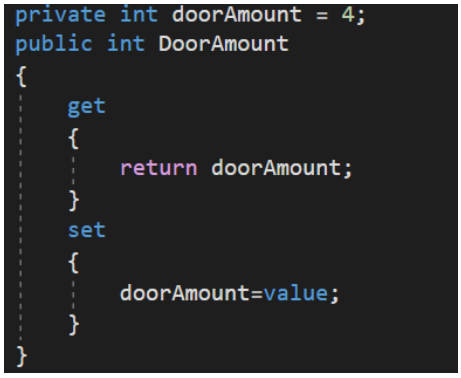
Het Private field is nu niet meer benaderbaar buiten de Class

Daarna maken we de zogeheten Getter en Setter aan. Zie code hieronder:

```
public int DoorAmount
{
    get
    {
        return doorAmount;
    }
    set
    {
        doorAmount=value;
    }
}
```

De code om een getter en setter te maken

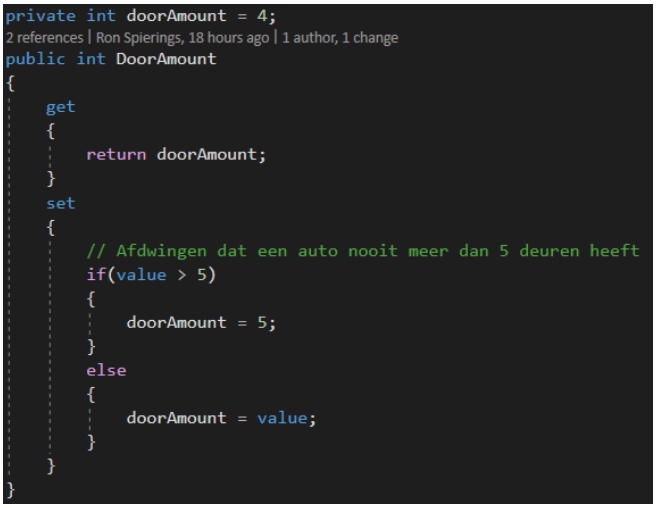
Dit levert dus de volgende code op:



```
private int doorAmount = 4;
public int DoorAmount
{
    get
    {
        return doorAmount;
    }
    set
    {
        doorAmount=value;
    }
}
```

Totaaloverzicht van een getter en setter

Binnen de Setter kun je code kwijt. Zie onderstaande voorbeeld:



```
private int doorAmount = 4;
2 references | Ron Spierings, 18 hours ago | 1 author, 1 change
public int DoorAmount
{
    get
    {
        return doorAmount;
    }
    set
    {
        // Afdwingen dat een auto nooit meer dan 5 deuren heeft
        if(value > 5)
        {
            doorAmount = 5;
        }
        else
        {
            doorAmount = value;
        }
    }
}
```

3. Voorbeeld applicatie maken

Je hebt nu behoorlijk wat theorie gekregen die je waarschijnlijk nog niet helemaal duidelijk is. OOP is helaas een lastig begrip om uit te leggen. In deze paragraaf ga je een voorbeeldapplicatie maken waarin we de verschillende termen voorbij zien komen.

