

leren. durven. doen.



C# FUNDAMENTALS

OOP OVERERVING (INHERITANCE)



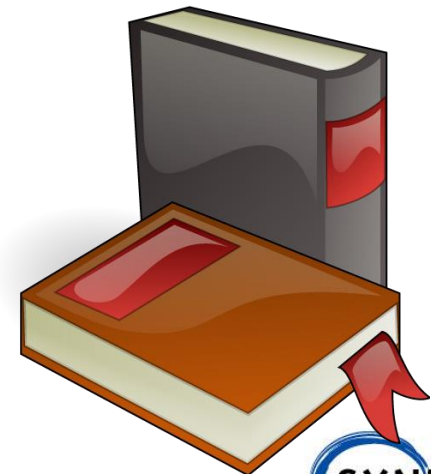
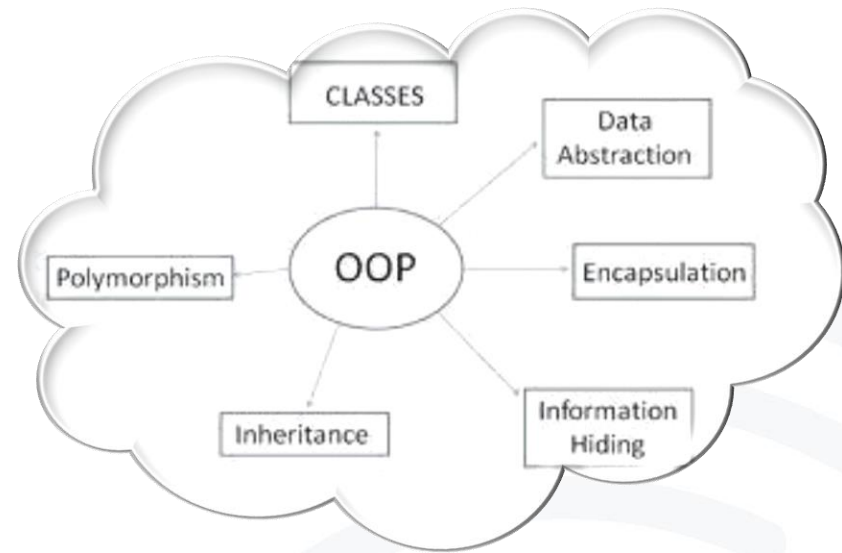
Object-georiënteerd Programmeren (OOP) Overerving



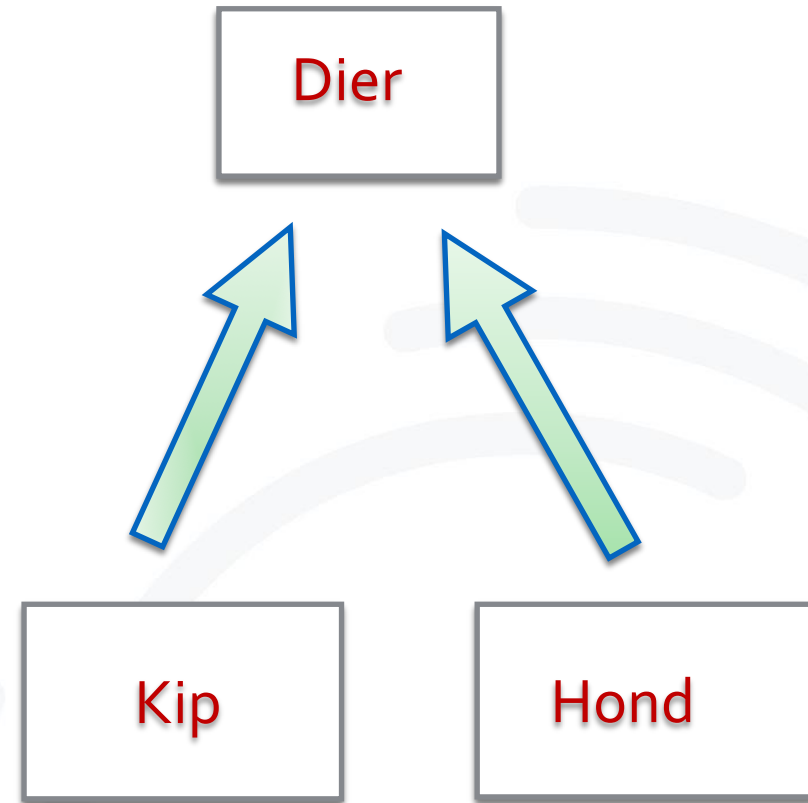
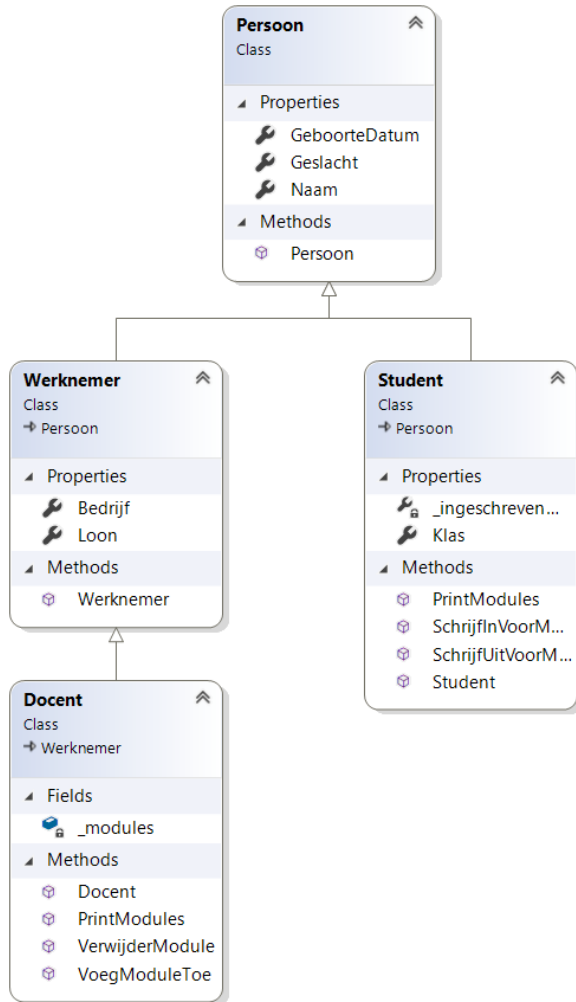
Inhoud

