

leren. durven. doen.



C# FUNDAMENTALS

C# PROGRAMMEREN 1



OOP

Klassen en Objecten



Inhoud

1. Inleiding Klasse en Object

- Wat zijn Objecten?
- Wat zijn Klassen?

2. Klasse in C#

- Declaratie van Klasse?
- Klasse Fields en properties
- Access Modifiers
- Gebruik van Klasse en Object
- Constructoren
- Klasse Methoden



Klassen en Objecten

Modelleren van echte-wereld objecten

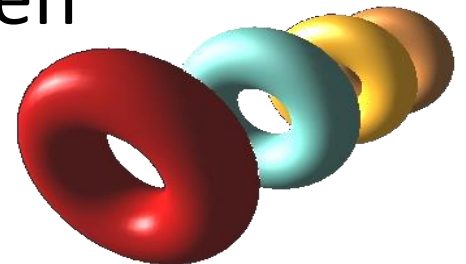


Wat zijn Objecten?

- Software objecten modelleren real-world objecten of abstracte concepten
 - Voorbeelden:
 - bank, rekening, klant, hond, fiets, wachtrij
- Elk object heeft een toestand (gegevens) en gedrag (operaties die mogelijk zijn)
 - Bankrekening Toestand:
 - rekeninghouder, saldo, soort rekening
 - Bankrekening gedrag:
 - HaalAf(), Stort(), ...

Wat zijn Objecten? (2)

- Op welke manier kunnen software objecten real-world objecten voorstellen?
 - Door gebruik van variabelen/data voor de toestand van een object weer te geven
 - Door gebruik van methoden om het gedrag te bepalen
- Een software object is een samenstelling van Velden, eigenschappen en methoden



Object is abstractie van echte wereld object

□ rekening

□ klant

□ boodschappenlijst

...

□ string

□ numbers

□ List

□ ...

**“Echte” wereld
objecten**

Software objecten

Klasse

- Klasse definieert de “Blauwprint” (structuur) voor objecten
 - soort prototype, template
- Klasse definieert *members*:
 - Velden en Eigenschappen waarbij object van Klasse gegevens (toestand) kan in bewaren
 - Velden: **Fields**
 - Eigenschappen: **Properties**
 - Gedrag (mogelijke operaties)
 - d.m.v **methods**
- Een class definieert de methods en soort data geassocieerd aan een software object

Vershil Klasse & Object - Voorbeeld

Class *Hond*

Eigenschappen:

- Ras (string)
- Kleur (string)
- Leeftijd (int)

Gedrag:

- Spreek()
- Zit()
- Eet()

Object *Lassie*

Eigenschappen:

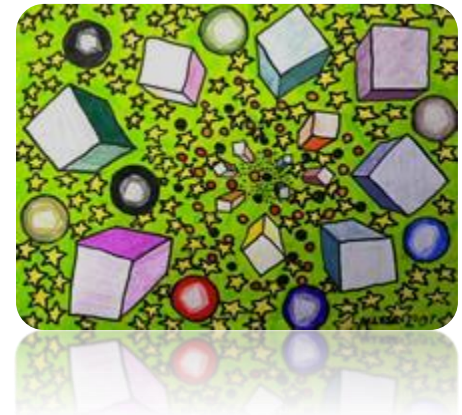
- Ras: "Collie"
- Kleur: "Bruin"
- Leeftijd: 2 jaar

Gedrag:

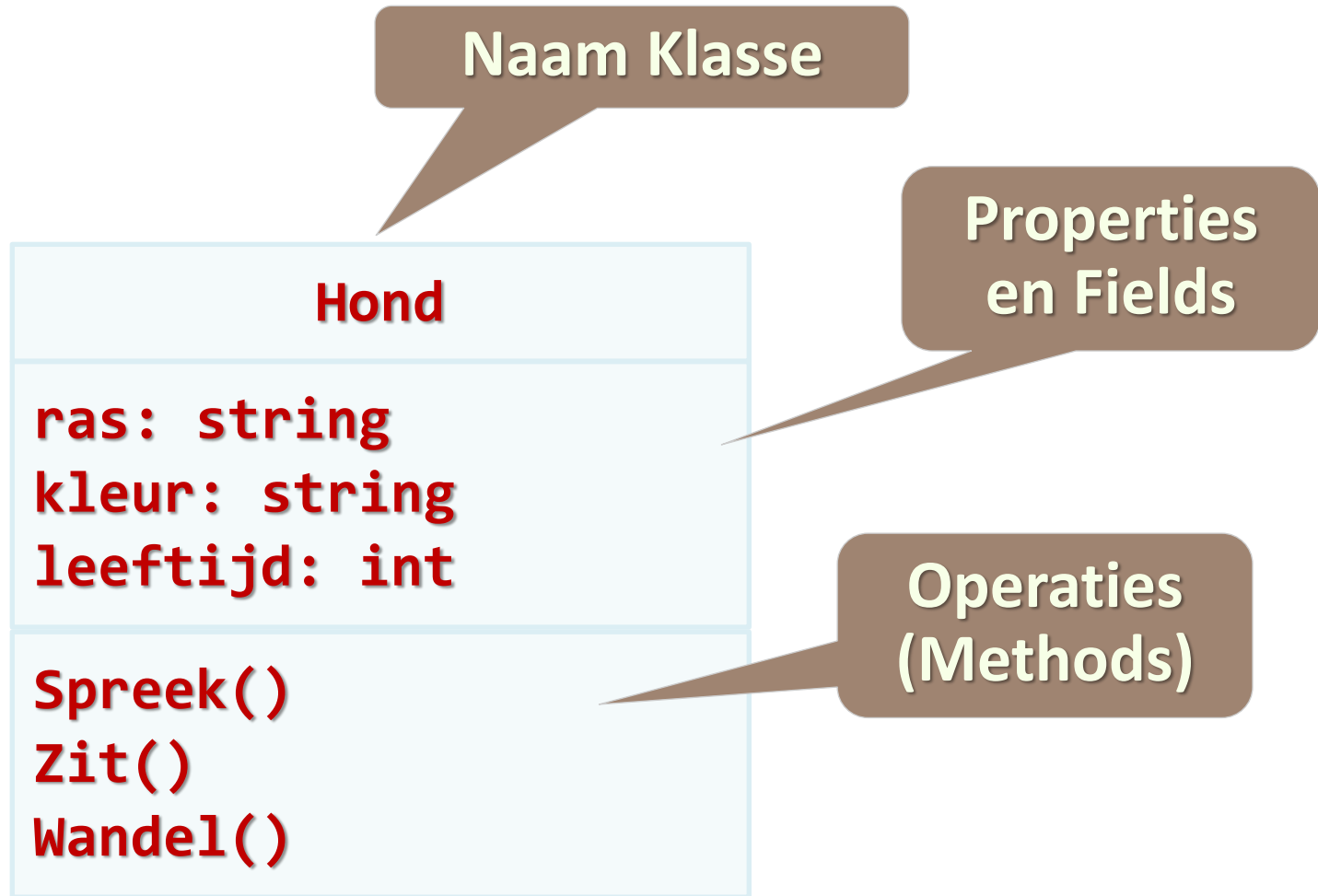
- Spreek()
- Zit()
- Eet()