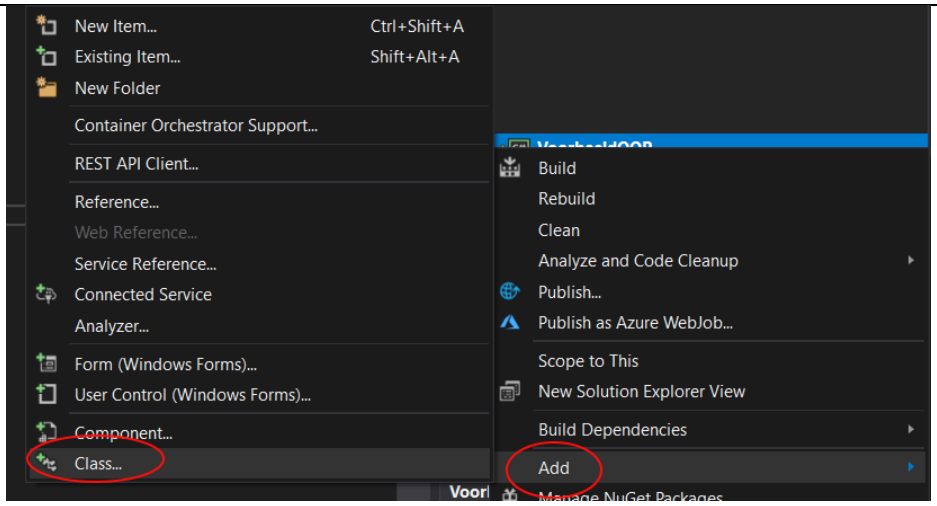
Volg onderstaande stappen:

- ☑ Maak in je de instructie folder een nieuw Console Applicatie met als naam: **VoorbeeldOOP**

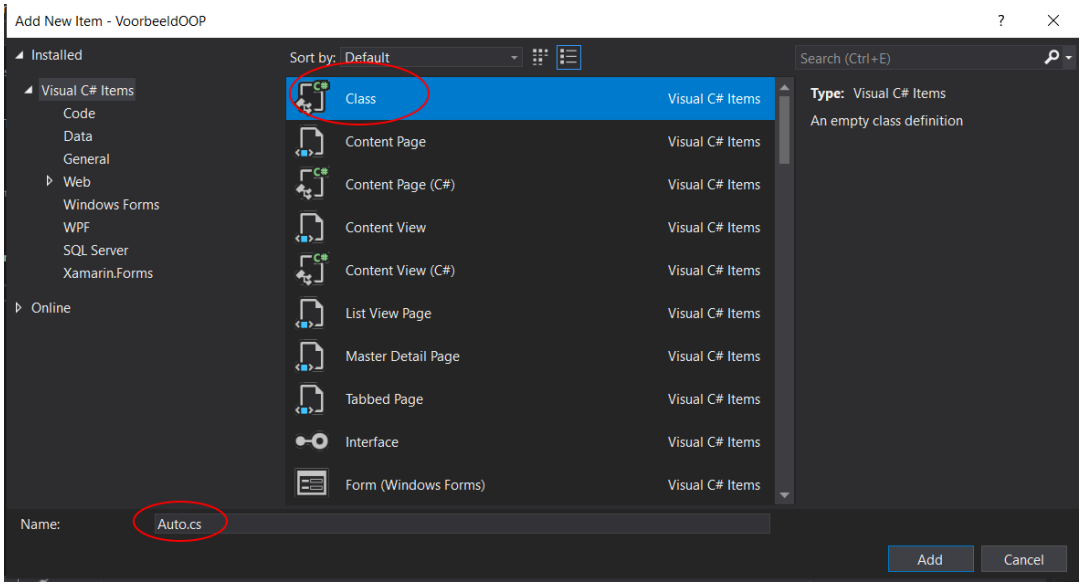
  Aan dit project ga je de klasse auto toevoegen. Zoals je weet is een klasse niets anders dan een bouwtekening waarmee je straks een object kunt maken.

- ☑ Ga met je muis op het projectnaam (VoorbeeldOOP) staan in de **Solution Explorer.**

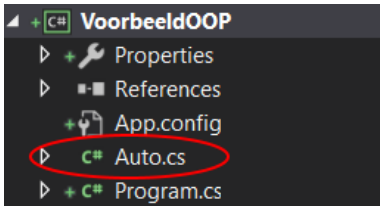
- ☑ Klik met je rechtermuisknop en kies de optie: **Add → Class.**



- Controlleer of "Class" ook daadwerkelijk geselecteerd is en geef het bestand de naam **Auto.cs**. En druk op **Add.** Zie onderstaande afbeelding:



- Er is nu een nieuw bestand genaamd "Auto.cs" in je project toegevoegd. Open dit bestand door er op te dubbelklikken.