Overerving (Inheritance)

- Wat is Overerving
 - Inheritance – Voordelen
 - Overervingstypen
 - Classes versus Interfaces
 - Klasse-Hierarchieën
 - Toegangsniveau's
- Overerving en Toegankelijkheid
- Overerving : Belangrijke Aspecten
- Overerving : Method hiding
- Overerving: virtual en override



Overerving



Wat is Overerving

- Via overerving kan je één klasse baseren op een andere klasse.
- Bij **Klasse-overerving** worden **alle members** van de basisklasse (ouder) impliciet overgenomen in de afgeleide klasse (kind)
 - Alle fields, methods, properties, ...
 - Sommige members kunnen worden verborgen (hidden)
- De klasse waarvan wordt overgeërfd is de **base, basisklasse** (parent) klasse (bv. klasse Dier)
- De class die overerft is **derived, afgeleide** (child) klasse (bv. klasse Hond)

Overerving – Voorbeeld

Zowel kip als hond zijn dieren
en erven m.a.w. alle eigenschappen en gedrag van type Dier
Dier is de basis (base) klasse en Kip en Hond zijn afgeleide klassen
van de klasse Dier



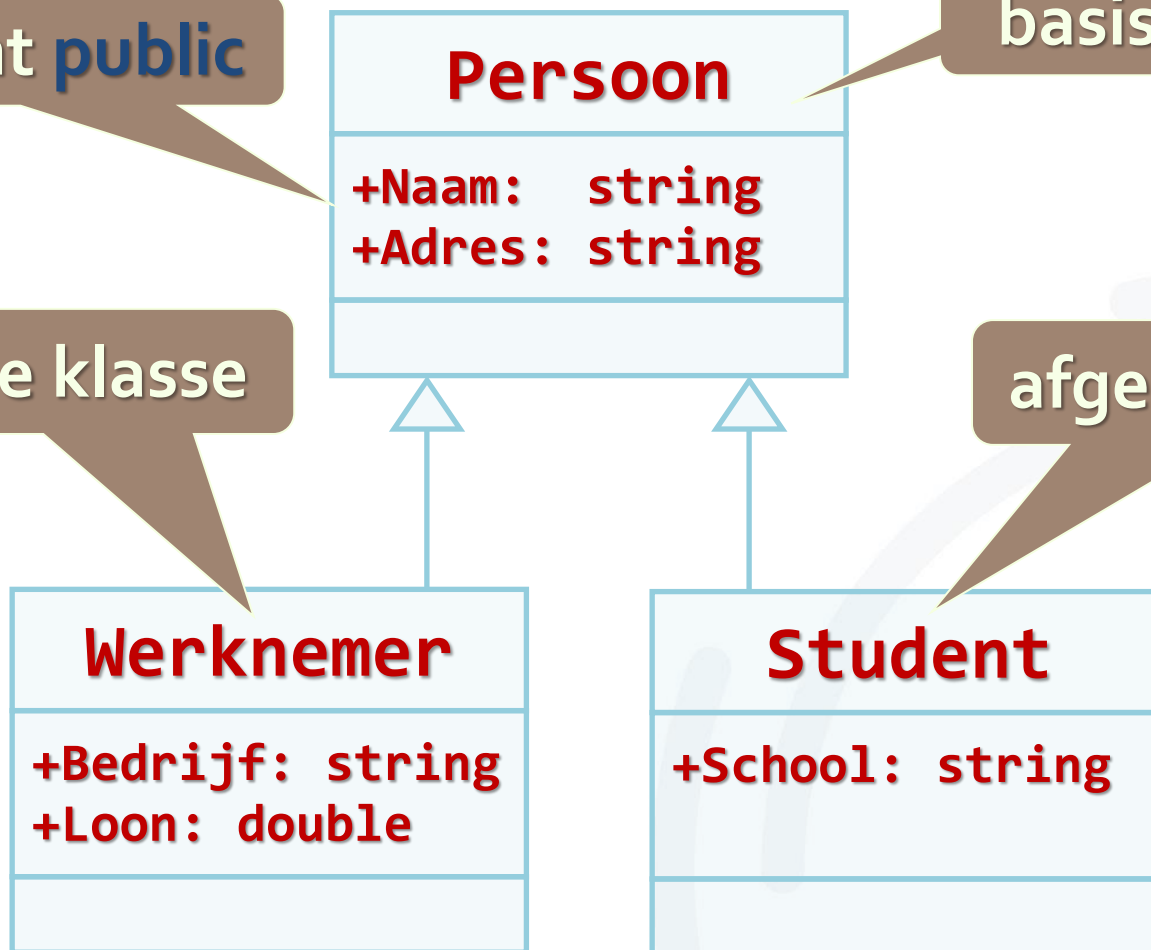
Overerving – Voorbeeld afgeleide klassen van basis klasse person (uml diagram)

+ betekent **public**

basis klasse

afgeleide klasse

afgeleide klasse



Overerving – Voorbeeld afgeleide klassen van basis klasse person (code)

```
class Persoon
{
    public string Naam {get;set;}
    public string Adres { get; set; }
}
class Werknemer : Persoon
{
    public string Bedrijf { get; set; }
    public double Loon { get; set; }
}
class Student : Persoon
{
    public string School { get; set; }
}
```