Klassen in C#

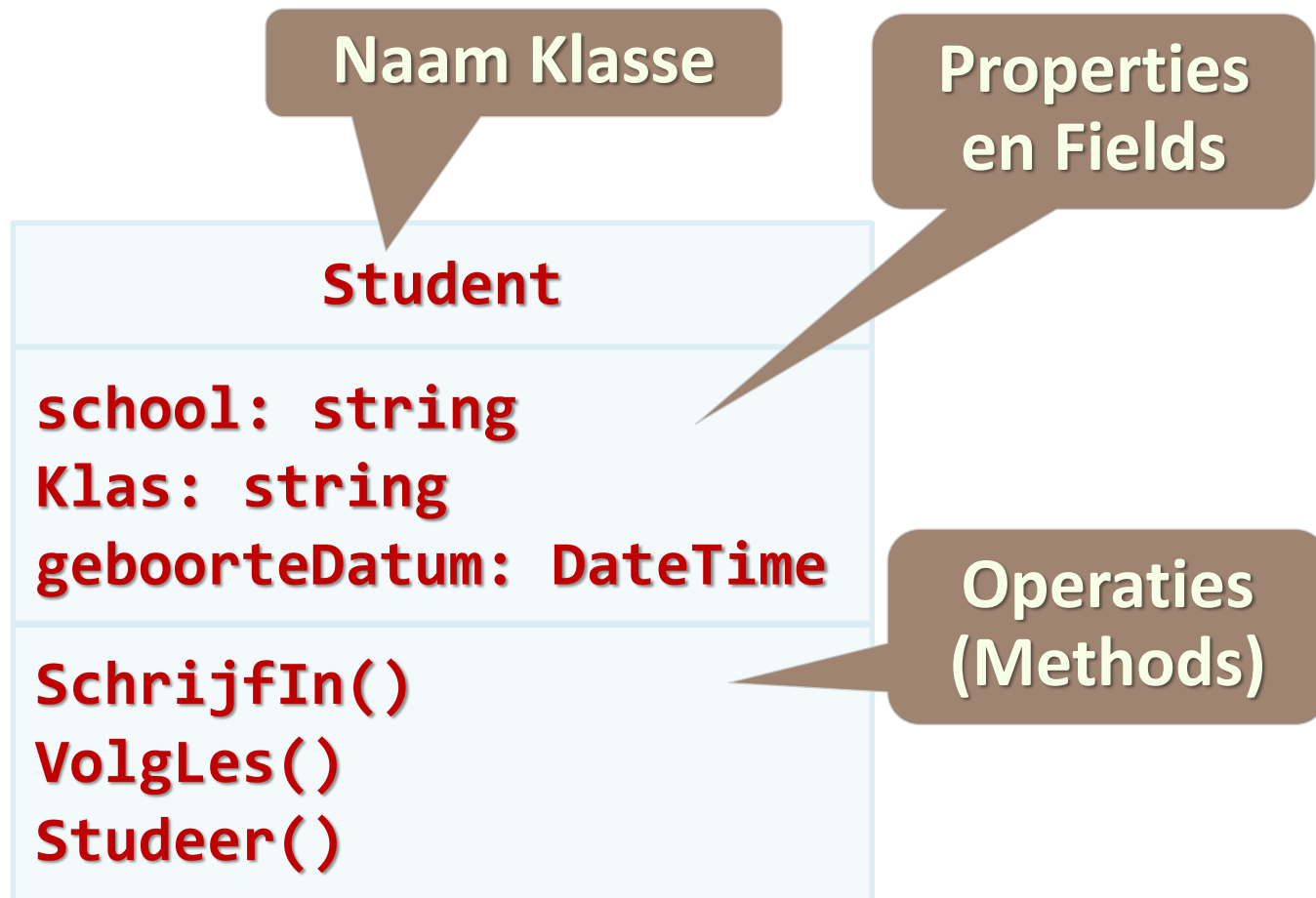
- Implementatie is **encapsulated** (ingekapseld, verborgen)
- Klassen in C# kunnen bevatten:
 - Velden: Fields (*member variables*)
 - Eigenschappen: *Properties*
 - Operaties: *Methods*
 - Constructoren: *Constructors*
 - Inner typen (enum, struct, of andere class definities)
 - ...(events, indexers, operators, ...)



