

leren. durven. doen.



# *C# FUNDAMENTALS*

OOP - POLYMORFISME



# Objectgeoriënteerd programmeren (OOP) Polymorfisme



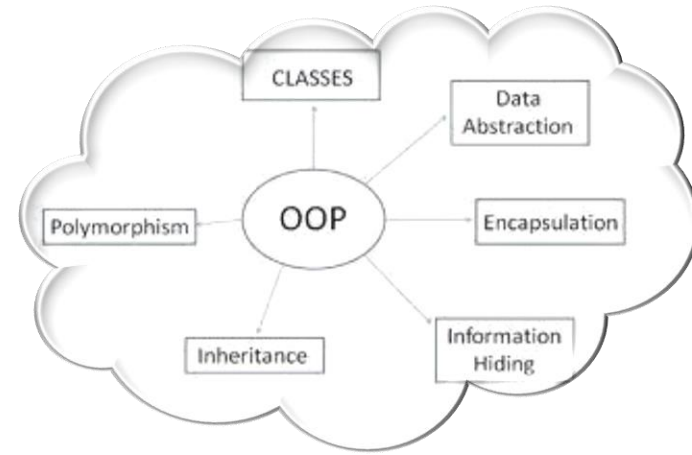
# Inhoud

Wat is Polymorfisme?

Virtuele Methods

Override Modifier

Polymorfisme – Hoe werkt het?





# Polymorfisme



# Wat is Polymorfisme?

- Polymorphism = mogelijkheid om meerdere vormen aan te nemen (bv Fifi is zowel een hond als een dier)
  - Een class kan gebruikt worden via zijn parent class
  - Een child class kan methods van de parent class uitbreiden (**override**) over verbergen (**new**)
- Polymorphism laat toe om abstracte operaties te declareren en te implementeren in de child class
  - Abstracte operatie worden hier gedefinieerd in de base interface en geïmplemented in de child classes
    - Gedeclareerd als **abstract** or **virtual**

# Polymorfisme (2)

- **Waarom een object via zijn** basis type aanspreken?
  - Om abstracte operaties te declareren
  - Om verschillende verwante types in dezelfde verzameling (collective, bv lijst) te kunnen bewaren
    - Bv Bijhouden van Hond, Kat, Kip in een **Collectie van Dier**
  - Om een meer specifiek object als argument mee te geven aan een methode die een base type als parameter declareert
    - Bv methode declaratie: `void BoerderijGeluiden(List<Dier> dieren){...}`
    - in `Main()`:

```
foreach(Dier dier in dieren)
{dier.Spreek();}
```
  - Om een meer algemeen(base) type field te declareren die later kan worden geïnitialiseerd worden met een meer gespecialiseerd (afgeleid) type
    - Bv `Dier dier;`
    - `Dier = new Hond();`

# Virtuele Methoden

- Virtuele method is een method dat op dezelfde manier wordt aangeroepen op instanties van base als derived (afgeleide) classes, maar met een verschillende implementatie
- Een method is virtual wanneer het gedeclareerd is als **virtual**

```
public virtual void BerekenOppervlakte()
```

- Methodes als virtual gedeclareerd in een basisklasse kunnen overschreven (overridden) worden in de afgeleide class d.m.v. keyword **override**