Voordelen van overerving

- **Overerving heeft meerdere voordelen**
 - Uitbreidbaarheid
 - Herbruikbaarheid (hergebruik van code)
 - Voorzien van abstractie
 - Overbodige code vermijden
- **Gebruik inheritance voor “is-een” relatie**
 - Bv. Hond “is-een” Dier
- **Niet gebruiken voor “heeft-een” relatie**
 - Bv. Hond “heeft-een” leeftijd



Overerving in .NET

- Een klasse kan van slechts 1 basis klasse overerven (**enkelvoudige overerving**)

Bv. Klass *Hond* is afgeleide (derived) Klasse van *Dier*

- Een klasse/interface kan meerdere interfaces implementeren

Dit is de manier van .NET om “soort van” **meervoudige overerving** te voorzien

Klasse Overerving in C#

- Specificeer de naam van de basis klasse achter de naam van de afgeleide (afgeleide) klasse (met : ertussen)

```
public class Figuur  
{ ... }  
public class Cirkel : Figuur  
{ ... }
```



- Gebruik indien nodig het keyword **base** om de constructor van de parent aan te roepen
- Bv constructor van Cirkel roept constructor van Figuur aan via : base(...)

```
public Cirkel (int x, int y) : base()  
{ ... }
```



Oververving- basisklasse Dier

```
public class Dier
{
    private string kleur;
    public Dier(string kleur)
    {
        this.kleur = kleur;
    }
    public string Kleur
    {
        get { return this.kleur; }
    }
    public void Spreek()
    {
        Console.WriteLine("Dier spreekt");
    }
}
```

Klasse Dier

Field

Constructor

Property

Methode

Oververving- Hond is afgeleid van Dier

afgeleide klasse

basisklasse

```
public class Hond : Dier
{
```

```
    private string naam;
    private string ras;
```

```
    public Hond() : base("onbekende kleur") { }
```

```
    public Hond(string kleur) : base(kleur) { }
```

```
    public Hond(string naam, string ras, string kleur) : base(kleur)
    {
```

```
        this.naam = naam;
        this.ras = ras;
    }
```

```
    public string Naam
    {
```

```
        get { return this.naam; }
        set { this.naam = value; }
    }
```

```
    public string Ras
    {
```

```
        get { return this.ras; }
    }
```

Aanroep constructor
Dier klasse

3 Hond-Constructors

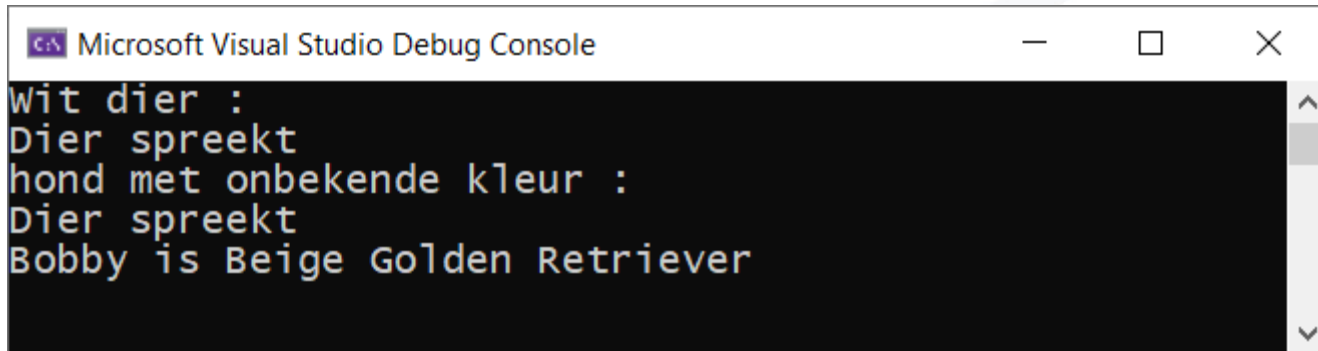
2 Properties van
Hond

Klasse overerving- Voorbeeld Dier en Hond (vervolg)

```
class Program
{
    static void Main()
    {
        Dier dier = new Dier("Wit");
        Console.WriteLine($"{dier.Kleur} dier :");
        dier.Spreek();

        Hond hond1 = new Hond();
        Console.WriteLine($"hond met {hond1.Kleur} :");
        hond1.Spreek();

        Hond hond2 = new Hond("Bobby", "Golden Retriever", "Beige");
        Console.WriteLine($"{hond2.Naam} is {hond2.Kleur} {hond2.Ras}:");
    }
}
```



Microsoft Visual Studio Debug Console

```
Wit dier :
Dier spreekt
hond met onbekende kleur :
Dier spreekt
Bobby is Beige Golden Retriever
```

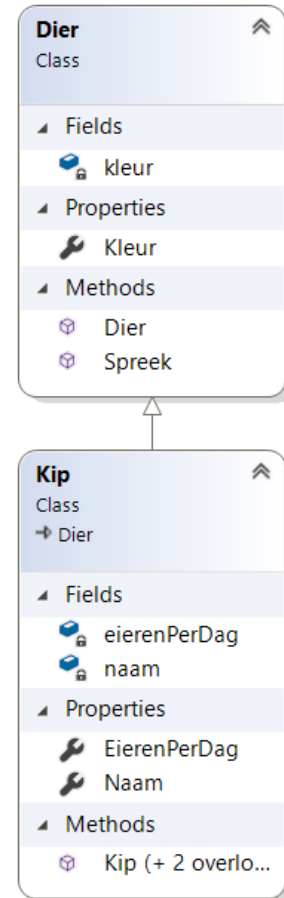


Oefening afgeleide klasse Kip van Dier

- **Neem de code van de klasse Dier en Maak een nieuwe afgeleide klasse Kip van Dier. Klasse Dier mag niet worden aangepast**
- **De klasse Kip heeft**
 - 2 private fields **naam** en **eierenPerDag**
 - 2 public Properties:
Naam met lees-en schrijftoegang naar field naam
EierenPerDag met enkel leestoegang naar field eierenPerDag
 - 3 constructors:
 1. Een parameterloze constructor die de constructor van Dier aanroept met de tekst "onbekende kleur"
 2. Een constructor met 1 parameter kleur die de constructor van Dier met deze parameter aanroept
 3. Een constructor met 3 parameters naam, eierenPerDag en kleur die de constructor van Dier aanroept