Klasse – voorbeeld 1



Klasse – voorbeeld 2



Instanties van class Hond (Objecten)

Voorbeelden + Demo

Klasse

Dog

```
ras: string  
kleur: string  
leeftijd: int
```

```
Spreek()  
Zit()  
Eet()
```

Lassie

```
ras = "Collie"  
kleur = "Bruin"  
leeftijd = 2
```

Object



Bobby

```
ras = "WhiteTerrier"  
kleur = "Wit"  
leeftijd = 3
```

Object



Kirby

```
ras = "Pitbull"  
kleur = "Beige"  
leeftijd = 8
```

Object



Klasse en Object van Klasse

- Definitie van Klasse Hond

```
class Hond
{
    string ras; //field
    string kleur; //field
    int leeftijd; //field
    //...
}
//...
// Declaratie van 2 objecten van Klasse Hond:
Dog Lassie;
Dog Bobby;
```

Oefening: Maak een Class Kip en maak de 3 objecten aan van Class Kip

Klasse

Kip

```
kleur: string  
eierenPerDag: short
```

```
Spreek()  
LegEi()  
Eet()
```

Louise

```
kleur = "Bruin"  
eierenPerDag = 2
```

Object



Sofie

```
kleur = "Bruin"  
eierenPerDag = 1
```

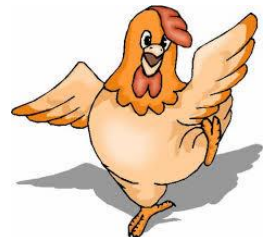
Object



Claire

```
kleur = "Bruin"  
EierenPerDag = 0
```

Object



Objecten

- Een **object** is een concrete **instantie** van een bepaalde klasse
- Een object creatie van een class wordt **instantiatie** genoemd
- Een object heeft een toestand of status (**state**)
 - Een aantal waarden geassocieerd met hun fields
- Voorbeeld:
 - **Class**: Hond
 - **Object**: `Hond lassie = new Hond();`
 - **Field** waarde ras van object Lassie = “Collie”

Velden (Fields)

Fields definiëren en gebruiken



Velden (Fields)

- **Fields** zijn *data members* van een klasse
 - Kunnen variabelen of constanten zijn
 - Aanspreken van een field zal geen actie op het object veroorzaken
 - Enkel toegang tot de waarde van een field
- Voorbeeld:
 - **String.Empty** ("" string)
 - **String.Length**



Soorten Velden

- Fields zijn **data members** gedefinieerd binnen een class
 - Fields houden de interne status of toestand van een object bij
 - Kunnen **static** zijn
 - Kunnen **private / public / protected / ...**

```
class Dog
{
    private string name;
    private string breed;
    private int age;
    protected string color;
}
```

**Declaraties
van velden**

Toegang tot Velden

- **Constance velden** (kunnen enkel gelezen worden)
- **Variable velden** kunnen zowel gelezen als gewijzigd worden
- Gebruik properties om de fields aan te spreken
- Voorbeelden:

```
// String.Empty is Property van klasse String  
String empty = String.Empty;
```

```
// Int32.MaxValue geeft constante terug  
int maxInt = Int32.MaxValue;
```



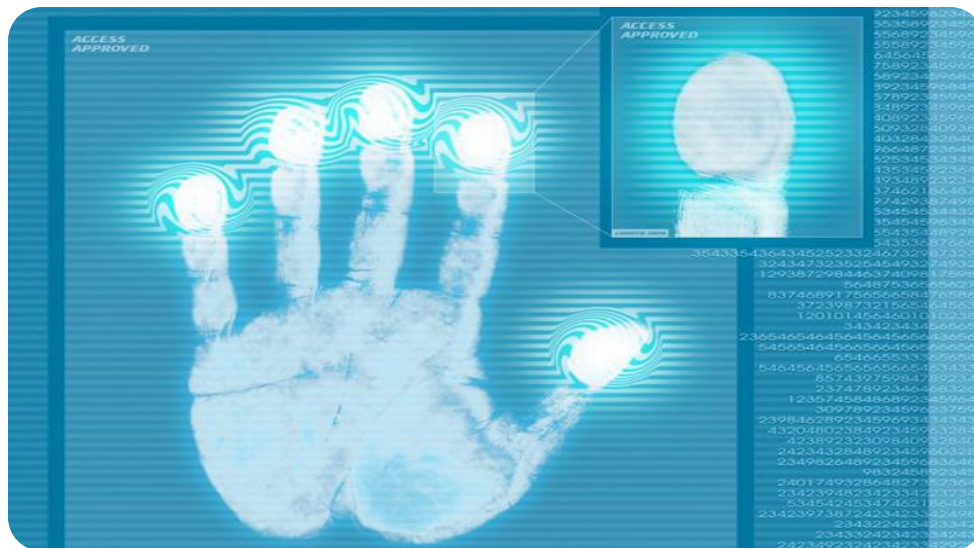
Velden met constante waarde

- 2 types van velden met enkel-leestoeegang:
 - Compile-time constanten – `const`
 - Worden vervangen door hun waarde bij compilatie
 - Runtime constanten – `readonly`
 - Eenmalig toegegend bij object creatie

```
class Program
{
    public const float PI = 3.14159f;
    public readonly string grootte = "XXL";
}
```

Toegang regelen

Public, Private, Protected, Internal



Toegang Regelen via Access Modifiers

- Classes, Fields, Properties, Methoden,...kunnen access modifiers hebben
 - Gebruikt om te toegang te regelen
 - Ondersteunt het OOP principe "**encapsulatie**"
- Access Modifiers:
 - **public** – toegankelijk vanuit alle klassen
 - **protected** – toegankelijk door class zelf en zijn afgeleide klassen
 - **private** – toegankelijk enkel binnen klasse zelf
 - **internal** (default) – toegankelijk binnen de eigen assembly, d.w.z. binnen hetzelfde VS project

'this' Sleutelwoord

- keyword **this** verwijst naar het huidige object (instantie) van de klasse
- Voorbeeld:

```
class Kip
{
    private string kleur="Bruin"; // veld

    public void PrintKleur() // methode
    {
        Console.WriteLine(this.kleur);
    }
}
```




Definiëren van eenvoudige Classes

Voorbeeld

Voorbeeld: Definieer een Klasse "Hond"

- definieer een eenvoudige klasse **Hond** die informatie voorstelt over een hond
- De hond heeft een **naam** en **ras**
 - De klass laat toe om **naam** en **ras** te lezen en te wijzigen via methoden **GetNaam()**, **GetRas()**, **SetNaam()** en **SetRas()**
 - De hond kan blaffen (**Spreek()**). Dan wordt de tekst “Woef!” naar de console geschreven

Klasse Hond – Voorbeeld

```
public class Hond
{
    private string naam;
    private string ras;

    public void SetNaam(string naam)
    {
        this.naam = naam;
    }
    public string GetNaam()
    {
        return this.naam;
    }
}
```

(voorbeeld wordt vervolgd..)