# Override

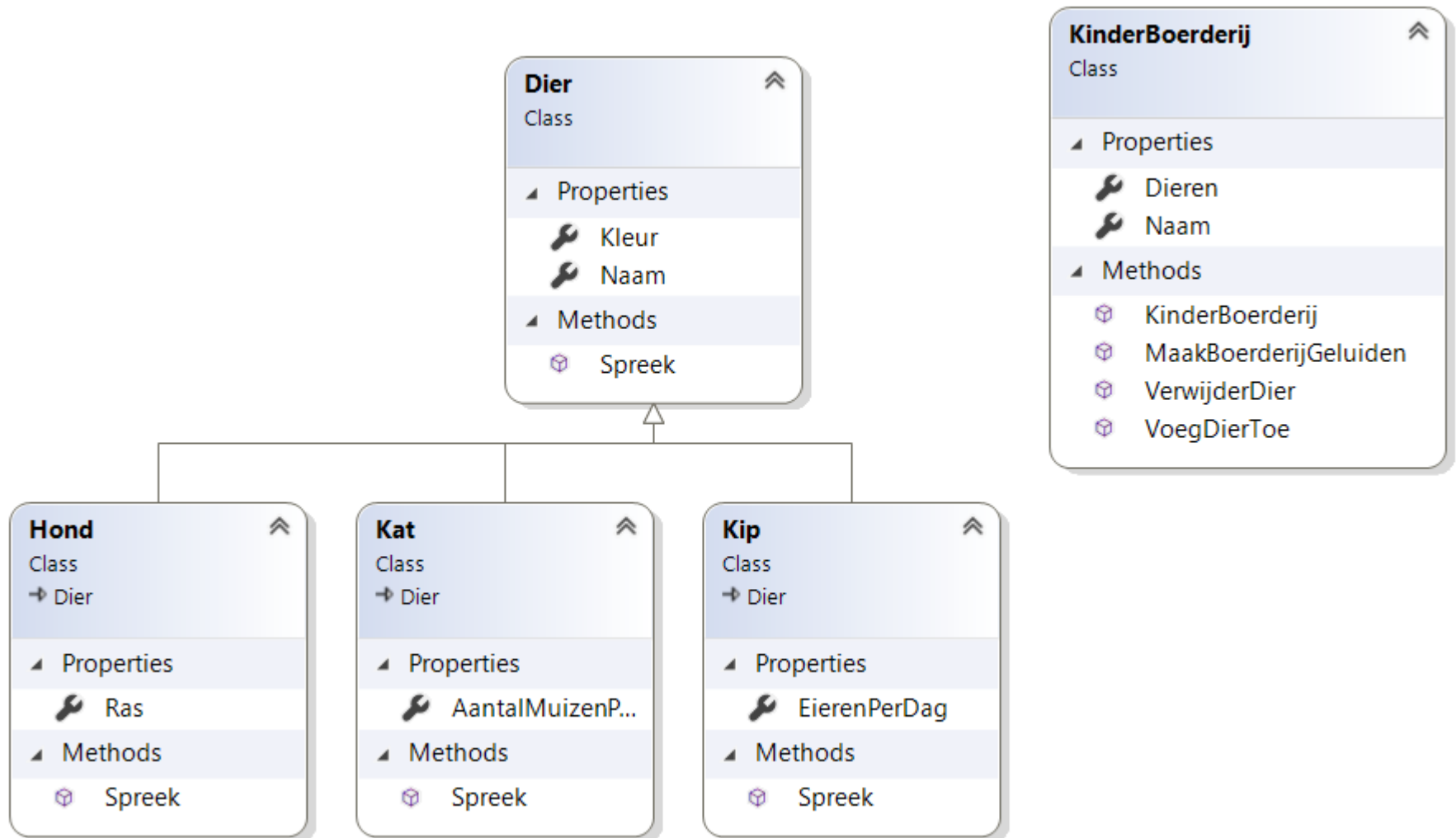
- D.m.v. **override** kunnen we een methode of property,... herdefiniëren
- Een override methode levert een nieuwe implementatie van een overgeërfde member van een base class
- Het is niet mogelijk om een override te doen van een niet-virtuele of static method
- De overriden methode van de basisklasse moet virtual, abstract, of zelf als override gedeclareerd zijn



# Polymorfisme – Hoe werkt het?

- Polymorfisme verzekert dat de geschikte methode van de afgeleide klasse wordt
- Polymorfisme functioneert door het gebruikt van een techniek “late method binding” genaamd
  - De geschikte method wordt bepaald at runtime, vlak voor de aanroep van de method
  - Wordt toegepast voor alle `abstract / virtual` methods
- Opmerking: Late binding is langzamer dan normale (early) binding

