Interfaces

Zie Scherp Scherper - Hoofdstuk 17

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- uitleggen wat een **interface** is en hoe ze verschilt van een klasse
- een interface **definiëren** en in een klasse **implementeren**
- kiezen tussen een **interface** en een **abstracte klasse**
- **meerdere** interfaces combineren (waar overerving tekortschiet)
- met `is` en `as` objecten op een interface bevragen
- interfaces inzetten voor losser gekoppelde, **SOLID**-code

Interfaces in de echte wereld

“Interface” = tussen vlakken: de afgesproken **verbinding** tussen twee systemen.

- Een auto: stuur en pedalen, ongeacht benzine of elektrisch
- Een pc: USB, HDMI, audio, … wereldwijd afgesproken aansluitingen

> Ken je de interface, dan kan je ze gebruiken zonder de interne werking te kennen.

Belangrijk

Dit gaat **niet** over grafische User Interfaces. U weze gewaarschuwd.

Interfaces in OOP

> Een **interface** is een belofte: een lijst van publieke methoden en properties die een klasse zegt te hebben, **zonder** de implementatie.

Het is als een sticker “USB 3.0 compatibel”: je weet wat het apparaat kan, niet hoe het binnenin werkt.

Een interface in C

```
1  interface ISuperHeld
2  {
3      void SchietLasers();
4      int VerlaagKracht(bool isZwak);
5      int Power { get; set; }
6  }
```

- `interface` in plaats van `class`, naam start met een **`I`**
- **Geen** `public` (alles is publiek), **geen** code: elke signatuur eindigt op `;`

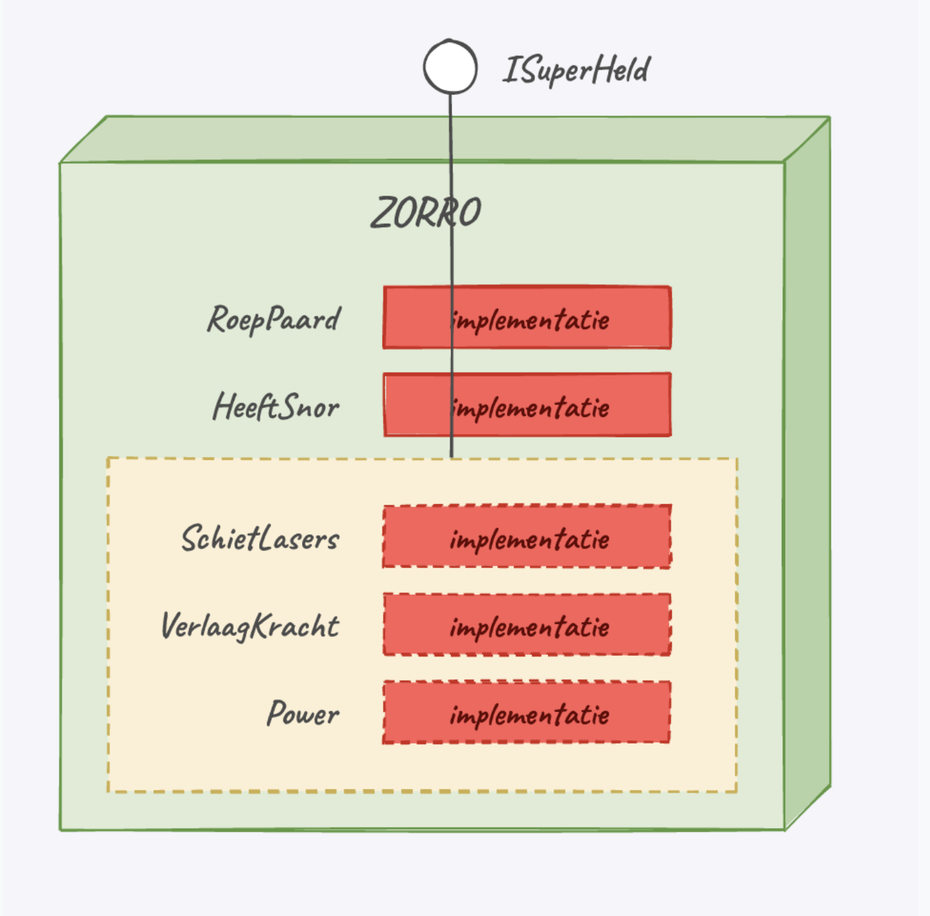
De regels

- Geen **instantievariabelen** en geen **constructors**
- Geen **access modifiers**: alles is `public`
- Een interface erft niet van een klasse, wel van andere **interfaces**
- In de regel **geen code**, enkel signaturen

> **Tip**
>
> Sinds C# 8 bestaan *default interface methods* (met code), maar we houden het bewust bij zuivere signaturen.