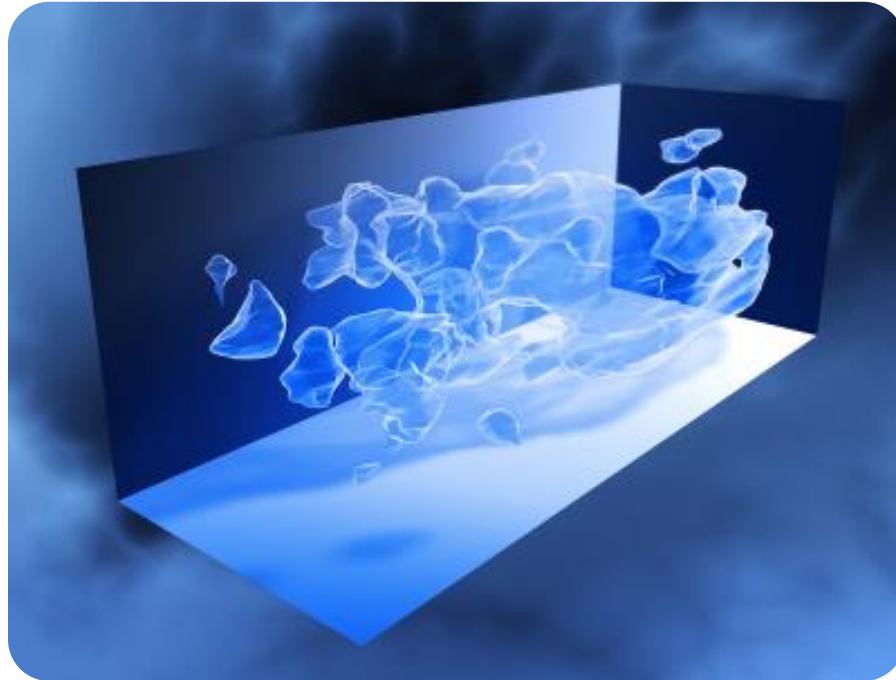


Klasse Hond – Voorbeeld (2)

```
public void SetRas(string ras)
{
    this.ras = ras;
}
public string GetRas(string ras)
{
    return this.ras;
}
public void Spreek()
{
    Console.WriteLine("{0} zegt: Woef!",
        this.naam ?? "[hond zonder naam]");
}
}
```



Gebruik van Klasse



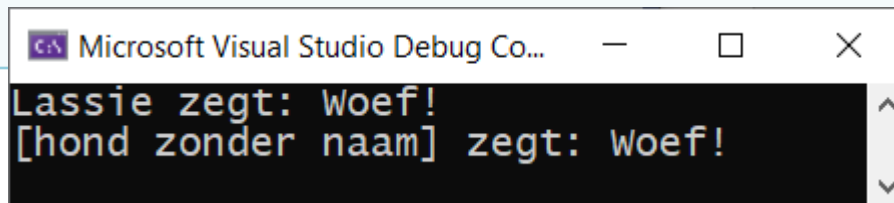
Hoe Klasse gebruiken? (Niet-Static)

1. Maak een **instantie** aan

- Roep de public methoden/Properties aan

```
class Program
{
    static void Main()
    {
        Hond hond1 = new Hond();
        hond1.SetNaam("Lassie");
        hond1.SetRas("Collie");
        hond1.Spreek();

        Hond hond2 = new Hond();
        hond2.SetRas("Onbekend");
        hond2.Spreek();
    }
}
```



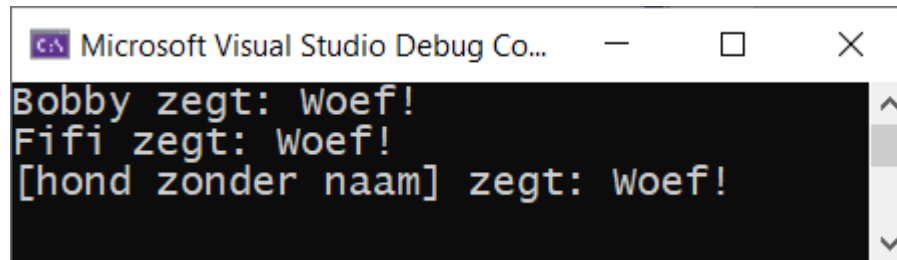


Honden-meeting

Oefening: Honden-meeting

Gebruik de **Hond** klasse uit het voorbeeld

- Maak 3 hond-objecten in Main()
 - De eerste hond noemt “Bobby”, de tweede “Fifi” en de derde heeft geen naam
- Zet alle honden in een List<Hond>
- Doe de honden elk via een lus spreken (foreach)



```
Microsoft Visual Studio Debug Co...  
Bobby zegt: Woef!  
Fifi zegt: Woef!  
[hond zonder naam] zegt: Woef!
```