# Polymorfisme -Voorbeeld



# Polymorfisme –Voorbeeld (2)

```
public class Dier
{
    public string Naam { get; set; }
    public string Kleur { get; set; }
    public virtual void Spreek()
    {
        Console.WriteLine($"{Kleur} dier maakt geluid");
    }
}
public class Hond : Dier
{
    public string Ras { get; set; }
    public override void Spreek()
    {
        Console.WriteLine($"{Naam} zegt Woef!!");
    }
}
```

# Polymorfisme –Voorbeeld(3)

```
public class Kat : Dier
{
    public int AantalMuizenPerDag { get; set; }
    public override void Spreek()
    {
        Console.WriteLine($"{Naam} zegt Miauuuw!!");
    }
}
public class Kip : Dier
{
    public int EierenPerDag { get; set; }
    public override void Spreek()
    {
        Console.WriteLine($"{Naam} zegt Tok toook!!");
    }
}
```

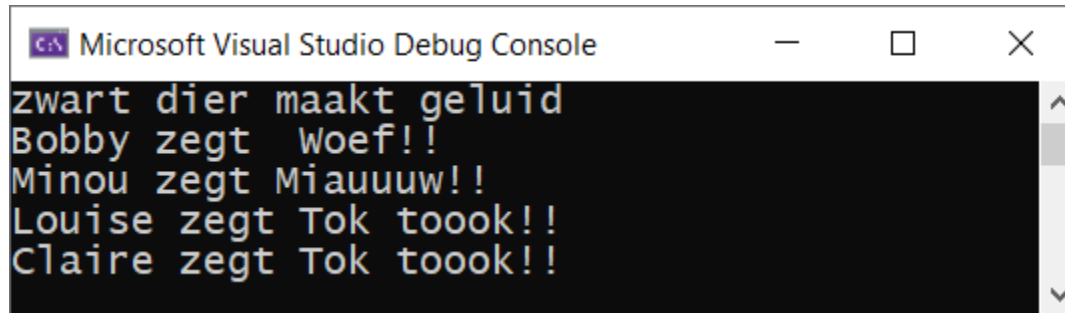
# Polymorfisme –Voorbeeld (4)

```
public class KinderBoerderij
{
    public string Naam { get; set; }
    public List<Dier> Dieren { get; private set; }

    public KinderBoerderij(string naam)
    {
        Naam = naam;
        Dieren = new List<Dier>();
    }
    public void MaakBoerderijGeluiden()
    {
        foreach (Dier dier in Dieren)
        {
            dier.Spreek();
        }
    }
    public void VoegDierToe(Dier dier)
    {
        Dieren.Add(dier);
    }
    public void VerwijderDier(Dier dier)
    {
        Dieren.Remove(dier);
    }
}
```

# Polymorfisme –Voorbeeld(5)

```
class Program
{
    static void Main()
    {
        KinderBoerderij boederij = new KinderBoerderij("Vierkantshoeve Gijzenzele");
        boederij.VoegDierToe(new Dier() { Naam = "Beestje", Kleur = "zwart" });
        boederij.VoegDierToe(new Hond() { Naam="Bobby", Ras ="Labrador"});
        boederij.VoegDierToe(new Kat() { Naam = "Minou", AantalMuizenPerDag=2});
        boederij.VoegDierToe(new Kip() { Naam = "Louise", EierenPerDag = 2 });
        boederij.VoegDierToe(new Kip() { Naam = "Claire", EierenPerDag = 1 });
        boederij.MaakBoerderijGeluiden();
    }
}
```



