Ernst Peter Tamminga
Managing consultant werkzaam bij XCESS expertise center b.v.
http://www.xcess.nl.

# Visual Basic .NET en objectoriëntatie: een nieuw tijdperk breekt aan

GROTE VOORDELEN VOOR DE PROFESSIONELE DEVELOPER

**Dit artikel behandelt objectoriëntatie (OO) in VB .NET: de basisbegrippen, de implementatie, geavanceerde uitbreidingen, gelardeerd met voorbeeldcode waarin het concept, de toepassing en voordelen van OO in VB .NET ruim aan de orde komen. Na het lezen van dit artikel zul je er van overtuigd zijn dat je niet meer zonder OO kunt in de .NET-wereld – en dat kan ook niet, aangezien de gehele .NET-omgeving gebaseerd is op objecten. Kortom: wen aan OO en doe er je voordeel mee!**

Objectoriëntatie (OO): dat hebben we nodig en snel! Alleen daarmee kunnen we vooruit in de wereld, zijn we productiever, hebben we beter onderhoudbare programma's, kunnen we beter structureren en iedereen heeft het, dus waarom wij niet? Dat zijn uitspraken die je in het kamp van de Visual Basic developer sinds VB 3.0 vele malen hebt kunnen horen. Sinds jaar en dag staat de wens voor een volledige ondersteuning van OO hoog op de lijst van de developers met gewenste wijzigingen voor een nieuwe versie van VB. Nu is het dan eindelijk zover. De nieuwste versie van VB – VB .NET – kent volledige ondersteuning van OO. En nu? Wat doen we ermee? Hoe kunnen we ons voordeel doen met OO?

## Basisbegrippen OO

Voordat we uitgebreid in gaan op de mogelijkheden van OO binnen VB .NET, geef ik eerst een samenvatting van het basisbegrip van 'Object Oriented' programmeren. In dit artikel gebruik ik de Engelse termen, omdat enerzijds de Engelse termen vaak in deze begrippen worden genoemd, en omdat anderzijds VB .NET-code ook op basis van Engelse termen werkt. Kortom: een bewuste keuze om niet te vertalen. Binnen OO kennen we de volgende algemene basisbegrippen:

VB.NET wordt ondersteund door OO. En wat nu?

*Object*: een abstractie (in code) van iets dat in de werkelijkheid voorkomt. Bijvoorbeeld een Persoon-object, dat kenmerken heeft van een daadwerkelijke persoon, met naam (Jan Jansen), geboortedatum (23 december 1962), enzovoort.

*Class*: een soort sjabloon van een object. Een class is de code die het object definieert, een abstractie van een concept dat in werkelijkheid bestaat; de basis voor het maken van 'instances' van specifieke objecten, bijvoorbeeld de beschrijving van kenmerken van een Persoon in algemene zin.

*Instance*: het voorkomen van een object dat gebaseerd is op de regels van een class.

*Abstraction*: de mogelijkheid code te maken die als 'black box' fungeert. Niet op ieder niveau zul je je druk maken om de details. Je lost het (deel)probleem op dat niveau op, om vervolgens op een hoger niveau alleen de oplossing te gebruiken zonder je om de details druk te maken.

*Encapsulation*: een scheiding maken tussen de formele specificatie (interface) en de wijze waarop deze wordt geïmplementeerd. Zo lang de interface niet wijzigt, heb je de vrijheid om de implementatie te veranderen.

*Polymorphism*: de mogelijkheid één routine te maken die op een zelfde manier

interactie pleegt met verschillende objecten, bijvoorbeeld de mogelijkheid een Printopdracht te geven aan een formulier, een rapport, een recordset, een scherm.
*Inheritance*: de mogelijkheid een nieuwe class te maken die alle eigenschappen en gedrag van een bestaande class hergebruikt zonder deze opnieuw te hoeven definiëren.