Oefening Kippenhok

Klasse

Kip

```
naam: string  
eierenPerDag: short
```

```
LegEieren()
```

kip1

```
naam = "Louise"  
eierenPerDag = 2
```

Object



kip2

```
naam = "Louise"  
eierenPerDag = 1
```

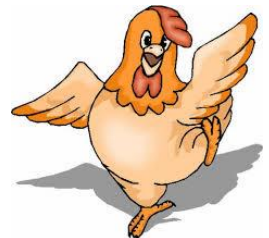
Object



kip3

```
naam = "Claire"  
eierenPerDag = 0
```

Object



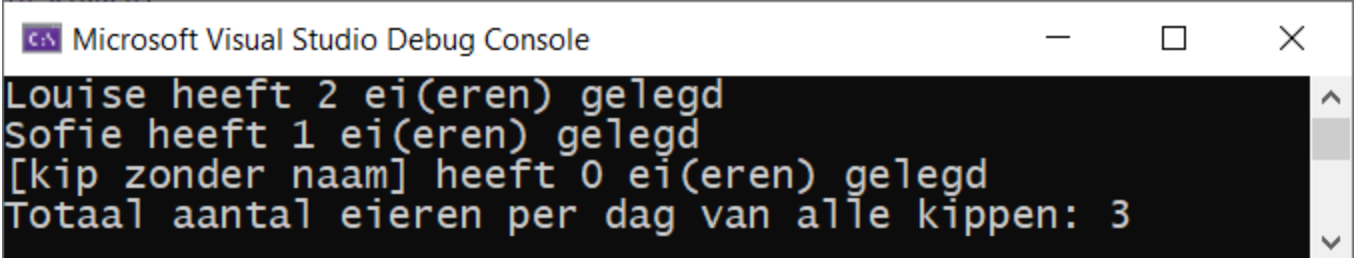
Oefening: Kippenhok

1. Definieer een klasse **Kip**

- Maak private fields `kleur` en `eierenPerDag`
- Maak methoden `SetNaam()`, `GetNaam()`, `SetEierenPerDag()` en `GetEierenPerDag()`. Om de fields te schrijven/lezen
- Maak methode `LegEieren()`, waarbij er naar de console wordt geschreven hoeveel eieren per dag de kip legt

2. Maak 3 kip objecten in `Main()`

- De eerste kip noemt “Louise”, de tweede “Sofie” en de derde heeft geen naam. Louise legt 2 eieren per dag, Sofie 1 per dag en de 3de legt geen eieren.
- Zet alle kippen in een `List<Kip>`
- Laat elke kip een ei leggen in een lus (foreach)
- Geef het totaal aantal eieren (bereken via de lus) die per dag zijn gelegd



```
Microsoft Visual Studio Debug Console

Louise heeft 2 ei(eren) gelegd
Sofie heeft 1 ei(eren) gelegd
[kip zonder naam] heeft 0 ei(eren) gelegd
Totaal aantal eieren per dag van alle kippen: 3
```



Constructoren

Definiëren en gebruiken van klasse Constructoren

Wat is een Constructor?

- **Constructoren** zijn speciaal soort methoden
 - Worden aangeroepen bij **creatie van een nieuwe instantie** van een klasseobject
 - Worden gebruikt om field-waarden te initialiseren
- Constructoren hebben dezelfde naam als de klasse
 - hebben **geen** terugkeertype (return type)
 - Kunnen parameters hebben
 - Kunnen zijn: **private, protected, internal, public**

Constructoren (2)

- Een klasse kan meerdere constructoren bezitten met 0 (parameterloze constructor), 1 of meerdere parameters

Constructoren voorbeelden

- Een Constructor wordt aangeroepen wanneer de **new** operator wordt gebruikt (bij creatie object van klasse)

Voorbeelden:

```
String s = new String(new char[] { 'a', 'b', 'c' });
```

```
String s = new String('*', 5); // s = "*****"
```

```
DateTime dt = new DateTime(2009, 12, 30);
```

```
DateTime dt = new DateTime(2009, 12, 30, 12, 33, 59);
```

```
Int32 value = new Int32();
```

Parameterloze Constructor

- De constructor zonder parameters is de standaard (**default**) constructor
- Voorbeeld:
 - Creatie van een object om willekeurige getallen te genereren met een default 'seed'

```
using System;  
...  
Random randomGenerator = new Random();
```

parameterloze
constructor
aanroep

System.Random klasse voorziet de
generatie van random getallen

Constructor met Parameters

- Voorbeeld Random klasse
 - Creatie van objecten om willekeurige getallen te genereren waarbij initiële 'seeds' gespecificeerd worden

```
using System;
...
Random randomGenerator1 = new Random(123);
Console.WriteLine(randomGenerator1.Next());
// 2114319875

Random randomGenerator2 = new Random(456);
Console.WriteLine(randomGenerator2.Next(50));
// 47
```


Zelf Constructoren definiëren in klasse



```
public class Hond
{
    private string naam;
    private string ras;

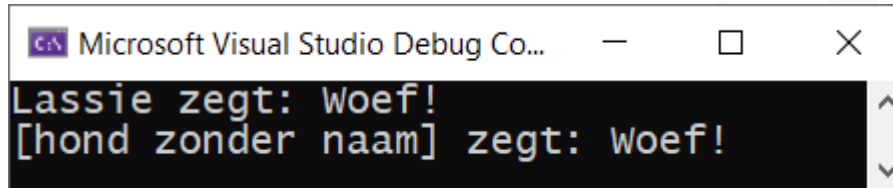
    public Hond() //parameterloze constructor
    {
        this.ras = "Onbekend";
    }

    public Hond(string naam, string ras) //constructor met 2
parameters
    {
        this.naam = naam;
        this.ras = ras;
    }
    public void Spreek()
    {
        Console.WriteLine("{0} zegt: Woef!",
            this.naam ?? "[hond zonder naam]");
    }
}
```

(vervolg op volgende slide...)

Zelf Constructoren definiëren (2)

```
class Program
{
    static void Main()
    {
        Hond hond1 = new Hond("Lassie", "Collie");
        hond1.Spreek();
        Hond hond2 = new Hond();
        hond2.Spreek();
    }
}
```



Microsoft Visual Studio Debug Co...
Lassie zegt: Woef!
[hond zonder naam] zegt: Woef!