Een interface visualiseren

Een interface is als een blad met **gaten**: leg je het op een klasse die de belofte nakomt, dan vallen de gaten precies over de juiste methoden en properties.



De interface hangt aan de klasse.

Een klasse implementeert een interface

```
internal class Zorro : ISuperHeld
{
    public void RoepPaard() { }
    public bool HeeftSnor { get; set; }

    public void SchietLasers() { Console.WriteLine("pewpew"); }
    public int VerlaagKracht(bool isZwak)
    {
        if (isZwak) return 5;
        return 10;
    }
    public int Power { get; set; }
}
```

Zolang `Zorro` niet **alles** uit `ISuperHeld` implementeert, compileert de klasse niet.

> **Tip**
>
> In VS: klik op het lampje bij de klasse-signatuur en kies “Implement interface”.

Interface of abstracte klasse?

	Abstracte klasse	**Interface**
Gedeelde **code**	ja	nee
Instantievariabelen	ja	nee
Constructors	ja	nee
Hoeveel **combineren**?	1	meerdere
Relatie	“**is een**”	“**kan iets**”

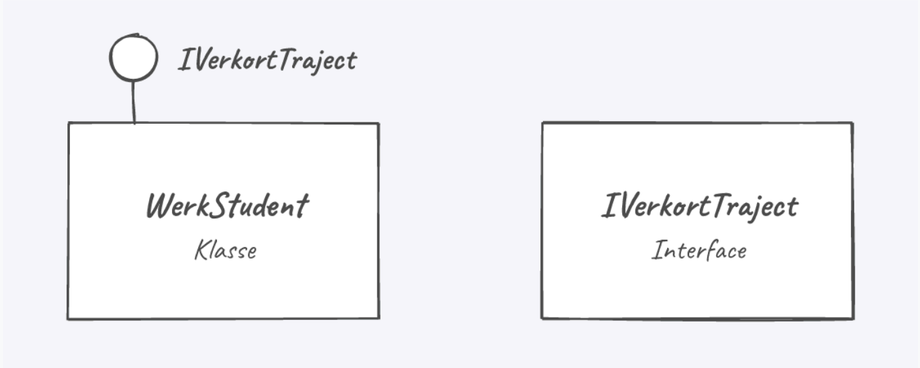
Tip

Gedeelde code en een echte is-een-relatie? Abstracte klasse. Enkel afdwingen dat ongerelateerde klassen “iets kunnen” (`IVliegt` voor `Vogel` én `Vliegtuig`)? Interface.

UML en meerdere interfaces

Een klasse erft van **maar één** parent, maar mag **meerdere** interfaces dragen:

```
1  internal class Zorro : Man, ISuperHeld { }
2  internal class Batman : Man, ISuperHeld, ICo
```



Interface in UML (de “lolly”).

Eerst de parent-klasse, dan de interface(s).

> **Opmerking**
>
> Interfaces mogen ook van elkaar erven: `interface IGod : ISuperHeld`.

is met interfaces

```
if (gameOfThrones is IVerwijderbaar)
{
    Console.WriteLine("Ik kan dit verwijderen");
}
```

Net als bij polymorfisme schitteren interfaces in een **lijst** vol verschillende objecten:

```
foreach (var persoon in werknemers)
{
    if (persoon is IManager manager)
    {
        // enkel de managers
    }
}
```

Interfaces in de praktijk

Een `Minister` zijn als “bij-job”, niet als hoofd-klasse. Definieer een interface:

```
1  interface IMinister
2  {
3      void Adviseer();
4  }
```

Nu kan **eender wie** minister worden zonder z’n bestaande klasse op te geven:

```
1  internal class Ceo : IMinister
2  {
3      public void Adviseer() { Console.WriteLine("Vrijhandel is essentieel!"); }
4      public void MaakJaarlijkseOmzet() { Console.WriteLine("Geld!!!"); }
5  }
```

