

C# Advanced Polymorphism

Koen Bloemen



**DE HOGESCHOOL
MET HET NETWERK**

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be





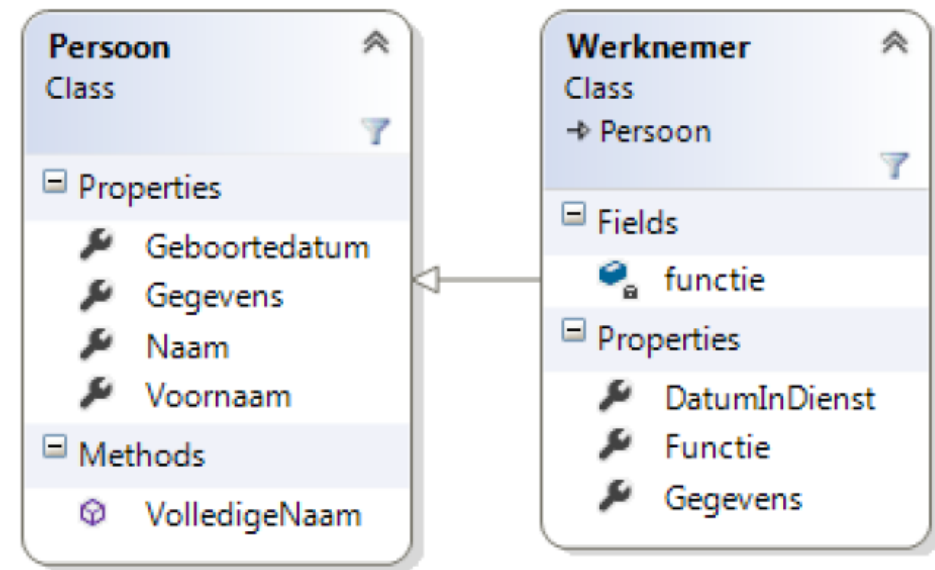
Polymorphism Generics

Polymorphism

- Dankzij virtual methods in de base class kan je ze overriden in de afgeleide class.
- Eigen implementatie van deze method in afgeleide class.
- Dankzij polymorfisme kunnen we Werknemer objecten ook in een List<Persoon> zetten.

Voorbeeld van polymorfisme:

- Voeg **virtual** method Info() toe aan base class Persoon
- **Override** method Info() in de afgeleide class Werknemer



Polymorphism

Voorbeeld polymorfisme (implementatie Persoon)

- We zorgen voor een basisimplementatie van Info() in de base class Persoon.
- Info() is virtual \Rightarrow dus in afgeleide class kunnen we eigen implementatie voorzien

```
public class Persoon
{
    public string Voornaam {get; set;}
    public string Naam {get; set;}
    public DateTime Geboortedatum {get; set;}
    public virtual string VolledigeNaam() => $"{Voornaam} {Naam}";
    public virtual string Gegevens =>
        $"{Voornaam} {Naam}\r\n{Geboortedatum.ToLongDateString()}";
    public virtual string Info() => "Ik ben een persoon.";
}
```

Polymorphism

Voorbeeld polymorfisme (implementatie Werknemer):

- Omdat method Info() in de base class Persoon virtual is, kunnen we Info() overriden in de afgeleide class.
- Dus in de afgeleide class kunnen we een eigen implementatie voorzien.

```
public class Werknemer : Persoon
{
    public DateTime DatumInDienst { get; set; }
    public string Functie { get; set; } = "Lector";
    public override string Gegevens => $"{base.Gegevens} - {Functie}";
    // Via base.Info() kan ik de Info() method
    // van base class Persoon oproepen, daarna zet ik eigen boodschap.
    public override string Info() =>
        $"{base.Info()}\r\nIk werk ook trouwens.";
}
```

Polymorphism

Dankzij polymorfisme kunnen we Persoon en Werknemer in dezelfde List<Persoon> zetten

```
Persoon persoon = new Persoon();
persoon.Naam = "Degroot";
persoon.Voornaam = "Johan";
persoon.Geboortedatum = DateTime.Parse("12/01/1992");
Werknemer werknemer = new Werknemer();
werknemer.Naam = "De Bolle";
werknemer.Voornaam = "Octaaf";
werknemer.Functie = "Kruidenier";
werknemer.Geboortedatum = DateTime.Parse("18/05/1950");
werknemer.DatumInDienst = DateTime.Parse("28/12/1980");
List<Persoon> personen = new List<Persoon>();
personen.Add(persoon);
personen.Add(werknemer);
foreach (var p in personen)
{
    Console.WriteLine(p.Gegevens); // Persoon en Werknemer hebben eigen Gegevens property
    Console.WriteLine(p.Info()); // en ook eigen Info() method met elk hun eigen implementatie
}
```

Objecten converteren

- Van base class naar afgeleide class converteren
 - Verbredende conversie
- Van afgeleide class naar base class converteren
 - Versmallende conversie
 - Hier kunnen gegevens verloren gaan, maar is soms nodig
 - Bijvoorbeeld: stel ik heb een functie die enkel de leeftijd van een Persoon kan berekenen. Dan moet ik Werknemer object omzetten naar Persoon