## (Verwachte) voordelen van OO

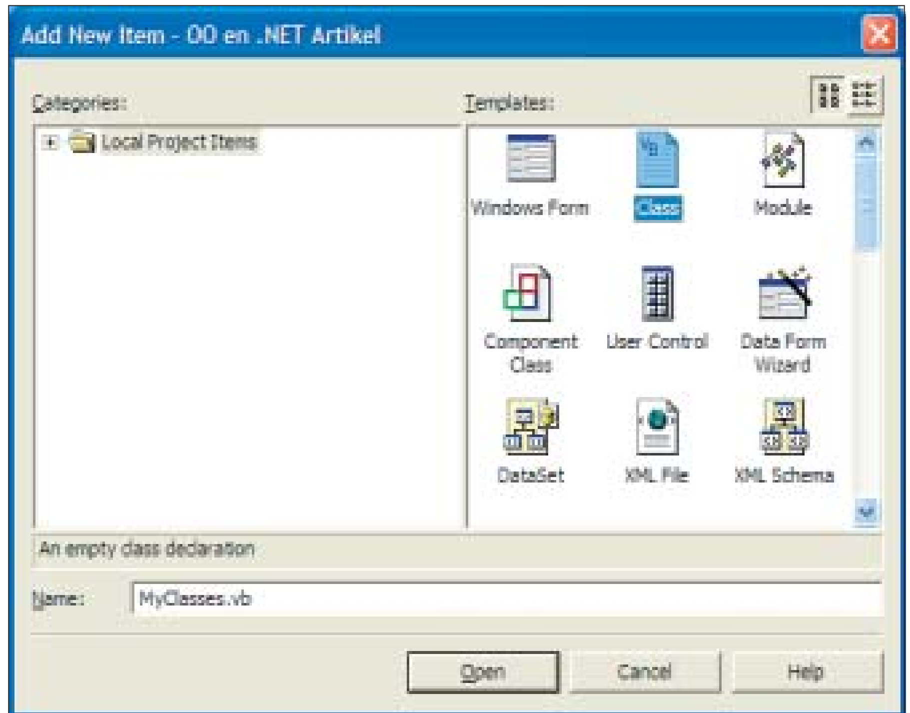
Een groot deel van de OO-begrippen was al beschikbaar in VB sinds versie 3 en 4. Absoluut nieuw is de beschikbaarheid van inheritance in VB.NET. Het enige dat 'echt' ontbreekt om VB .NET alle eigenschappen van OO toe te kennen is het ontbreken van de mogelijkheid van multiple inheritance, een functionaliteit waarvan je je kunt afvragen of dit überhaupt bijdraagt aan de kwaliteit van het ontwikkelen. En waarom is het beschikbaar hebben van OO dan nuttig? Niet zozeer omdat er al lang om gevraagd is, maar omdat men verwacht dat met een juist gebruik van de mogelijkheden die een volledige OO-taal met zich meebrengt er minder, efficiënter en beter onderhoudbare code kan worden geschreven. VB (.NET) developers worden productiever, werken sneller en leveren een betrouwbaarder resultaat dan 'gewone' VB-developers.

## De basis van OO in VB .NET

*Declaratie van objecten*: de start voor OO in VB.NET wordt gemaakt door de manier waarop met objecten wordt omgegaan. En dit kende je al deels van vorige VB-versies. VB.NET gaat op een structurele manier om met objecten, te starten met het New keyword. Een standaard manier om nieuwe objecten aan te maken is met de volgende regels code:

```
Dim objNew As Persoon
objNew = New Persoon()
```

Het resultaat is een nieuw object (instance van de class Persoon), gereed




Afbeelding 1. Add New Item

voor gebruik. Een alternatieve manier om het binnen één coderegel te noteren is:

```
Dim objNew As New Persoon()
```

Deze declaratie maakt zowel een declaratie van het nieuwe object als een nieuwe instance. Had deze wijze van declareren in de voorgaande versies van VB een negatief effect op performance, in VB.NET is er geen verschil ten opzicht van de eerste wijze van declareren. Een derde variatie op hetzelfde thema is:

```
Dim objNew As Persoon = New
Persoon()
```

En dit is de meest flexibele en tegelijk een compacte vorm voor declaratie.

