



// 01

Een belofte, geen code

Een interface beschrijft welke **publieke methoden en properties** een klasse belooft te hebben, niet hoe die werken.

```
interface ISuperHeld
{
    void SchietLasers();
    int VerlaagKracht(bool isZwak);
    int Power { get; set; }
}
```

- **interface** in plaats van **class**, de naam begint met een **I**
- Geen **public** voor de leden: alles is al publiek
- Geen body: elke methode eindigt meteen op **;**
- Geen **instantievariabelen** en geen **constructors**
- Erft niet van een klasse, wel van andere interfaces:
`interface IGod : ISuperHeld`

// 02

Een interface implementeren

Zet de interface achter een dubbele punt en schrijf **elk** lid uit, met **public**.

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

// LET OP

Ontbreekt er één lid, dan compileert de klasse niet. Een interface geeft je geen code cadeau.

// 03

Interface of abstracte klasse?

	ABSTRACTE KLASSE	INTERFACE
gedeelde code	ja	nee, enkel signaturen
instantievariabelen	ja	nee
constructors	ja	nee
hoeveel combineren?	maar 1	meerdere
relatie	"is een"	"kan iets"

Vogel en Vliegtuig delen geen parent: een gedeelde **IVliegt** kan enkel als interface.

// 04

Minister als bij-job

Met `Ceo : IMinister` is een `Ceo` naast zijn eigen job ook minister. Een `List<IMinister>` aanvaardt elk object met die interface:

```
List<IMinister> ministers = new List<IMinister>();
ministers.Add(new Ceo());
foreach (IMinister minister in ministers)
{
    minister.Adviseer();
}
```

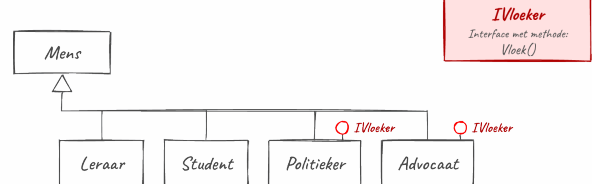
// 05

Eén parent, meerdere interfaces

Een klasse erft van hoogstens **één** klasse, maar kan **meerdere** interfaces dragen. Eerst de klasse, dan de interfaces:

```
internal class Batman : Man, ISuperHeld, ICoureur
```

In UML hangt een interface als een **lolly** aan de klasse:



// 06

is en as met interfaces

Met **is** vraag je of een object een interface heeft. Met pattern matching heb je het meteen als **IVloeker**:

```
Mens[] mensjes = {new Leraar(), new Politieker()};
foreach (Mens mens in mensjes)
{
    if (mens is IVloeker vloeker)
    {
        vloeker.Vloek();
    }
}
```

UITVOER

Godvermiljaardedju, zei de politieker

Of met **as**, en dan testen op **null**:

`IVloeker tijdelijk = mens as IVloeker;`

// 07

Sorteren met IComparable

`Array.Sort` op eigen objecten crasht (`InvalidOperationException`): .NET weet niet hoe ze te sorteren. Implementeer daarom **IComparable**:

COMPARETO GEEFT	BETEKENT
negatief getal	dit object komt vóór <code>obj</code>
0	zelfde plaats
positief getal	dit object komt na <code>obj</code>

```
// in internal class Land : IComparable
public int CompareTo(object obj)
{
    Land temp = obj as Land;
    if (temp == null)
    {
        throw new ArgumentException("Geen Land");
    }
    if (Oppervlakte > temp.Oppervlakte) return 1;
    if (Oppervlakte < temp.Oppervlakte) return -1;
    return 0;
}
```

Nu sorteren `Array.Sort(landen);` en `landLijst.Sort();`; de array of lijst zelf. Gelijke oppervlakte? Vergelijk verder op naam met `Naam.CompareTo(temp.Naam)`.

// LET OP

IComparable bestaat al in .NET: schrijf ze niet zelf opnieuw. De methode heet exact `CompareTo`; met een `Vergelijk` compileert `Land` niet.