De premier werkt met IMinister

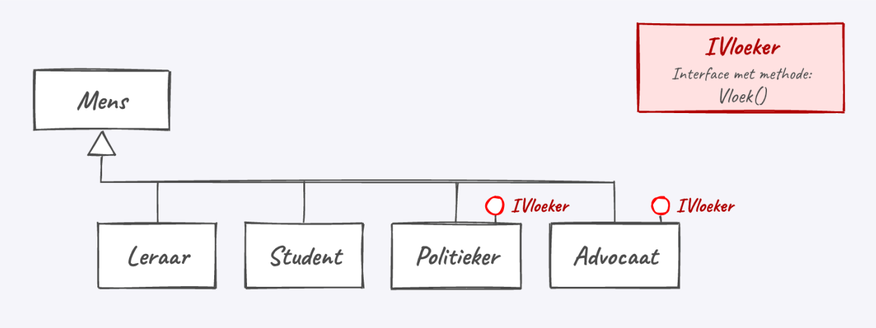
```
List<IMinister> alleMinisters = new List<IMinister>();
alleMinisters.Add(new Ceo());
alleMinisters.Add(new MinisterVanMilieu());

foreach (IMinister minister in alleMinisters)
{
    minister.Adviseer();
}
```

Een CEO én een echte minister staan samen in één lijst. OOP benadert de realiteit.

Alles samen: vloekende mensen

Mensen spreken; enkel **vloekers** vloeken:

```
1  interface IVloeker { void Vloek(); }
2
3  internal class Leraar : Mens { }
4  internal class Politieker : Mens, IVloeker
5  {
6      public void Vloek() { Console.WriteLine("G
7  }
```



Klasse-schema vloekende mensen.

Hoe laat je in een `Mens[]` enkel de vloekers vloeken?

Drie oplossingen

```
1  // 1) is + harde cast
2  if (mens is IVloeker)
3      ((IVloeker)mens).Vloek();
4
5  // 2) as + null-check
6  IVloeker v = mens as IVloeker;
7  if (v != null) v.Vloek();
8
9  // 3) pattern matching (de moderne manier)
10 if (mens is IVloeker vloeker)
11     vloeker.Vloek();
12 else
13     mens.Spreek();
```

SOLID en de black-box

Werk **tegen de interface**, niet tegen de implementatie. Het boeit je niet meer wat er in een klasse zit; aan haar interfaces zie je wat ze kan.

Dat is de kern van de **SOLID**-principes: abstractie en *separation of concerns*. De ultieme black-box-revelatie. *See what I did there?*

Conclusie

- Een **interface** is een belofte van publieke methoden en properties, **zonder** code
- Een klasse die een interface draagt, **moet** alles ervan implementeren
- Kies een **interface** voor “kan iets” en om er **meerdere** te combineren; een **abstracte klasse** voor gedeelde code en “is een”
- Met `is` en `as` (en pattern matching) bevraag je objecten op een interface
- Interfaces zorgen voor losser gekoppelde, **SOLID**-code

> **Tip**
>
> Volgende halte: **bestandsverwerking**: lezen van en schrijven naar bestanden.

Meer weten

Kennisclips

- Interfaces introductie
- Interfaces & polymorfisme
- Interfaces in de praktijk: meme-detective
- Interfaces en polymorfisme: vloekende mensen
- Interfaces en polymorfisme: fuifsimulator

Oefeningen H17 · Quizlet flashcards

Zie verder: andere talen

Dezelfde concepten uit dit hoofdstuk, maar dan in andere talen. Handig om later code in Python of JavaScript te leren lezen.

Zie verder: interfaces elders

Java en **TypeScript** hebben net als C# een echt `interface`-keyword:

```
interface ISuperHeld {
    schietLasers(): void;
    power: number;
}
```

Python doet aan *duck typing*: geen verplichte interface, zolang een object toevallig de juiste methoden heeft, mag je het gebruiken. C# laat de compiler je belofte hard controleren.

Zie verder: waarom maar één parent?

Dat je meerdere interfaces maar één klasse mag erven, koos **Java** ook bewust. **C++** laat wél meervoudige overerving van klassen toe:

```
1  class Zorro : public Man, public SuperHeld { };  // mag in C++
```

Krachtig, maar het leidt tot het *diamond problem*: erven twee parents dezelfde methode, dan weet de compiler niet meer welke. Door enkel meervoudige *interfaces* (zonder eigen code) toe te laten, ontwijken C# en Java dat volledig.

Samenvattende poster

Een handige samenvattende poster (Opgelet: gemaakt m.b.v. ChatGPT Image Gen 2).