Gevorderde overervingsconcepten

Zie Scherp Scherper - Hoofdstuk 14

Tim Dams

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

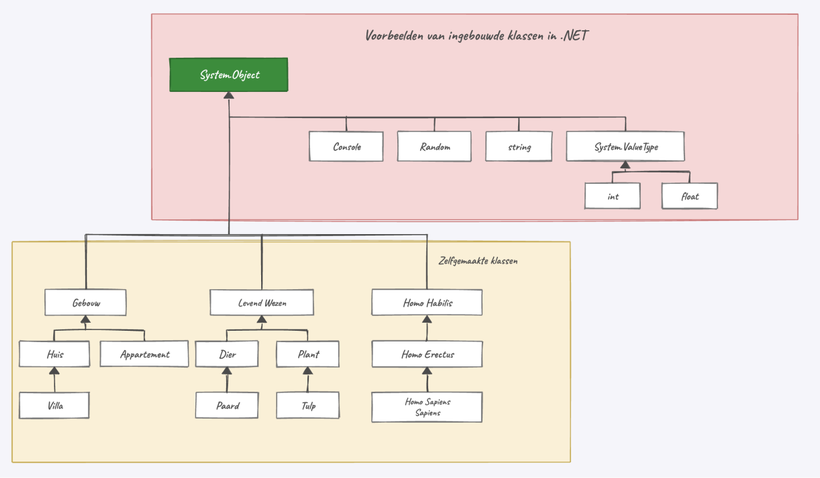
- uitleggen dat alles erft van `System.Object`
- de ingebouwde methoden `ToString`, `Equals` en `GetHashCode` overriden
- een **abstracte klasse** en **abstracte methoden** maken
- het verschil tussen `abstract` en `virtual`/`sealed` uitleggen
- een **eigen exception** ontwerpen en opwerpen

De oer-klasse: System.Object

Alles in C# erft van **`System.Object`**: je eigen klassen, `Random`, `Console`, zelfs `int` en `bool`.

```
1  internal class Student : System.Object
2  { }
```

Schrijf je geen parent, dan is `System.Object` automatisch de parent.



Alles stamt af van System.Object.

Vier ingebouwde methoden

Elke klasse heeft er meteen vier, van `System.Object`:

Methode	Doel
`ToString()`	een `string` die het object voorstelt
`Equals()`	zijn twee objecten gelijk?
`GetHashCode()`	unieke hash van het object
`GetType()`	het datatype van het object

> **Opmerking**
>
> Drie ervan zijn **`virtual`** (`ToString`, `Equals`, `GetHashCode`): die kan je overriden. `GetType()` niet: dat moet altijd het echte type teruggeven.

GetType()

```
1  Student stud1 = new Student();
2  Console.WriteLine(stud1.GetType());      // StudentManager.Student
3  Console.WriteLine(stud1.GetType().Name); // Student
```

`GetType()` geeft een `Type`-object terug, met o.a. een `Name`-property.

ToString(): het werkpaardje

`Console.WriteLine(stud1)` roept stiekem `stud1.ToString()` aan. Standaard geeft dat de klassenaam. Override het voor iets nuttigs:

```
internal class Student
{
    public string Voornaam { get; set; }
    public int Geboortejaar { get; set; }

    public override string ToString()
    {
        return $"{Voornaam} ({Geboortejaar})";
    }
}
```

Nu toont `Console.WriteLine(stud1)` netjes `Tim Dams (1981)`.

Equals() overriden

Jij bepaalt wanneer twee objecten “gelijk” zijn:

```
1  public override bool Equals(object o)
2  {
3      if (o is not Student temp) // null of geen Student? Niet gelijk.
4          return false;
5      return Geboortejaar == temp.Geboortejaar && Voornaam == temp.Voornaam;
6  }
```

> **Opmerking**
>
> `o is not Student temp` checkt het type én steekt `o` meteen als `Student` in `temp`. Dat `is`-patroon zie je voluit in hoofdstuk 18.

GetHashCode() overriden

Override je `Equals`, override dan **ook** `GetHashCode` (gelijke objecten moeten dezelfde hash geven, belangrijk voor `Dictionary` en `HashSet`):

```
1  public override int GetHashCode()
2  {
3      return HashCode.Combine(Voornaam, Geboortejaar);
4  }
```

> **Tip**
>
> Geef dezelfde properties mee die je in `Equals` vergelijkt. `HashCode.Combine` doet het rekenwerk voor je.

Even moed verzamelen

“Ok, wow? Vier methoden en wat beloofde compatibiliteit? Call me unimpressed.”

Begrijpelijk. We leggen puzzelstukjes klaar die straks samenkomen. **Polymorfisme** wordt onze doelpuntenmaker, en `System.Object` geeft telkens de perfecte voorzet.

Abstracte klassen

Geef mij de blauwdruk van een “meubel”. Een tafel? Een kast? Een bed? Sommige concepten zijn te **abstract** om er een object van te maken.

```
1  internal abstract class Dier
2  {
3      public string Naam { get; set; }
4  }
```

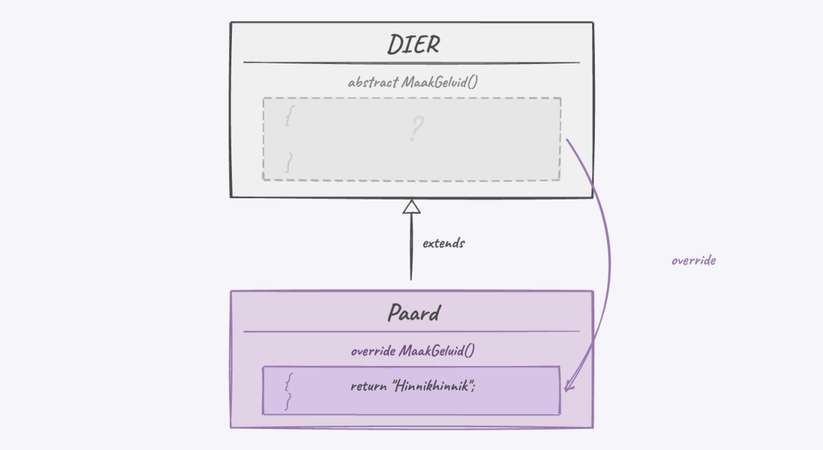
- Je kan **niet** `new Dier()` doen
- Je kan er **wel** van overerven: `Paard : Dier`, `Wolf : Dier`

> **Tip**
>
> `abstract` en `sealed` zijn tegenpolen: bij `abstract` *moet* je overerven, bij `sealed` *mag* het net niet.