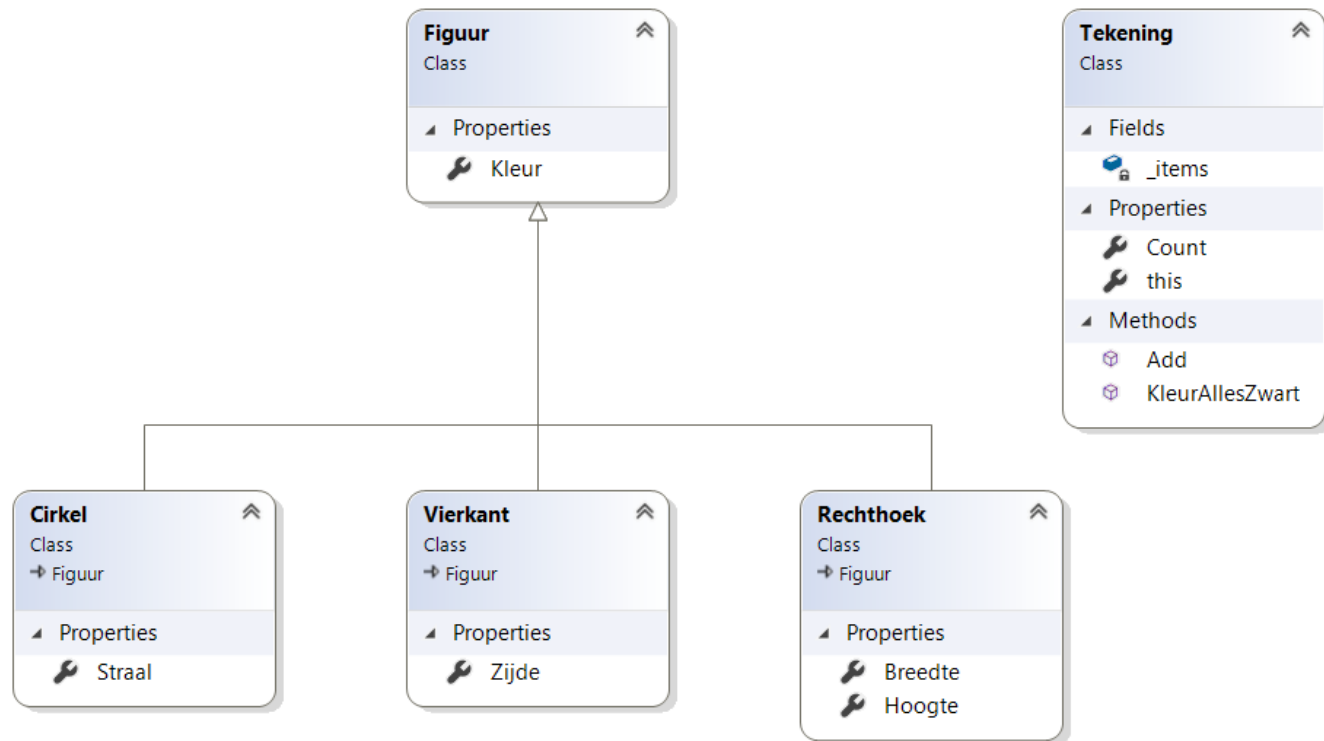
Voeg code toe aan de Main() die het volgende resultaat geeft op de console (pas voorbeeldcode vorige slide aan)

```
Microsoft Visual Studio Debug Console
Zwart dier :
Dier spreekt
kip met onbekende kleur :
Dier spreekt
Claire is Bruin en legt 2 ei(eren) per dag
```



Wanneer overerving gebruiken - Voorbeeld

- We hebben een tekening met verschillende figuren
- Een tekening “**heeft**” 1 of meer figuren (gebruik field/property)
- Een Cirkel, Rechthoek en Vierkant “**zijn**” Figuren (gebruik overerving)



Voorbeeld Tekening met Figuren (2)

```
class Figuur
{
    public string Kleur { get; set; }
}
class Vierkant : Figuur //Vierkant "is een" figuur
{
    public double Zijde { get; set; }
}

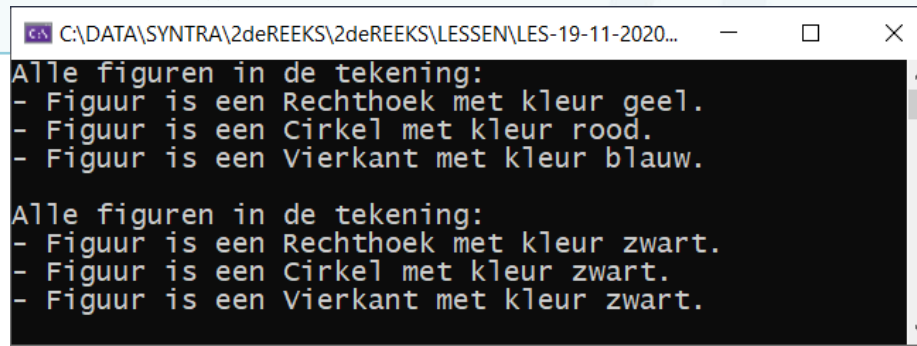
class Rechthoek : Figuur //Rechthoek "is een" figuur
{
    public double Breedte { get; set; }
    public double Hoogte { get; set; }
}
class Cirkel : Figuur //Cirkel "is een" figuur
{
    public double Straal { get; set; }
}
```