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

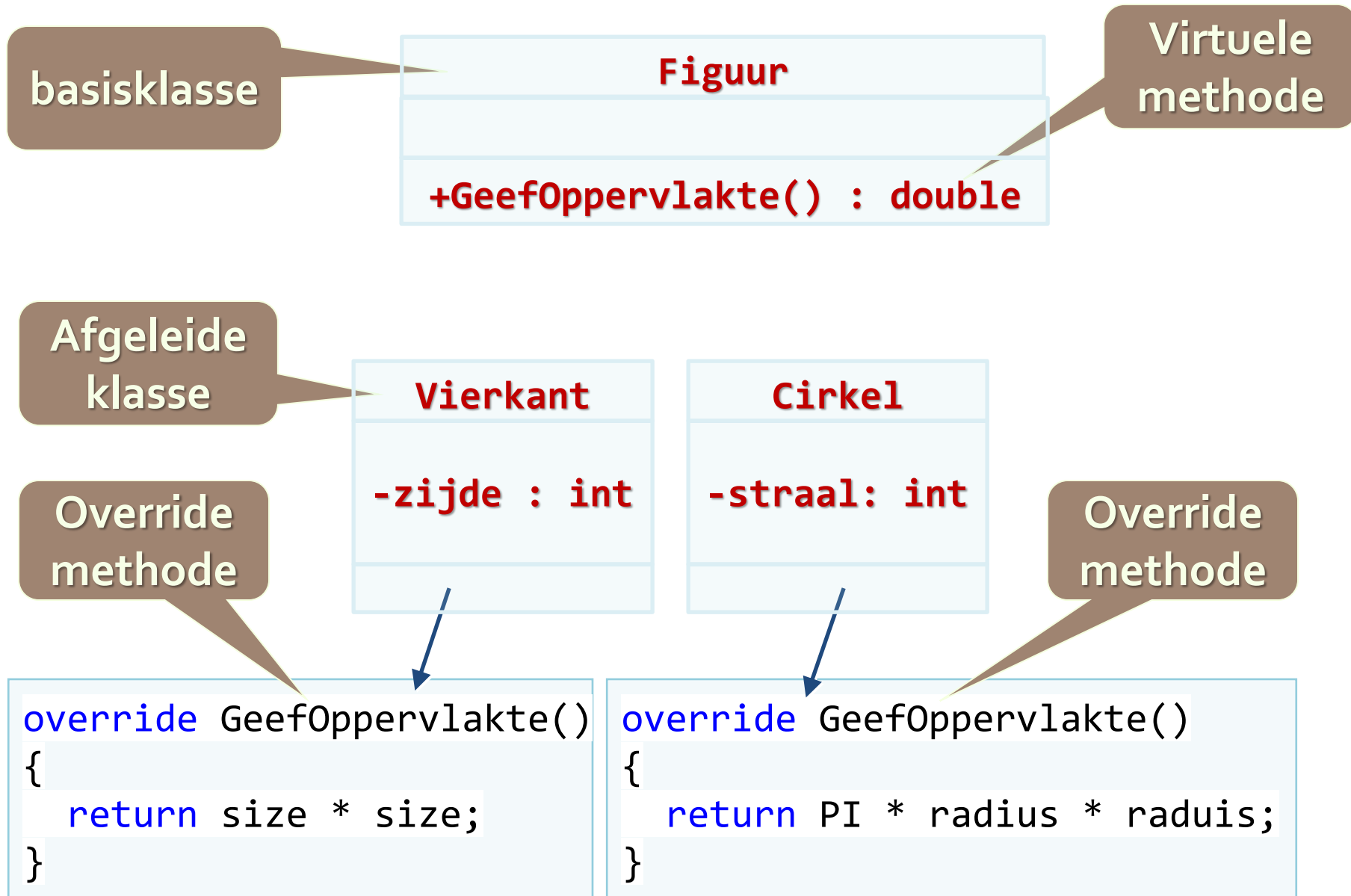
```
zwart dier maakt geluid
Bobby zegt woef!!
Minou zegt Miauuuw!!
Louise zegt Tok toook!!
Claire zegt Tok toook!!
```