Objecten converteren

```
// Functie BerekenLeeftijd accepteert enkel een argument van type Persoon
public static int BerekenLeeftijd(Persoon p)
{
    DateTime vandaag = DateTime.Today;
    int leeftijd = today.Year - p.Geboortedatum.Year;
    if (p.Geboortedatum.Date > vandaag.AddYears(-leeftijd))
        leeftijd--;
    return leeftijd;
}
```

```
Werknemer werknemer = new Werknemer();
werknemer.Naam = "De Bolle";
werknemer.Voornaam = "Octaaf";
werknemer.Functie = "Kruidenier";
werknemer.Geboortedatum = DateTime.Parse("18/05/1950");
int leeftijd = BerekenLeeftijd((Persoon) werknemer); // versmallende
conversie
// Versmallende conversie gebeurt automatisch, dus kan ook zo:
int leeftijd = BerekenLeeftijd(werknemer);
```


Generics

- Generic = generiek \Rightarrow geen specifiek datatype
- Gebruiken om eender welk datatype toe te laten in:
 - Classes
 - Methods (return type en/of parameters)
 - Properties (return type)
 - Membervariabelen
- Voorbeelden van generic classes:
 - `List<T>` en `Dictionary<Key, Value>`
 - Gebruiken als: `List<int>`, `List<string>`, ...

Generics

Onze eigen generic class

```
class DataStore<T>
{
    public T Data { get; set; } // analoog voor methods
}
```

T is de **type parameter**

- Is een soort placeholder waarbij T eender welk type kan zijn.
- T kan het type zijn voor
 - Return waarden, parameters, membervariabelen,...
 - **Gebruik gewoon T zoals je anders int, string, float,... gebruikt!**
- Hier
 - Data is een generic property die eender welk type T returnt.

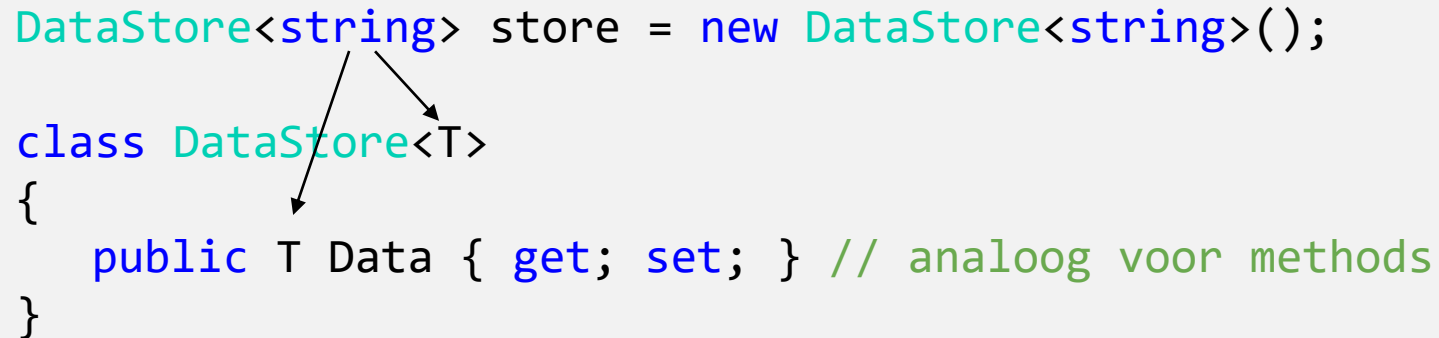
Generics

Onze eigen generic class gebruiken

```
DataStore<string> store = new DataStore<string>();  
store.Data = "Hello World!";  
//store.Data = 123; // geeft een compile-time error...
```

Tijdens **compileren van programma** wordt T ingevuld als string

```
DataStore<string> store = new DataStore<string>();  
  
class DataStore<T>  
{  
    public T Data { get; set; } // analoog voor methods  
}
```

A diagram illustrating the compilation process. Two arrows originate from the 'string' type in the first line of code. One arrow points to the 'T' in the class definition 'class DataStore<T>', and the other points to the 'T' in the property definition 'public T Data'. This visualizes how the generic type 'T' is replaced by 'string' during compilation.

Generics

Onze eigen generic class met meerdere type parameters

```
class KeyValuePair<TKey, TValue>
{
    public TKey Key { get; set; }
    public TValue Value { get; set; }
}
class Triple<T1, T2, T3>
{
    public T1 Waarde1 { get; set; }
    public T2 Waarde2 { get; set; }
    public T3 Waarde3 { get; set; }
}
```


Generics

Generic classes vergroten het hergebruik van code. **Niet overdrijven!!!**

```
DataStore<string> strStore = new DataStore<string>();  
strStore.Data = "Hello World!";  
//strStore.Data = 123; // geeft compile-time error...  
  
DataStore<int> intStore = new DataStore<int>();  
intStore.Data = 100;  
//intStore.Data = "Hello World!"; // geeft compile-time error...  
  
KeyValuePair<int, string> kvp1 = new KeyValuePair<int, string>();  
kvp1.Key = 100;  
kvp1.Value = "Hundred";  
  
KeyValuePair<string, string> kvp2 = new KeyValuePair<string, string>();  
kvp2.Key = "CS";  
kvp2.Value = "Computer Science";
```