Voorbeeld Tekening met Figuren (3)

```
class Tekening
{
    private List<Figuur> _figuren; //Tekening "heeft" 1 of meer figuren
    public Tekening()
    {
        _figuren = new List<Figuur>();
    }
    public Figuur this[int index]
    {
        get { return _figuren[index]; }
        set { _figuren[index] = value; }
    }
    public int Count
    {
        get { return _figuren.Count; }
    }
    public void Add(Figuur item)
    {
        _figuren.Add(item);
    }
    public void KleurAllesZwart()
    {
        for (int index = 0; index < Count; index++)
            this[index].Kleur = "zwart";
    }
}
```

Voorbeeld Tekening met Figuren (2)

```
class Program
{
    static void Main()
    {
        Rechthoek r1 = new Rechthoek { Hoogte = 5d, Breedte = 4d, Kleur = "geel" };
        Cirkel c1 = new Cirkel { Straal = 10d, Kleur = "rood" };
        Vierkant v1 = new Vierkant { Zijde = 15d, Kleur = "blauw" };
        Tekening tekening1 = new Tekening();
        tekening1.Add(r1);
        tekening1.Add(c1);
        tekening1.Add(v1);
        Print(tekening1);
        tekening1.KleurAllesZwart();
        Print(tekening1);
        Console.ReadLine();
    }
    static void Print(Tekening fn)
    {
        Console.WriteLine("Alle figuren in de tekening: ");
        for (int index = 0; index < fn.Count; index++)
        {
            Figuur f = fn[index];
            string naamNameSpace = f.GetType().Namespace;
            string naamAfgeleideType = f.GetType().ToString().Replace(naamNameSpace + ".", "");
            string basisType = f.GetType().BaseType.Name;
            Console.WriteLine($"{basisType} is een {naamAfgeleideType} met kleur {f.Kleur}.");
        }
        Console.WriteLine();
    }
}
```

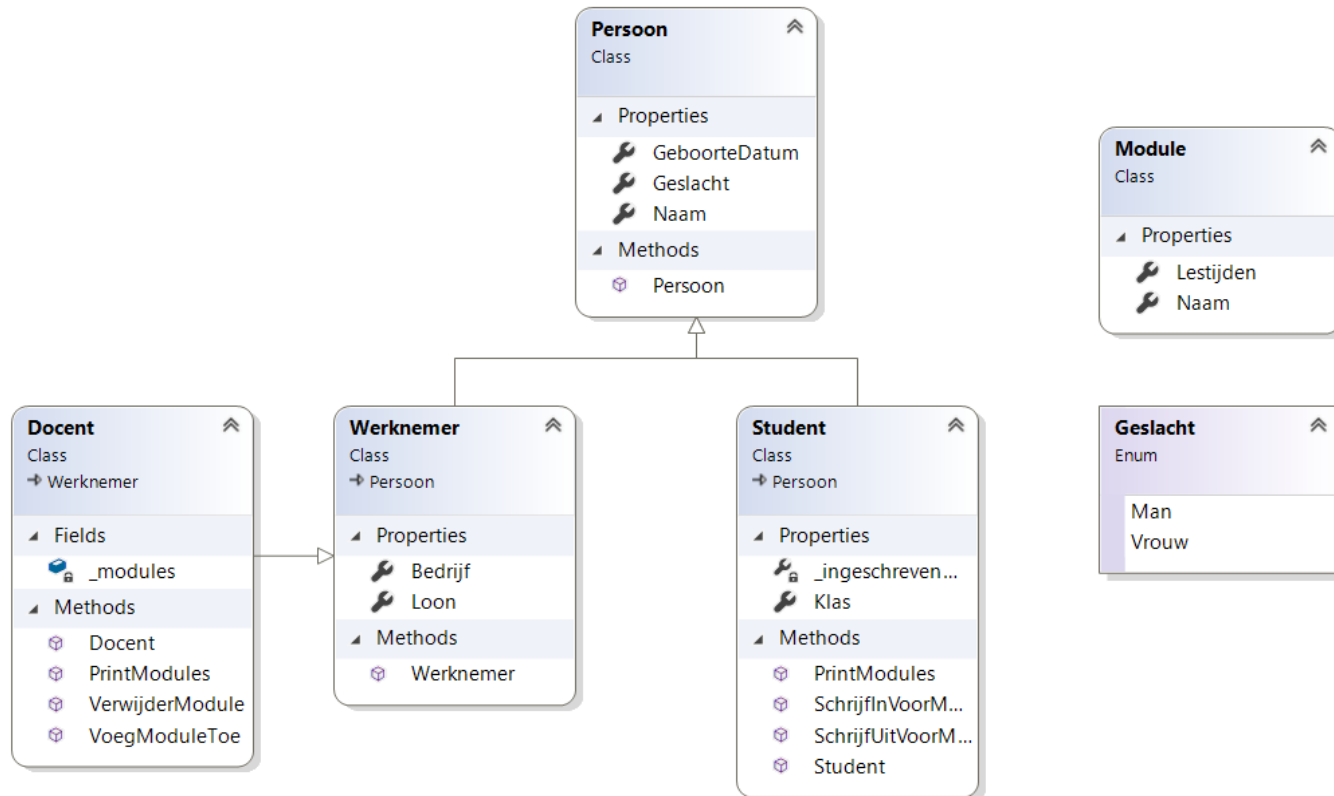


```
C:\DATA\SYNTRA\2deREEKS\2deREEKS\LESSEN\LES-19-11-2020...
Alle figuren in de tekening:
- Figuur is een Rechthoek met kleur geel.
- Figuur is een Cirkel met kleur rood.
- Figuur is een Vierkant met kleur blauw.

Alle figuren in de tekening:
- Figuur is een Rechthoek met kleur zwart.
- Figuur is een Cirkel met kleur zwart.
- Figuur is een Vierkant met kleur zwart.
```