*Definitie van een class*: nu we weten hoe we een Object moeten declareren op basis van een classnaam, wordt het tijd een class te maken: de sjabloon van het object. In VB .NET zijn alle onderdelen met VB .NET code, (tekst)-files met de extensie .vb. Het maken van een class verloopt eenvoudig door in een Project een Add Class menuoptie te gebruiken (zie afbeelding 1).

Geef deze .vb file de naam MyClasses, en tik de volgende regels om de class Persoon aan te maken:

```
Public Class Persoon

End Class
```

Overigens weerhoudt niets ons ervan om meer dan één class per bestand aan te maken, bijvoorbeeld:

```
Public Class Persoon

End Class
Public Class Medewerker

End Class
```

Binnen de class hebben we variabelen nodig om bijvoorbeeld eigenschappen (properties) te bewaren.

```
Public Class Persoon
    '– Declareer hier class
variabelen
    Private mstrNaam As String
    Private mdtGeboren As Date
End Class
```

De scope (hier Private) van de variabelen kent de volgende keywords:
*Private*: de variabele is alleen beschik-

baar voor de code binnen de class
*Public*: de variabele is volledig beschikbaar buiten de class (niet aan te raden)
*Friend*: de variabele is alleen beschikbaar binnen het actuele project
*Protected*: de variabele is alleen beschikbaar binnen de class zelf en alle classes die van deze class 'inheriten'
*Protected Friend*: combinatie van de voorgaande twee. Alleen beschikbaar bij inherited classes van deze class binnen of buiten het actuele project.

## Properties in de class

Methods en properties aanmaken binnen de class verlopen eenduidiger dan in VB6. Zodra je in de editor begint met het tikken van het woorden Public Property, wordt automatisch een Property Get/Set-structuur aangemaakt:

```
Public Class Persoon
   Private mstrNaam As String
   Private mdtGeboren As Date

   '- Hier een Property Get/Set
tructuur
   Public Property Naam() As
String
        Get
            Return mstrNaam
        End Get
        Set(ByVal Value As
String)
            mstrNaam = Value
        End Set
    End Property

    Public Property Geboorte_
datum() As Date
        Get
            Return mdtGeboren
        End Get
        Set(ByVal Value As Date)
            mdtGeboren = Value
        End Set
    End Property

End Class
```

Nu is het eenvoudig geworden de benodigde code in het Get/Set-block in te voegen. Voor properties gelden dezelfde keywords als voor 'gewone' variabelen die op class-niveau beschikbaar zijn (Public, Private, Protected, Friend), met als extra aanvulling:

*ReadOnly*: de property kan alleen gelezen worden, er kan geen nieuwe waarde aan worden toegekend (en daarmee vervalt ook het Set-gedeelte van het Property-block)
*WriteOnly*: de property kan alleen een waarde gegeven worden en kan niet worden uitgelezen (en daarmee vervalt ook het Get-gedeelte van het Property-block)

## Methods in de class

Methods zijn er in twee varianten: een Sub, waarbij geen resultaatwaarde wordt teruggegeven, en een Function, waarbij als resultaat wel een waarde wordt teruggegeven. Ook hier zijn weer dezelfde keywords te gebruiken als voor variabelen die op classniveau beschikbaar zijn (Public, Private, Protected, Friend).

Een bijzondere (standaard) method is de New() method, ook wel constructor-method genoemd. De New() method is de vervanging van de Class_Initialize() method van VB6. Een New() method (Sub) van de Class wordt aangeroepen, zodra het object wordt aangemaakt. De New() method bestaat altijd, ook al definiëren we deze niet in onze eigen class.