- Bestudeer de code. Wat zie je wat je al kent (vergeleken met Program.cs)?

- Neem in de class **Auto** de onderstaande code over:

```
class Auto
{
    private int snelheid = 0;
    private int maxSnelheid;

    public Auto() // de constructor
    {
        maxSnelheid = 180;
    }

    public int GasGeven() // methode die snelheid verhoogt
    {
        if (snelheid < maxSnelheid)
        {
            snelheid += 10;
        }
        else
        {
            Console.WriteLine("De max snelheid is bereikt");
        }
        return snelheid;
    }

    public int Remmen()// methode die snelheid verlaagt
    {
        if (snelheid > 0)
        {
            snelheid -= 20;
        }
        else
        {
            Console.WriteLine("De auto staat al stil");
        }
        return snelheid;
    }

    public void SetMaxSnelheid(int snelh)
    {
        maxSnelheid = snelh;
    }
```

- Je hebt nu een klasse gemaakt. Om deze klasse te gebruiken, moet je nog wat code toevoegen aan **Program.cs**.
- Open **Program.cs** in Visual Studio.
- Plaats onderstaande code in **Program.cs:**

```
class Program
{
    static void Main(string[] args)
    {
        // 2 verschillende auto's initialiseren
        Auto audi = new Auto();
        Auto bmw = new Auto();

        // Voer de Methode SetMaxSnelheid uit op het Object bmw
        bmw.SetMaxSnelheid(220);

        // Simpele for loop (die 30 keer loopt)
        for (int i = 0; i < 30; i++)
        {
            int snelheid = audi.GasGeven();
            Console.WriteLine("De audi rijdt: " + snelheid + "km/uur");
            snelheid = bmw.GasGeven();
            Console.WriteLine("De bmw rijdt: " + snelheid + "km/uur");
            Console.WriteLine();
        }
        Console.ReadKey();

    }
}
```

☑ Start nu je programma en kijk goed naar de uitvoer zichtbaar op je scherm.

```
C:\Applicaties\C-Sharp\ProgrammerenLeerjaar2\VoorbeeldOOP\bin\Debu...

De max snelheid is bereikt
De audi rijdt: 180km/uur
De max snelheid is bereikt
De bmw rijdt: 220km/uur

De max snelheid is bereikt
De audi rijdt: 180km/uur
De max snelheid is bereikt
De bmw rijdt: 220km/uur

De max snelheid is bereikt
De audi rijdt: 180km/uur
De max snelheid is bereikt
De bmw rijdt: 220km/uur

De max snelheid is bereikt
De audi rijdt: 180km/uur
De max snelheid is bereikt
De bmw rijdt: 220km/uur
```

☑ In de volgende paragraaf staat de uitleg van de code. Zoals je kunt zien in het resultaat rijden er twee auto's met verschillende maximum snelheden.

 Je kunt nu **oefening 4.1** maken.

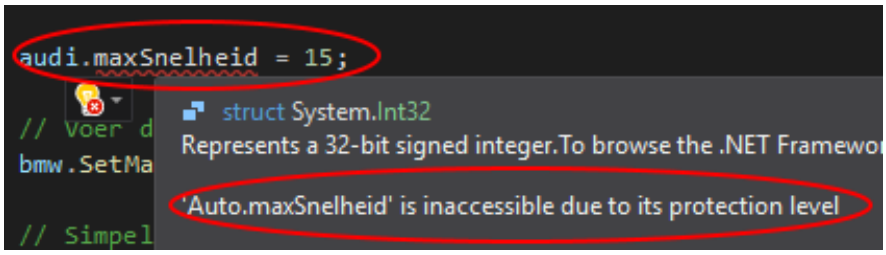
1.1 De code in de klasse Auto

De klasse auto is bedoeld om verschillende auto objecten te maken. We hebben in de klasse vastlegt wat de Fields (eigenschappen) van auto's zijn en we gebruiken methodes voor het gedrag.