Oefening klassen, “heeft een” en “is een” relaties

- Maak basisklassen en afgeleide klassen volgens het diagram
Gebruik de code op de volgende 2 slides om objecten van de klassen aan te maken en aan te spreken



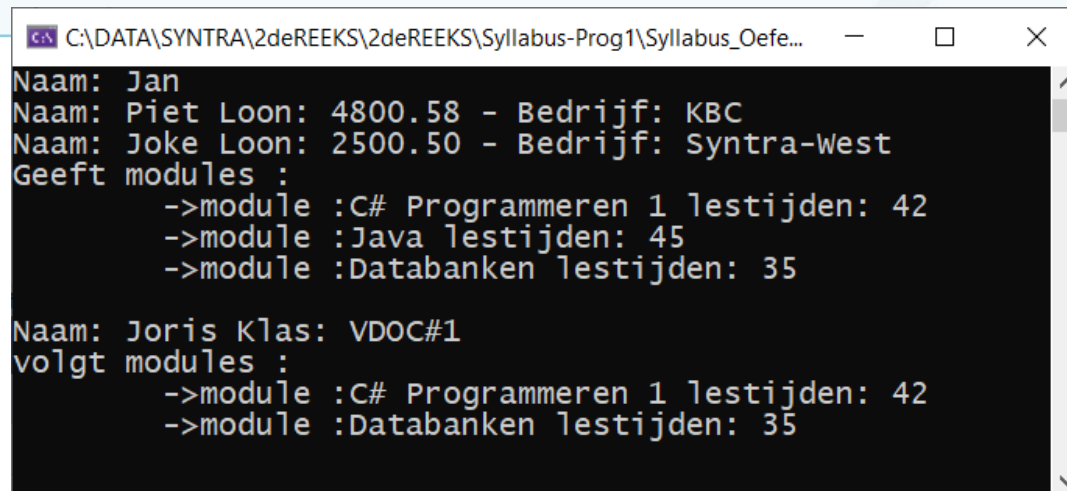
Oefening overerving klassen (2)

```
class Program
{
    static void Main()
    {
        Persoon p1 = new Persoon("Jan", new DateTime(2005, 10, 12), Geslacht.Man);
        Print(p1);
        Module m1 = new Module() { Naam = "C# Programmeren 1", Lestijden = 42 };
        Module m2 = new Module() { Naam = "Java", Lestijden = 45 };
        Module m3 = new Module() { Naam = "Databanken", Lestijden = 35 };
        Werknemer w1 = new Werknemer("Piet", "KBC", 4800.58m, new DateTime(2000, 4, 1), Geslacht.Man);
        Print(w1);
        Docent d1 = new Docent("Joke", "Syntra-West", 2500.50m, new DateTime(2003, 9, 15), Geslacht.Vrouw);
        d1.VoegModuleToe(m1);
        d1.VoegModuleToe(m2);
        d1.VoegModuleToe(m3);

        Print(d1);
        Student s1 = new Student("Joris", "VDOC#1", new DateTime(2002, 5, 1), Geslacht.Man);
        s1.SchrijfInVoorModule(m1);
        s1.SchrijfInVoorModule(m2);
        s1.SchrijfInVoorModule(m3);
        s1.SchrijfUitVoorModule(m2);
        Print(s1);
        Console.ReadLine();
    }
}
```

Oefening overerving klassen (3)

```
static void Print(Persoon persoon)
{
    Console.WriteLine($"Naam: {persoon.Naam} ");
    if (persoon is Werknemer)
    {
        Werknemer w = persoon as Werknemer;
        Console.WriteLine($"Loon: {w.Loon} - Bedrijf: {w.Bedrijf}");
    }
    if (persoon is Docent)
    {
        Docent d = persoon as Docent;
        Console.WriteLine($"Geeft modules : ");
        d.PrintModules();
    }
    if (persoon is Student)
    {
        Student c = persoon as Student;
        Console.WriteLine($"Klas: {c.Klas} ");
        Console.WriteLine($"nvolgt modules : ");
        c.PrintModules();
    }
    Console.WriteLine();
}
```



```
C:\DATA\SYNTRA\2deREEKS\2deREEKS\Syllabus-Prog1\Syllabus_Oefe...
Naam: Jan
Naam: Piet Loon: 4800.58 - Bedrijf: KBC
Naam: Joke Loon: 2500.50 - Bedrijf: Syntra-West
Geeft modules :
    ->module :C# Programmeren 1 lestijden: 42
    ->module :Java lestijden: 45
    ->module :Databanken lestijden: 35

Naam: Joris Klas: VDOC#1
volgt modules :
    ->module :C# Programmeren 1 lestijden: 42
    ->module :Databanken lestijden: 35
```

Overervingsmogelijkheden

- **Overerving** laat toe dat afgeleide klassen kenmerken van de basisklasse kunnen overnemen
 - Attributen (fields en properties)
 - Operaties/gedrag (methods)
- **Afgeleide klassen kunnen de basisklasse uitbreiden**
 - nieuwe fields en methods toevoegen
 - Herdefiniëren van bestaande methods (wijzigen van bestaand gedrag/operaties)