Maken we wel een New() method in onze class aan, dan kunnen we deze gebruiken om bijvoorbeeld standaardwaardes in te stellen, zoals onderstaand voorbeeld weergeeft.

```
Public Class Persoon
   Private mstrNaam As String
   Private mdtGeboren As Date

   '- Maak een eigen New()
Method aan met defaults
   Public Sub New()
       mstrNaam = "Onbekend"
   End Sub

End Class
```

Binnen een class kunnen we meer dan één keer een method definiëren met dezelfde naam, dit heet Overloading. Het onderscheid tussen de ene en de andere method (met dezelfde naam) moet blijken uit verschillende argumenten die de method ontvangt. De unieke combinatie van argumenten (bij dezelfde method) heet het signature van de method. Dat kan natuurlijk ook voor de New() method:

```
Public Class Persoon
   Private mstrNaam As String
   Private mdtGeboren As Date

   '- De basisimplementatie van
New() zonder argument
   Public Sub New()
       mstrNaam = "Onbekend"
   End Sub

   '- En een tweede variant met
een argument
   '- deze "overloads" de eerste
 definitie
   Public Sub New(ByVal strNaam
as String)
       mstrNaam = strNaam
   End Sub

End Class
```

Nu hebben we een New() method waarmee we direct defaultwaardes aan het object kunnen meegeven. Dit kunnen we dan als volgt in een variabele declaratie gebruiken:

```
Dim objNew As Persoon = New_
Persoon("Jan Jansen")
```

Op basis van de signature bepaalt de .NET runtime-omgeving welke van de New() implementaties gebruikt moet worden.

## Class View in Visual Studio

Ben je eenmaal behoorlijk aan de slag met Classes, dan is de class view window een uitstekende manier om de structuur van alle classes, inclusief methods, properties, enzovoort, zichtbaar te maken.

## Shared Methods, Variables en Events

De methodes, properties, variabelen en events tot nu toe zijn elementen die

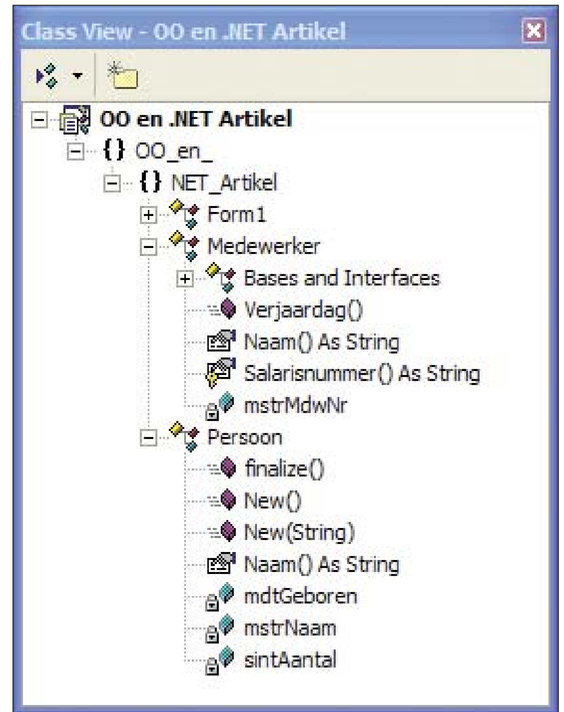
op instance-niveau (het objectniveau) beschikbaar zijn. Elk object heeft zijn eigen kopie, er kan pas aan gerefereerd worden wanneer het object bestaat. Binnen een class is het ook mogelijk Shared methods, variabelen, enzovoort te maken. Deze onderdelen zijn beschikbaar binnen de class, zonder dat er een object voor hoeft te bestaan. Shared methods zijn procedures die dus rechtstreeks aan de class gekoppeld zijn; shared variabelen zijn variabelen waarvan er maar één per class bestaat en door alle objecten van die class geshared wordt, onafhankelijk van het aantal objecten dat er op basis van de class zijn "geïnstantieerd".

Een voorbeeld van een Shared variabele, waarmee het aantal persoon-objecten dat is aangemaakt wordt geteld:

```
Public Class Persoon

   '—Van de volgende variabele
is er maar 1 per class
   Private Shared sintAantal as
Integer

   Public Sub New()
      '— Tel het aantal objecten
       sintAantal += 1
   End Sub

   '— Geef het aantal objecten
terug
   Public ReadOnly Property
AantalPersonen() As Integer
       AantalPersonen = sint_
Aantal
   End Sub

   Public Overloads Sub_
Finalize()
        sintAantal -= 1
   End Sub

End Class
```

De Finalize-methode verdient een extra toelichting. Dit is een vervanging van het Class_Terminate event van VB6. Het verschil is wel dat de Finalize-methode niet wordt uitgevoerd op het moment dat een object verdwijnt (bijvoorbeeld omdat het op = Nothing wordt gezet), maar pas wanneer de Garbage Collection van de .NET runtime wordt uitgevoerd. Dit is op een later, niet vooraf bekend moment.