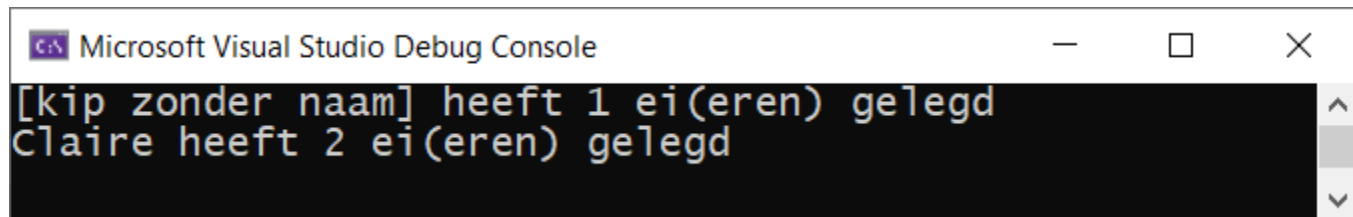


Constructoren

Demo

Oefening 1: Kip constructoren

1. Neem de laatste oefening van klasse **Kip**
 - Maak een parameterloze constructor die het aantalEierenPerDag initialiseert op 1
 - Maak een constructor met 2 parameters om de naam en het aantal eierenPerDag te initialiseren
2. Pas Main() aan
 - Verwijder de aanroepen naar SetNaam() en SetEierenPerDag()
 - Maak een kip object die de parameterloze constructor aanroept
 - Maak een 2de kip object die de constructor met 2 parameters aanroept. Geef naam Claire en aantal eieren per dag = 2 mee bij aanroep van deze constructor
 - Laat de 2 kip objecten een eieren leggen (via method-aanroep LegEieren())



```
Microsoft Visual Studio Debug Console
[kip zonder naam] heeft 1 ei(eren) gelegd
Claire heeft 2 ei(eren) gelegd
```

Oefening 2 – constructoren bij Klasse Student

Vul de code van klasse **Student** aan (zie volgende slide):

- Maak een parameterloze constructor die de klas initialiseert op “VDOC#1”
- Maak een constructor met 1 parameter naam die de naam instelt en ook klas op “VDOC#1” zet
- Maak een constructor met 4 parameters om alle fields van de klasse te initialiseren
- Voor Geslacht moet je eveneens enum aanmaken

Student

```
naam: string  
geslacht: Geslacht(enum)  
klas: string  
geboorteDatum: DateTime  
PringGegevens()
```

Klasse Student met 3 Constructoren (2)

```
public class Student
{
    private string naam;
    private DateTime geboorteDatum;
    private string klas;
    private Geslacht geslacht;
    // Maak Parameterloze constructor
    // Maak Constructor met 1 parameter naam
    // Maak Constructor met 4 parameters
```



```
public void PrintGegevens()
{
    Console.WriteLine("Student {0} :",this.naam ?? "[onbekende naam] ");
    string geboorte = (this.geboorteDatum == new DateTime(1, 1, 1)) ? "onbekend" : this.geboorteDatum.ToString("dd/MM/yyyy");
    Console.WriteLine($"Klas:{this.klas} - geboren op: {geboorte} - geslacht: {Enum.GetName(typeof(Geslacht),this.geslacht)} ");
    Console.WriteLine();
}
}
```

Oefening 2: Klasse Persoon met 2 Constructoren (2)

```
class Program
{
    static void Main()
    {
        Student student1 = new Student();
        student1.PrintGegevens();
        Student student2 = new Student("Jan");
        student2.PrintGegevens();
        Student student3 = new Student("Piet", new DateTime(2000, 10, 1), Geslacht.Man,
"Java1");
        student3.PrintGegevens();
    }
}
```

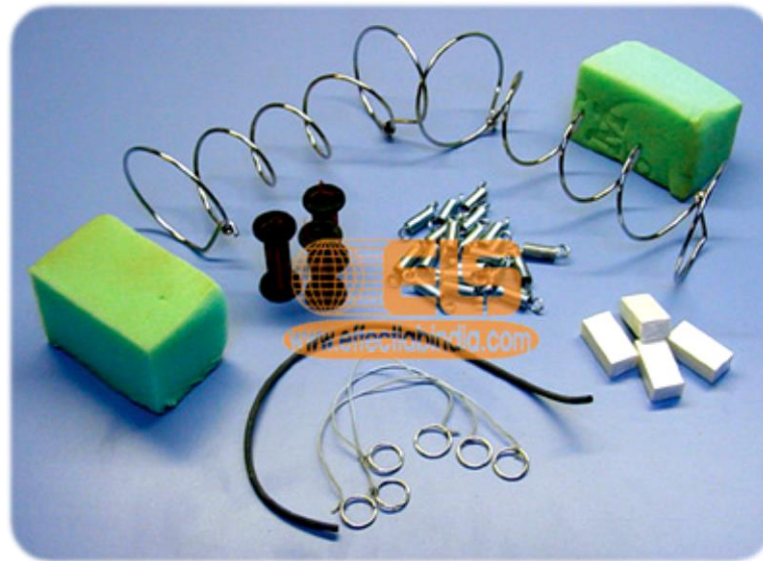


```
Microsoft Visual Studio Debug Console

Student [onbekende naam] :
Klas:VDOC#1 - geboren op: onbekend - geslacht: Man
Student Jan :
Klas:VDOC#1 - geboren op: onbekend - geslacht: Man
Student Piet :
Klas:Java1 - geboren op: 01/10/2000 - geslacht: Man
```

Eigenschappen (Properties)

Definiëren en gebruiken van Properties



Wat en Waarom Properties?