De twee fields die voorlopig hiervoor voldoende zijn, zijn de huidige snelheid en de maximumsnelheid. Daarvoor maken we twee variabelen aan van het type integer. Merk op, dat deze **private** zijn gemaakt. Dit houdt in, dat *snelheid* en *maxSnelheid* niet door een andere klasse (zoals Program) aan te passen is.

1.2 Verschil private en public

- Probeer vanuit Program.cs maar eens de **private** fields snelheid en maxSnelheid aan te passen door onderstaande code te gebruiken.



*Je ziet hier de error die je krijgt, wanneer je vanuit een andere klasse dan Auto om een **private** field aan te passen.*

- Maak nu de fields snelheid en maxSnelheid **public** (zie onderstaande afbeelding). En probeer bovenstaande stap opnieuw.

```
public int snelheid = 0;
public int maxSnelheid;
```

- Vraag: Wat zou het verschil zijn tussen **private** en **public**?

We stellen ze allebei in op een standaard (start) waarde. De huidige snelheid is uiteraard nul. Dit doen we tegelijk bij de definitie van de variabele. De maximumsnelheid stellen we in op 180 en dat doen we in de **constructor**. In dit geval maakt het geen verschil. De **constructor** die gemaakt is krijgt geen parameters mee en hierin wordt alleen de maximumsnelheid ingesteld.

Het gedrag van de auto hebben we in twee methodes vastgelegd. De eerste **GasGeven()** zorgt ervoor dat de snelheid verhoogt wordt tot de maximumsnelheid. Je ziet hierbij meteen het voordeel dat het aanpassen van de snelheid gebeurt via de methode **GasGeven()** en niet rechtstreeks door de variabele snelheid op te hogen. Je kunt namelijk afvangen dat de snelheid niet hoger wordt dan de maximumsnelheid.

De methode **Remmen()** verlaagt de snelheid. Uiteraard mag de snelheid niet lager dan 0 worden.

Deze klasse zelf kunnen we niet uitvoeren. Hiervoor zullen eerste in het programma Program.cs objecten van de auto moeten maken.

1.3 De code in Program.cs

Het programma **Program.cs** is het feitelijke programma dat uitgevoerd wordt als je jouw applicatie opstart. Als dit programma uitgevoerd wordt, wordt er eerste gezocht naar een methode met de naam **Main()**. Als deze methode gevonden wordt, wordt deze uitgevoerd.

Het eerste dat gebeurd is dat er van de klasse auto twee afzonderlijke objecten gemaakt worden. Eén met de naam **audi** en één met de naam **bmw**. Vervolgens wordt via de methode **SetMaximuSnelheid()** de maximumsnelheid voor de bmw veranderd in 220. Die van de audi blijft 180, de default waarde zoals we die in de klasse gedefinieerd hebben.

Vervolgens wordt in een for-lus de methode **GasGeven()** voor beide auto's aangeroepen en de actuele snelheid die deze methode terug geeft wordt op het scherm getoond.

2. Samenvatting

In dit hoofdstuk heb je een korte introductie gehad in Object georiënteerd programmeren. Je bent nu in staat om zelf een eenvoudige klasse te maken en deze te gebruiken als objecten.

Je hebt het verschil gezien tussen een *class* (de bouwtekening) en een *object* (een daadwerkelijke uitwerking van de bouwtekening). Ook heb je geleerd dat een *class* bepaalde eigenschappen (*fields* en *properties*) heeft en dat deze pas met een waarde (*value)* gevuld worden, zodra er van de *class* een *new()* instantie van gemaakt is.

Je hebt de volgende onderwerpen deze reader behandeld. Probeer per onderwerp in één zin te vertellen de functie ervan is:

- Object
- Klasse (Class)
- Fields
- Getters / Setters (Properties)
- Waardes van de eigenschappen (values)

 Je kunt nu **oefening 4.2** maken.