Wil je toch direct na het verdwijnen van de objectreferentie een clean-up uitvoeren, dan dien je de IDisposable interface te implementeren, hetgeen te ver voert voor de diepgang van dit artikel.




Afbeelding 2. Class View Window

## Inheritance: hét voordeel van OO

Ik gebruik bij het trainen en opleiden van (gevorderde) ontwikkelaars vaak het (ook weer slim gekopieerde) motto: Goed programmeren is slim kopiëren. Hoe vaak komt het niet voor dat je een nieuw probleem moet oplossen en denkt: dat heb ik al eens een keer gedaan, of ik heb eerder een vergelijkbare programmeeroplossing gemaakt. Een goede developer maakt zijn code zodanig dat deze voor hergebruik in aanmerking komt, maakt handige functies en routines die meermalen worden benut. Dit kan natuurlijk door een echte kopie te gebruiken en deze aan te passen, maar nog 'slimmer' is het routines te maken die parametriseerbaar zijn. Met de juiste argumenten doet de routine wat je in die omstandigheden wilt. Verbeteringen aan de bron (de basisroutine) komen dan ten goede aan alle programma's die van dezelfde bron gebruik maken.

In dit opzicht is Visual Studio met alle producten erin en erbuiten een prachtig voorbeeld: de developers van Microsoft hebben onderdelen gemaakt, zoals de base classes, die wij met ons allen hergebruiken. Oftewel: wij profiteren van hun inspanningen!

Dezelfde motivatie geldt voor het begrip Inheritance binnen VB.NET. Dit kan, omdat VB.NET een volledige OO-taal is. De basis is het maken van een 'slimme' kopie van een stuk code (de base class), om vervolgens alleen aan te geven in je eigen versie van deze code wat je anders wilt hebben. En het 'slimme' is dat je niet daadwerkelijk een kopie maakt, maar een soort 'virtuele' kopie: de logica van de base class wordt overerfd (inherited) en blijft in zijn bronvorm bestaan. Een uiterst slimme vorm van kopiëren derhalve en dus een mogelijkheid voor zeer goed programmeren!

## Inheritance in de praktijk

Het 'virtueel' kopiëren en daarmee de start van het hergebruik is het keyword Inherits, zoals in het volgende voorbeeld aangegeven:

```
Public Class Persoon
   '— Hier komt de base code
End Class

Public Class Medewerker
   Inherits Persoon

   '— Hier komt de aanvullende_
code
End Class
```

Met het woord 'Inherits' krijg je alle mogelijkheden van de Class Persoon mee in Medewerker, de afgeleide van de base class Persoon. Kent de class Persoon een Naam en Geboortedatum-pro-

perty, dan is deze per direct ook in de class Medewerker beschikbaar. De afgeleide class is een verbijzondering ('Is A') van de base class. In de afgeleide class kunnen we nu aanvullende eigenschappen opnemen, bijvoorbeeld een salarisnummer van een medewerker.

```
Public Class Medewerker
   Inherits Persoon

   Private mstrSalNummer As
String

   '— Maak een nieuwe property
in de subclass aan
   Public ReadOnly Property
Salarisnummer() As String
        Get
            Return mstrSalNummer
        End Get
    End Property

End Class
```

Nu kent een object van het type Medewerker 3 properties: Salarisnummer (vanuit de class Medewerker), Naam en Geboortedatum (vanuit de class Persoon). Kortom: een virtuele kopie van de Persoon-definitie binnen de Medewerker.