# Polymorfisme

**Demo**



# Polymorfisme – Oefening



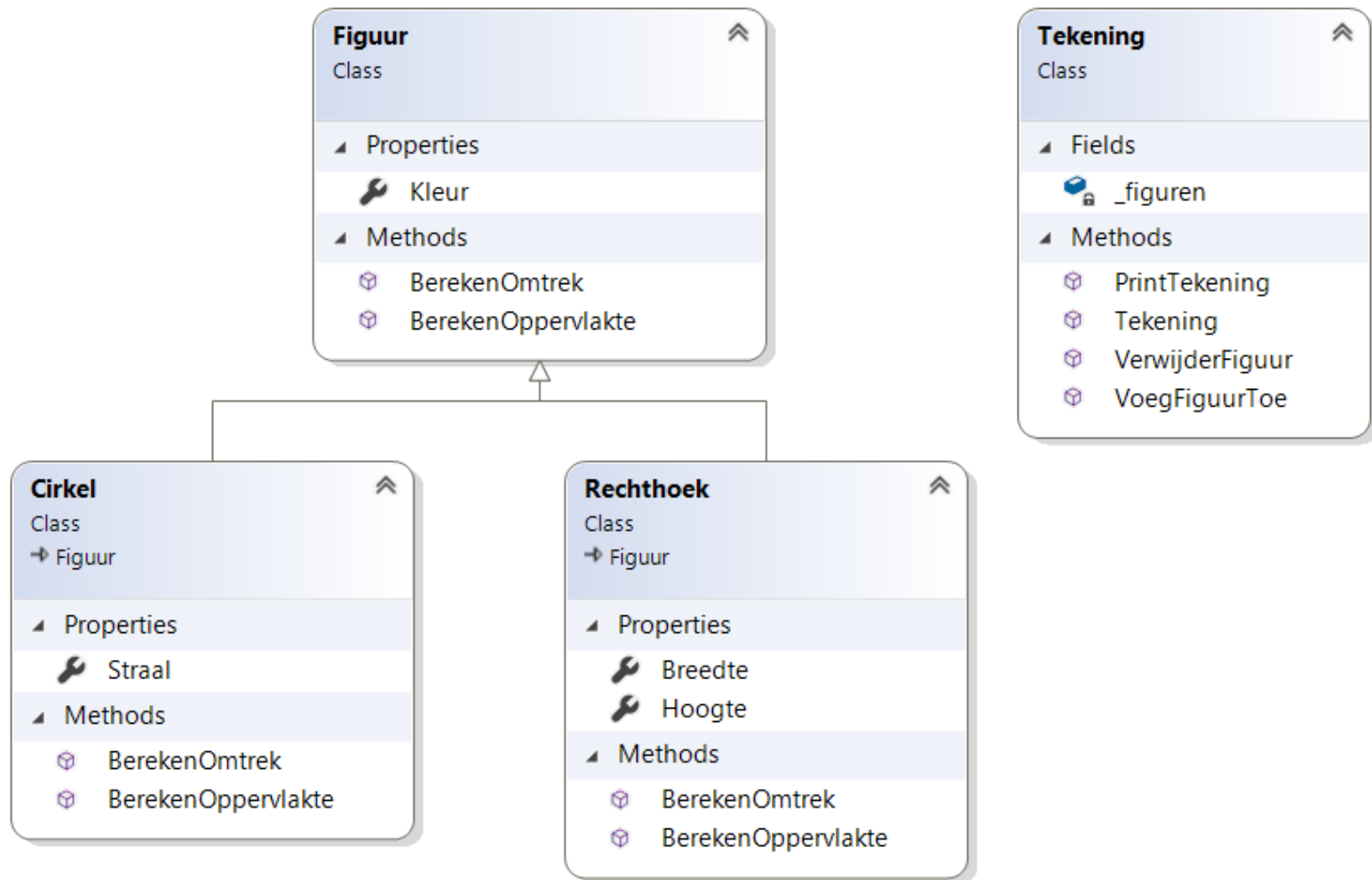


# Polymorfisme – Oefening

- Maak een klasse **Figuur** met public property **Kleur** (lees-en schrijftoegang) en 2 public methoden **BerekenOmtrek()** en **BerekenOppervlakte()** met terugkeertype **double**. Wanneer deze aangeroepen worden wordt de naam van de methode en de tekst “ van klasse Figuur is aangeroepen” en geven dan de waarde 0.0 terug
- Maak 2 afgeleide klassen **Cirkel** en **Rechthoek** van Figuur. Herdefinieer de methoden **BerekenOmtrek()** en **BerekenOppervlakte()**.
- Cirkel heeft een public Property **Straal** (read/write ) en Rechthoek Properties **Hoogte** en **Breedte** (read/write)
- Gebruik op de juiste plaats de keywords **virtual** en **override**
- Maak een klasse **Tekening** die een lijst van Figuren bijhoudt (private field `List<Figuur> __figuren` )  
Voorzie een constructor die de field `_figuren` initialiseert  
Maak 2 void methoden **VoegFiguurToe()** en **VerwijderFiguur()** die resp. een Figuur toevoegen aan/verwijderen uit de field `_figuren`