Abstracte methoden

Welk geluid maakt “een dier”? Onbekend.
Geef enkel de signatuur:

```
1  internal abstract class Dier
2  {
3      public abstract string MaakGeluid();
4  }
```



De child vult het gat in.

Child-klassen **moeten** dit overriden:

```
1  public override string MaakGeluid()
2  {
3      return "Hinnikhinnik";
4  }
```

Belangrijk

`virtual` = override **mag**, `abstract` = override **moet**. Een abstracte methode kan enkel in een abstracte klasse.

Abstracte properties

Ook properties kunnen `abstract`; de child vult de implementatie in:

```
internal abstract class Dier
{
    public abstract int MaxLeeftijd { get; }
}
internal class Olifant : Dier
{
    public override int MaxLeeftijd
    {
        get { return 100; }
    }
}
```

Een abstracte `{ get; }` is dus **geen** auto-property: er staat bewust geen implementatie.

Pong met abstract

Een gemeenschappelijke `SpelObject` voor alles op het scherm, zonder te weten **hoe** het getekend wordt:

```
internal abstract class SpelObject
{
    public int X { get; set; }
    public int Y { get; set; }
    public abstract void TekenOpScherm();
}
```

`Balletje` en een nieuw `ScoreBoard` erven hiervan en implementeren elk hun eigen `TekenOpScherm()`.

Zelf een exception opwerpen

Exceptions zijn ook maar klassen die erven van `Exception`. Werp er zelf een op met **`throw`**:

```
1  static int Bereken(int getal)
2  {
3      if (getal != 0)
4          return 100 / getal;
5      else
6          throw new DivideByZeroException("BOEM. ZWART GAT!");
7  }
```

Elders vang je die op met `catch (DivideByZeroException e)`.

Een eigen exception ontwerpen

Erf van `Exception` en voorzie de drie standaard-constructors:

```
internal class Timception : Exception
{
    public Timception() { }
    public Timception(string message) : base(message) { }
    public Timception(string message, Exception inner)
        : base(message, inner) { }
}
```

```
throw new Timception("Een wilde exception verschijnt!");
```

throw versus throw ex

> ⚠ **Waarschuwing**
>
> Gooi je een opgevangen exception opnieuw door, gebruik dan **`throw;`** (zonder variabele):
>
> ```
> 1 catch (Exception e)
> 2 {
> 3     // ...loggen...
> 4     throw;      // GOED: behoudt de stack trace
> 5     // throw e; // SLECHT: wist de stack trace
> 6 }
> ```
>
> `throw e;` wist de **stack trace**, waardoor de echte oorzaak veel moeilijker te vinden is.

Conclusie

- Alles erft van **`System.Object`**, met o.a. `ToString`, `Equals`, `GetHashCode` (allemaal `virtual`)
- Override **`ToString`** voor leesbare objecten; override **`Equals`** + **`GetHashCode`** samen
- Een **`abstract`** klasse kan je niet instantiëren, maar dient als basis voor children
- Een **abstracte methode** zonder body **moet** de child overriden
- Eigen **exceptions** erven van `Exception`; gebruik `throw;` om de stack trace te bewaren

> **Tip**
>
> Volgende halte: **compositie en aggregatie**: de “heeft-een”-relatie.

Meer weten

Kennisclips

- Abstracte klassen
- System.Object en ToString
- Zelf uitzonderingen maken
- Abstract en System.Object met Zoo-dieren
- Class diagram en de class designer in Visual Studio

Oefeningen H14 · 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: één oer-klasse

Ook **Java** kent dezelfde oer-klasse, daar gewoon `Object` genoemd. Elke klasse erft er impliciet van, met methoden als `toString()`, `equals()` en `hashCode()`:

```
1 class Student {
2     // erft automatisch van java.lang.Object
3 }
4
5 Student stud = new Student();
6 System.out.println(stud.getClass().getName()); // Student
```

Heel gelijkaardig dus. Ook **Python** doet dit met `object`.

Zie verder: abstract zonder keyword

C++ kent het keyword `abstract` niet, maar wel exact hetzelfde idee: een “pure virtual” methode. Je zet `= 0` achter de signatuur, en zo’n klasse kan je niet instantiëren:

```cpp
class Dier {
public:
    virtual std::string MaakGeluid() = 0; // pure virtual
};

class Paard : public Dier {
public:
    std::string MaakGeluid() override { return "Hinnikhinnik"; }
};
```

Python doet dit met de `ABC`-module en `@abstractmethod`.

Zie verder: eigen exceptions in Python

In **Python** werkt dit verrassend gelijkaardig: erf over van `Exception` en gooi op met `raise` (het equivalent van `throw`):

```
1  class Timception(Exception):
2      pass
3
4  def tims_methode():
5      raise Timception("Een wilde exception verschijnt!")
6
7  try:
8      tims_methode()
9  except Timception as e:
10     print(e)
```

Samenvattende poster

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