Voorbeeld uitbreiden gedrag van basisklasse

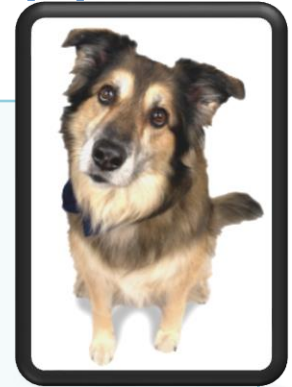
```
public class Zoogdier
{
    public int Leeftijd { get; set; }

    public Zoogdier(int leeftijd)
    {
        this.Leeftijd = leeftijd;
    }

    public void Slaap()
    {
        Console.WriteLine("Ssst! Ik slaap!");
    }
}
```



Voorbeeld Uitbreiding gedrag basisklasse (2)



```
public class Hond : Zoogdier
{
    public string Ras { get; set; }

    public Hond(int leeftijd, string ras)
        : base(leeftijd)
    {
        this.Ras = ras;
    }

    public void Kwispel()
    {
        Console.WriteLine("Aan het kwispelen...");
    }
}
```

Overerving

Demo



Overerving en toegankelijkheid

Access modifiers in C#

- **public** — toegankelijk vanuit alle klassen
- **private** — enkel toegankelijk in klasse zelf
- **protected** — toegankelijk in klasse zelf en zijn afgeleide klassen
- **internal** — (default) – toegankelijk binnen de eigen assembly, d.w.z. het huidig VS project
- **protected internal** — toegang is beperkt tot huidige assembly en afgeleide klassen van de huidige klasse

Overerving en Toegankelijkheid - voorbeeld

```
class Dier
{
    protected string Naam { get; set; }
    protected int Leeftijd { get; private set; }
    private void Spreek()
    {
        Console.WriteLine("Dier zegt: joepie ik ben een dier ...");
    }
    protected void Beweeg()
    {
        Console.WriteLine("Dier beweegt ...");
        this.Spreek();
    }
}
class Zoogdier : Dier
{
    public int AantalMaandenDrachtig { get; set; }

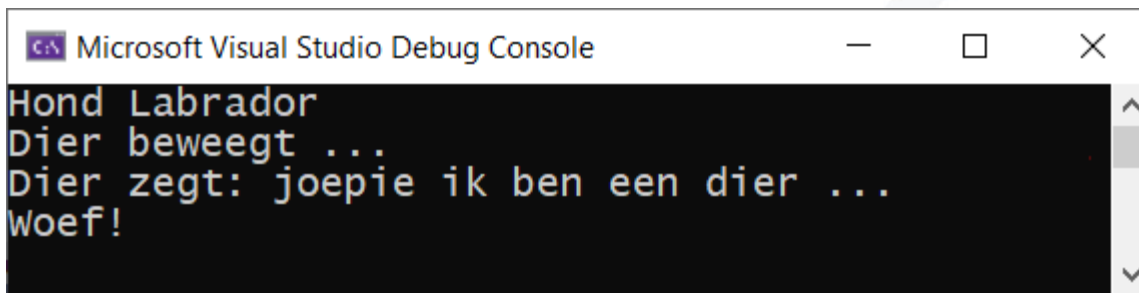
    // base.Spreek() kan niet worden aangeroepen (private)
    // this.Naam kan hier gelezen en gewijzigd worden (protected)
}
```

Overerving en Toegankelijkheid – voorbeeld(2)

```
class Hond : Zoogdier
{
    public string Ras { get; private set; }
    public Hond(string naam, string ras)
    {
        this.Naam = naam; // ok, Naam heeft protected set in parent klasse Dier
        this.Ras = ras;
        //this.Leeftijd = 2; // kan niet worden gewijzigd: private set in klasse Dier
    }
    public void Blaf()
    {
        // base.Spreek(); // kan niet worden aangeroepen(private bij klasse Dier)
        Console.WriteLine("Woef!");
    }
    public void Wandel()
    {
        base.Beweeg(); // ok Beweeg is protected in klasse Dier
    }
}
```

Overerving en Toegankelijkheid – voorbeeld (3)

```
class Program
{
    static void Main()
    {
        Hond hond = new Hond("Fifi", "Labrador");
        hond.AantalMaandenDrachtig = 6; //ok public property met set in klasse Zoogdier
        Console.WriteLine("Hond " + hond.Ras); //ok leestoegang tot Property Ras
        hond.Wandel(); //ok is public in klasse Hond
        hond.Blaf(); //ok is public in klasse Hond
        //hond.Beweeg(); //is protected kan hier niet worden aangeroepen
        //hond.Spreek() //is private in klasse Dier kan hier niet worden aangeroepen
        //hond.Naam = "Fifi"; // Naam is protected klasse Dier, kan hier niet worden aangeroepen
        //hond.Ras = "pitbull"; // Ras kan niet worden gewijzigd (private set in klasse Hond)
    }
}
```



```
Microsoft Visual Studio Debug Console
Hond Labrador
Dier beweegt ...
Dier zegt: joepie ik ben een dier ...
Woef!
```

Oefening Overerving en Toegankelijkheid

Maak een nieuwe afgeleide klasse Kat van Zoogdier:

- **Maak een (auto) property AantalMuizenPerDag die publiek leesbaar is maar enkel binnen de klasse Kat kan worden gewijzigd**
- **Maak een Kat constructor die de naam en aantalmuizen per dag initialiseert die in de Main() methode kan worden aangeroepen**
- **Maak een public methode Miauw() die de tekst “Miauw” naar de console schrijft**
- **Maak een public methode Speel() die de methode Beweeg() van de basisklasse Dier aanroept**

```
class Program
{
    static void Main()
    {
        Kat kat = new Kat("Minou", 2);
        kat.AantalMaandenDrachtig = 5;
        Console.WriteLine("Kat vangt " + kat.AantalMuizenPerDag + " muizen per dag");
        kat.Miauw();// Lukt dit? waarom(niet)?
        kat.Speel(); // Lukt dit? waarom(niet)?
        kat.Beweeg();//// Lukt dit? waarom(niet)?
        kat.Spreek();// Lukt dit? waarom(niet)?
        kat.Naam = "Minou"; // Lukt dit? waarom(niet)?
        kat.AantalMuizenPerDag = 3; //mag niet lukken
    }
}
```

Overerving en Toegankelijkheid

Demo



Inheritance: Belangrijke Aspecten

- **Structures** kunnen niet worden overgeërfd

- In C# is er geen echte **multiple** inheritance

Enkel meerdere interfaces kunnen worden geïmplementeerd

- **Static members** worden ook overgeërfd

- **Constructors** worden niet overgeërfd

- **overerving** is een **transitieve** relatie

Als C is afgeleid is van B en B is afgeleid van A, dan erft C ook over van A

Overerving - Method hiding

In (method) hiding wordt in een afgeleide klasse een nieuwe member toegevoegd die dezelfde naam (en parameters) heeft als een member die wordt overgeërfd uit de basisklasse.