- **Properties** dienen om toegang te controleren naar de (private) fields van de klasse
 - Om de gegevens (toestand) van de objecten van de klasse te controleren en correct te houden
 - bv prijs field van klasse Artikel moet steeds positieve waarde hebben
- Via **Property** regelen:
 - Enkel leestoegang tot field
 - Enkel schrijftoegang tot field
 - Zowel lees-als schrijftoegang

Definiëren van Properties

- Leestoegang tot field: via `get`
- Schrijftoegang tot field: via `set`
- Properties hebben:
 - Access modifier (`public`, `protected`, ...)
 - Return type
 - Unieke naam (Meestal de naam van het field waarop ze de lees/schrijftoegang regelen, startend met hoofdletter)
 - `get` (leestoegang) en / of `set` (schrijftoegang) gedeelte
 - Kan code bevatten voor input validatie

Definiëren van Properties (2)

- Naar gelang de accessors get en set voor een property gedefinieerd is, geeft de property:
 - Enkel leestoegang (enkel **get** accessor)
 - Lees- en Schrijftoegang (zowel **get** en **set** accessors)
 - Enkel schrijftoegang (enkel **set** accessor)
- Voorbeeld van een read-only property:
 - **Length van String klasse**
- Voorbeeld van een read-write property
 - **Console.BackgroundColor**



Voorbeeld Property definities

```
public class Hond
{
    private string naam;
    private string ras;
    public Hond(string naam, string ras) //constructor met 2 parameters
    {
        this.naam = naam;
        this.ras = ras;
    }
    public string Naam //property met lees-en schrijftoegang
    {
        get { return this.naam; }
        set { this.naam = value; }
    }

    public string Ras // property met enkel leesttoegang
    {
        get { return this.ras; }
    }

    public void Spreek()
    {
        Console.WriteLine("{0} zegt: Woef!",
            this.naam ?? "[hond zonder naam]");
    }
}
```



Voorbeeld Property definitions

```
class Program
{
    static void Main(string[] args)
    {
        Hond hond1 = new Hond("Lassie", "Collie");
        Console.WriteLine("Naam :" + hond1.Naam);
        Console.WriteLine("Ras : " + hond1.Ras);
        hond1.Naam = "Fifi";
        //hond1.Ras = "pitbull"; // lukt niet, enkel leesttoegang
        hond1.Spreek();
    }
}
```

A screenshot of the Microsoft Visual Studio Debug Console window. The window title is "Microsoft Visual Studio Debug Co...". The output text is: "Naam :Lassie", "Ras : Collie", and "Fifi zegt: Woef!".

```
Microsoft Visual Studio Debug Co...
Naam :Lassie
Ras : Collie
Fifi zegt: Woef!
```

Oefening 1 Properties definiëren



Pas de klasse **Hond** aan:

- Maak een nieuwe private field leeftijd en een bijhorende public Property Leeftijd met lees-en schrijftoegang tot dit field
- Maak een nieuwe private field kleur (string) en bijhorende property Kleur met enkel leestoegang
- Pas de constructor aan zodat zowel leeftijd als kleur van de hond kunnen worden geïnitieerd bij creatie
- Maak in Main() een hond object. Gebruik de aangepaste constructor om hond “Bobby”, “Golden Retriever”, “beige” en leeftijd 2 jaar aan te maken.
- Lees properties Kleur en Leeftijd en schrijf de waarden naar de console
- Pas de leeftijd van de hond aan via property Leeftijd naar 3 jaar
- Vraag in de Main() daarna de Kleur en Leeftijd van de hond via gebruik van de properties en schrijf de waarden naar de console

Oefening 1 Properties definiëren (2)




```
public class Hond
{
    private string naam; //Voeg private fields kleur en leeftijd toe
    private string ras;
    public Hond(string naam, string ras) //Pas constructor aan, voeg 2
parameters toe voor leeftijd en kleur
    {
        this.naam = naam;
        this.ras = ras;
    }
    //voeg public property Kleur(enkel leestoegang) toe
    //voeg public property Leeftijd toe (lees-en schrijftoegang)
    public string Naam //property met lees-en schrijftoegang
    {
        get { return this.naam; }
        set { this.naam = value; }
    }

    public string Ras // property met enkel leestoegang
    {
        get { return this.ras; }
    }
}
```

Oefening 1 Properties definiëren (3)

```
class Program
{
    static void Main()
    {
        //Maak hond1 object aan door aanroep van de constructor
        //Schrijf Leeftijd naar console
        //Zet leeftijd op 3 (via Property Leeftijd)
        //Schrijf Hond gegevens naar console. Vraag de waarden op via de Properties
        Console.WriteLine(" Hond {0} ", hond1.Naam ?? "[hond zonder naam]");
        Console.WriteLine("\t--> Ras " + hond1.Ras);
        //Schrijf Kleur naar de console
        //Schrijf Leeftijd naar de console

    }
}
```



The screenshot shows a window titled "Microsoft Visual Studio Debug Console". The output text is as follows:

```
Leeftijd Bobby: 2
Hond Bobby
    --> Ras Golden Retriever
    --> Kleur beige
    --> Leeftijd 3 jaar
```



Dynamische Properties

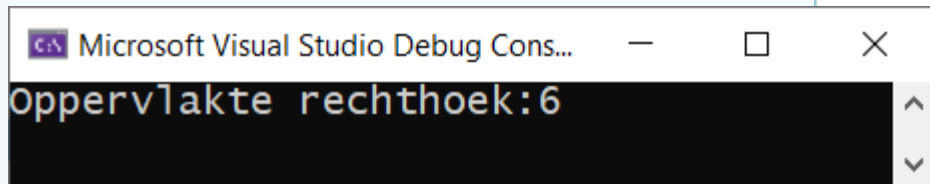
- Een Property kan ook een eenvoudige berekening geven op basis van 1 of meer fields. bv:

```
public class Rechthoek
{
    private double breedte;
    private double hoogte;

    public Rechthoek(double breedte, double hoogte)
    {
        this.breedte = breedte;
        this.hoogte = hoogte;
    }

    public double Oppervlakte //dynamische property (read-only)
    {
        get
        {
            return breedte * hoogte;
        }
    }
}

class Program
{
    static void Main()
    {
        Rechthoek rechthoek1 = new Rechthoek(2.0, 3.0);
        Console.WriteLine("Oppervlakte rechthoek:" + rechthoek1.Oppervlakte);
    }
}
```



Oefening 2 Properties definiëren



Pas de klasse **Student** aan:

- Maak een public property Naam met lees-en schrijftoegang naar het field naam
- Maak een public property GeboorteDatum met enkel leestoegang naar het field geboorteDatum
- Maak een public Dynamische property Leeftijd met enkel leestoegang die de leeftijd in Jaren teruggeeft
- Maak in Main() een studente aan via de constructor met naam “Joke” met geboortedatum 1 April 2002, geslacht Vrouw en Klas “AVC#1”
- Vraag in de Main() daarna de Naam , Geboortedatum en Leeftijd van de studente op via gebruik van de properties en schrijf de waarden naar de console
- Pas de naam van de studente aan naar “José” Main() en print nogmaals de gewijzigde naam van de studente naar de console

Oefening 2 Properties definiëren (2)



```
public enum Geslacht
{
    Man,
    Vrouw
}
public class Student
{
    private string naam;
    private DateTime geboorteDatum;
    private string klas;
    private Geslacht geslacht;
    // constructor met 4 parameters
    public Student(string naam, DateTime geboorteDatum, Geslacht geslacht,
string klas)
    {
        this.naam = naam;
        this.geboorteDatum = geboorteDatum;
        this.geslacht = geslacht;
        this.klas = klas;
    }
}
```

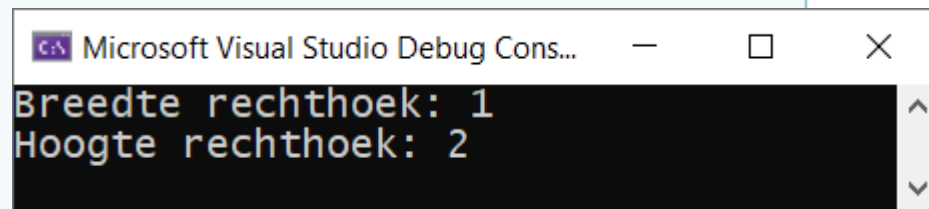
Microsoft Visual Studio Debug Console

```
Student Joke :
    geboren op: 01/04/2002 - leeftijd: 18 jaar
Student naam is nu: José
```

Automatische Properties

- Properties waarvoor de fields automatisch worden aangemaakt door de compiler

```
public class Rechthoek
{
    public double Breedte {get; set;} //Auto-property (onderliggend field
    wordt automatisch aangemaakt)
    public double Hoogte { get; set; } //Auto-property
    public Rechthoek(double breedte, double hoogte)
    {
        this.Breedte = breedte;
        this.Hoogte = hoogte;
    }
}
class Program
{
    static void Main()
    {
        Rechthoek rechthoek1 = new Rechthoek(2.0, 3.0);
        rechthoek1.Breedte = 1.0;
        rechthoek1.Hoogte = 2.0;
        Console.WriteLine("Breedte rechthoek: " + rechthoek1.Breedte);
        Console.WriteLine("Hoogte rechthoek: " + rechthoek1.Hoogte);
    }
}
```



Microsoft Visual Studio Debug Cons...

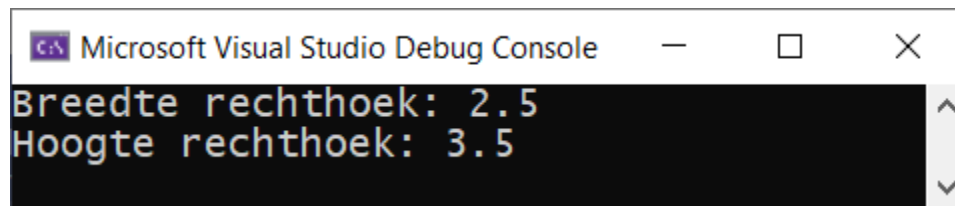
```
Breedte rechthoek: 1
Hoogte rechthoek: 2
```

Uitgebreide object Initializers

- In een *object initializer* kan je tussen accolades public Properties instellen

```
public class Rechthoek
{
    public double Breedte { get; set; }
    public double Hoogte { get; set; }
}

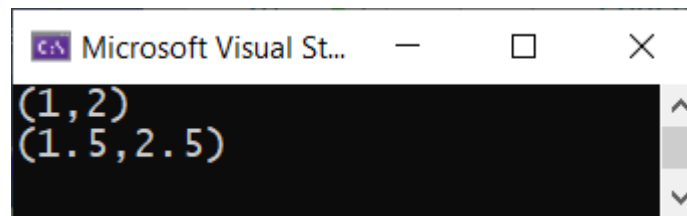
class Program
{
    static void Main()
    {
        Rechthoek rechthoek1 = new Rechthoek() { Breedte = 2.5, Hoogte = 3.5 };
        Console.WriteLine("Breedte rechthoek: " + rechthoek1.Breedte);
        Console.WriteLine("Hoogte rechthoek: " + rechthoek1.Hoogte);
    }
}
```



Oefening Auto- Property en object initializer

Definieer een klasse **Punt** :

- Maak public auto-properties X en Y met lees-en schrijftoegang
- Definieer geen constructor, de default parameterloze constructor wordt dan automatisch aangemaakt door de compiler
- Maak in Main() een Punt aan via object initializer: `Punt punt = new Punt() {X=1.0,Y=2.0};`
- Schrijf het Koppel (X,Y) van het punt naar de console
- Zet in Main() via de Auto-Properties `X = 1.5` en `Y = 2.5`
- Schrijf nogmaals het Koppel (X,Y) van het punt naar de console



```
Microsoft Visual St...  
(1, 2)  
(1.5, 2.5)
```

Object status correct houden –validatie

```
public class Persoon
{
    private string naam;
    public Persoon(string naam)
    {
        this.Naam = naam;
    }
    public string Naam
    {
        get { return this.naam; }
        set
        {
            if (String.IsNullOrEmpty(value))
                throw new ArgumentException("Ongeldige naam!");
            this.naam = value;
        }
    }
}

class Program
{
    static void Main()
    {
        Persoon persoon = new Persoon(""); //lukt niet Foutmelding bij ongeldige naam
        persoon.Naam = ""; //lukt niet, Foutmelding bij ongeldige naam
    }
}
```

In constructor wordt
property ingevuld

Een ongeldige naam kan
niet worden toegekend

leren. durven. doen.



Vragen?