## Overriding onderdelen van de base class

De kracht van Inheritance in VB.NET is niet alleen het virtueel kopiëren, maar zeker ook de mogelijkheid een deel van het gedrag van de base class naar eigen believen aan te passen: je programmeert alleen datgene dat je anders wilt. Hier komt het keyword Overrides aan de orde. Om de overschrijving van het basisgedrag mogelijk te maken, moet in de base class zijn opgegeven dat het overschreven mag worden. Dit gebeurt met het keyword Overridable:

```
Public Class Persoon
   Private mstrNaam As String
   Private mdtGeboren As Date

   '— Voordat we de naam mogen
 overschrijven in
   '— de subclass, moeten we
aangeven dat dit kan
   Public Overridable Property
Naam() As String
        Get
            Return mstrNaam
        End Get
        Set(ByVal Value As
String)
            mstrNaam =
Value.ToUpper()
        End Set
    End Property

   '— Hier komt de overige code

End Class
```

In de base class kan dus worden aangegeven dat een methode of een property overschreven mag worden. Stel dat we anders willen omgaan met de manier waarop nu met de naam van de medewerkers wordt gewerkt – in de Persoon class bestaat alles uit HOOFDletters. Dan kunnen we deze werkwijze in onze 'derived' Medewerkers class eenvoudig anders opgeven.

```
Public Class Medewerker
   Inherits Persoon

   '— Nu kunnen we onze eigen
implementatie maken
   Public Overrides Property
Naam() As String
        Get
            Return MyBase.Naam()
        End Get
        Set(ByVal Value As
String)
            MyBase.Naam = Value
        End Set
    End Property

End Class
```

In het voorbeeld zie je ook hoe je in de subclass toch weer gebruik kunt maken van de property die in de base class aanwezig is (MyBase keyword). Je hebt alleen het gedrag aangepast, oftewel: hergebruik van code met eigen toevoegingen.

Bij het opzetten van de base class hebben we de mogelijkheid om per property of methode aan te geven of, en zo ja, hoe overriding mag worden uitgevoerd. Hiervoor zijn de volgende keywords bij de definitie in de base class beschikbaar (naast de al eerder behandelde Protected, Private, enzovoort):

*NonOverridable*: de property (of method) kan niet overschreven worden. Dit is het default-gedrag voor een property of method van een base class.

*Overridable*: de property (of method) mag overschreven worden, maar dit hoeft niet.

*MustOverride*: de property (of method) moet overschreven worden. In dit geval geldt dat er geen uitwerking van de method of property in de base class wordt gemaakt en dat de base class als geheel het keyword MustInherit meekrijgt.

```
Public MustInherit Class
MyBaseClass

   '— Deze Property moet in de
subclass worden
   '— geïmplementeerd
   Public MustOverride Property
 MyProperty() As String

End Class
```

## Combinatie van base class declare en derived class instantiatie

Je kunt nu een verschil maken tussen een declaratie van een object en de instantiatie, bijvoorbeeld:

```
'— Een verschil tussen de
declare en de instatiatie
Dim objNew As Persoon
objNew = New Medewerker()
```

Het object zelf is nu van de class Persoon, maar is aangemaakt als een object van de afgeleide class: Medewerker. Dit lijkt complex, maar geeft ons prachtige mogelijkheden voor flexibiliteit (en voor het maken van generieke routines die bijvoorbeeld altijd werken voor objecten van het type Persoon én alle afgeleiden daarvan. Essentieel is om te onthouden dat het datatype van een reference variabele iets anders is dan

het datatype van het object zelf, en dat een reference variabele van een base class altijd een referentie kan bevatten van objecten van iedere subclass daarvan.

Nu wordt het nadenken om te bepalen (bij override en overload) op welk niveau de properties en methods worden aangeroepen. De volgende tabel helpt hierbij.

| Variabele | Object | Method of property die wordt gebruikt |
|---|---|---|
| Base | Base | Base |
| Base | Subclass | Subclass |
| Subclass | Subclass | Subclass |

## Er is nog veel meer!

OO binnen VB.NET is in dit artikel alleen behandeld op basis van de VB-programmacode. En hier heb ik diverse zaken niet behandeld, zoals Shadowing, Implementing Interfaces, MyClass, Event methods, enzovoort. Eigenlijk is OO in VB.NET-programmacode maar één onderdeel van OO binnen .NET als geheel. Alles binnen .NET is gebaseerd op objecten.