Vanaf de afgeleide klasse is de verborgen member uit de basisklasse niet meer bruikbaar.

Method hiding wordt ook *method shadowing* genoemd

Voorbeeld:

```
class Dier
{
    0 references
    public string Naam { get; set; }
    0 references
    public void Spreek()
    {
        Console.WriteLine("Dier zegt: joepie ik ben een dier ...");
    }
    0 references
    protected void Beweeg()
    {
        Console.WriteLine("Dier beweegt ...");
    }
}
1 reference
class Hond:Dier
{
    0 references
    public string Naam { get; set; }
    0 references
    public void Beweeg()
    {
        Console.WriteLine("Hond beweegt ...");
    }
}
```

CS0108: 'Hond.Naam' hides inherited member 'Dier.Naam'. Use the new keyword if hiding was intended.

Overerving - Method hiding- voorbeeld (2)

```
class Dier
{
    0 references
    public string Naam { get; set; }
    0 references
    public void Spreek()
    {
        Console.WriteLine("Dier zegt: joepie ik ben een dier ...");
    }
    0 references
    protected void Beweeg()
    {
        Console.WriteLine("Dier beweegt ...");
    }
}
```

```
class Hond:Dier
```

```
{
    0 references
    public string Naam { get; set; }
}
```

```
0 references
public void Spreek()
```

```
{
    Console.Wr
```

void Hond.Spreek()

CS0108: 'Hond.Spreek()' hides inherited member 'Dier.Spreek()'. Use the new keyword if hiding was intended.

[Show potential fixes \(Ctrl+;\)](#)

virtual en override keywords

- **Wanneer een afgeleide klasse B zijn basisklasse (parent) A uitbreidt:**
 - B kan nieuwe members (fields, properties, methoden,...) toevoegen
 - B kan niet de overgeërfde members verwijderen
- **Nieuwe members in B met dezelfde naam of signatuur als A **verbergt automatisch** de members van B**
- **Een klasse kan **virtuele** methoden en properties declareren:**
 - Afgeleide classes kunnen een **override** van de implementatie van de members voorzien indien nodig
 - Bv. **Object.ToString()** is een **virtual** methode


Voorbeeld virtual en override

```
class Dier
{
    public string Naam { get; set; }
    public int Leeftijd { get; private set; }
    public virtual void Spreek()
    {
        Console.WriteLine("Dier zegt: joepie ik ben een dier ...");
    }
}

class Zoogdier : Dier
{
    public int AantalMaandenDrachtig { get; set; }
    public override void Spreek()
    {
        Console.WriteLine("Dier zegt: hallo ik ben een zoogdier ...");
    }
}
```

Voorbeeld virtual en override (2)

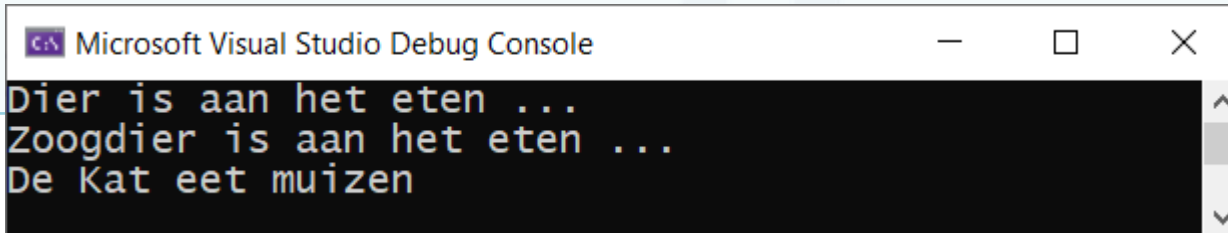
```
class Hond : Zoogdier
{
    public string Ras { get; private set; }
    public Hond(string naam, string ras)
    {
        this.Naam = naam;
        this.Ras = ras;
    }
    public override void Spreek()
    {
        Console.WriteLine("Woef!");
    }
}
```



Oefening virtual en override

- Neem de code van de klasse Dier en Zoogdier en Maak een nieuwe afgeleide klasse Kat van Zoogdier. Voeg zowel aan klasse Dier, Zoogdier en Kat de methode Eet() toe. Zet op de juiste plaats de keywords virtual en override om de volgende code te kunnen runnen:

```
class Program
{
    static void Main()
    {
        Dier dier = new Dier();
        dier.Eet();
        Zoogdier zoogdier = new Zoogdier();
        zoogdier.Eet();
        Kat kat = new Kat("Minou", 2);
        kat.Eet();
    }
}
```



```
Microsoft Visual Studio Debug Console
Dier is aan het eten ...
Zoogdier is aan het eten ...
De Kat eet muizen
```

REFERENTIES

**PRO C# 7 WITH .NET AND .NET CORE – ANDREW
TROELSEN – PHILIP JAPIKSE**

[HTTPS://WWW.LEARNCS.ORG/](https://www.learncs.org/)

**FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#
© SVETLIN NAKOV & CO**