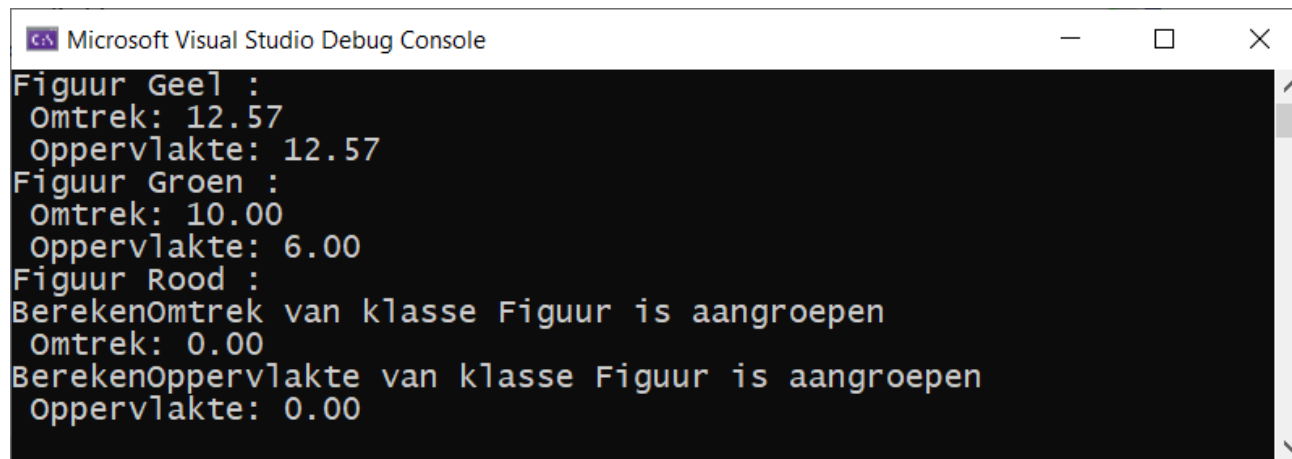
Maak een methode **PrintTekening** die in een foreach lus voor alle figuren uit `_figuren` de omtrek en oppervlakte berekent (door de methoden aan te roepen)

# Polymorfisme – Oefening (2)



# Polymorfisme – Oefening

```
class Program
{
    static void Main()
    {
        Tekening tekening = new Tekening();
        tekening.VoegFiguurToe(new Cirkel() {Kleur="Geel", Straal = 2 });
        tekening.VoegFiguurToe(new Rechthoek() { Kleur="Groen", Hoogte = 2, Breedte = 3 });
        tekening.VoegFiguurToe(new Figuur() { Kleur = "Rood"});
        tekening.PrintTekening();
    }
}
```



Microsoft Visual Studio Debug Console

```
Figuur Geel :
Omtrek: 12.57
Oppervlakte: 12.57
Figuur Groen :
Omtrek: 10.00
Oppervlakte: 6.00
Figuur Rood :
BerekenOmtrek van klasse Figuur is aangeroepen
Omtrek: 0.00
BerekenOppervlakte van klasse Figuur is aangeroepen
Oppervlakte: 0.00
```

## 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