Zeer aantrekkelijk is het feit dat Windows-formulieren en controls ook objecten zijn die je kunt 'overerven', zodat je bijvoorbeeld het gedrag van een textbox zodanig kunt aanpassen dat jouw variant ontstaat, bijvoorbeeld om alleen numerieke waarden te accepteren. En omdat alle controls van .NET zelf te overerven zijn, hoef je alleen het 'verschil' tussen een gewone textbox en de door jou gewenste variant te programmeren. Ongetwijfeld is dan jouw code minimaal ten opzichte van een zelfde oplossing in VB6. Kortom: met de juiste kennis stijgt jouw productiviteit (minder code), krijg je beter onderhoudbare code (minder code) en wordt de kwaliteit en flexibiliteit van jouw applicaties beter (minder code, meer hergebruik).

## Een afsluitend woord

Hoe zet je nu op de juiste manier Inheritance op? Hier is geen enkelvoudig recept voor te geven. Algemeen advies: begin eenvoudig, los niet te veel eigenschappen en gedrag in een base class op, maar denk alleen na over het algemene gedrag en de algemene eigenschappen.

Omdat OO (binnen .NET) zo flexibel is, kun je in een later stadium zonder veel moeite structuren aanpassen, nieuwe eigenschappen toevoegen, en het gedrag van objecten veranderen. Tegelijkertijd schuilt hier ook het gevaar om de boom van base classes en afgeleide classes uit te veel niveaus te laten bestaan; zie een diepgang van 4 tot 7 niveaus als praktisch bruikbaar. Meer niveaus maakt het onoverzichtelijk (en komt de performance niet ten goede), minder niveaus leiden ertoe dat je elke class helemaal aan het uitschrijven bent, in plaats van te werken met een algemene opzet met afwijkingen in derived classes. En bedenk ook dat je vaak eenvoudig start, alles overerft zonder veel aanpassingen om pas in een later stadium de verschillen te willen aanbrengen.

In dit licht gezien lijkt Inheritance precies op de biologische overerving. Als je kinderen hebt en ze nog jong zijn, doen ze alles precies na wat de 'base class', de ouders doen. Wat de ouders doen is goed, de subclass (kinderen) willen precies hetzelfde worden, er is geen behoefte aan overriding. Later in de tijd ontstaat de behoefte alles anders te doen, zeker bij de subclass in pubertijd.

Het gedrag van de base class is per definitie verkeerd, alles wordt overschreven. Tel je zegeningen en wees blij als je als base class op dat moment een paar NonOverridable eigenschappen op de juiste wijze hebt geïmplementeerd. De base class is nauwelijks meer aan te passen. Als je tegelijkertijd ook nog wat aardige properties hebt aangebracht in de base class (ouders) en in de subclass (de kinderen), dan is er later ongetwijfeld weer een moment waarin de structuur van overerving tot een werkzaam geheel komt.

Eén ding moet je na lezen van dit artikel duidelijk zijn: OO biedt grote voordelen voor de professionele developer, het concept is eenvoudig en volledig binnen VB.NET doorgevoerd en beschikbaar. Een nieuw tijdperk breekt aan. Veel plezier met je overerving!

### Nuttige internetadressen

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchooinvbnet.asp
  Object-Oriented Programming in Visual Basic .NET
- http://www.microsoft.com/mspress/developer/feature/040402.asp
  Understanding Inheritance in Visual Basic: It Doesn't Get Any Easier Than This
- http://www.microsoft.com/mspress/books/5199.asp
  Programming Microsoft® Visual Basic® .NET (Core Reference)
- http://msdn.microsoft.com/vbasic/techinfo/articles/upgrade/transition/default.asp
  The Transition from Visual Basic 6.0 to Visual Basic .NET
- http://www.gotdotnet.com/team/vb/
  .NET Framework Community website