Inhoud

1 Een beetje theorie vooraf 1

Wat is een programma? 2
Hoe maakt u een programma? 3
IL-code en JIT-compiler 5
Voordelen combinatie compiler en interpreter 6
Het .NET Framework 9
.NET en Java 9
.NET, COM en de Babylonische spraakverwarring 10
.NET, het netwerk en internet 11
Componenten van .NET Framework 12
Framework, SDK en bronnen 13
Visual C# 14
IDE 15
RAD 17
Componenten 19
Samenvatting 20
Voorbeelden maken 20
Oefening 22

2 Het eerste programma 23

Start met een project 24
Brontekst schrijven 28
Program.cs 29
Form1.cs en Form1.Designer.cs 39
namespace en using 44
Programmeren in RAD 45
RAD-omgeving 45
Componenten toevoegen 50
Componenten configureren 52
Events afhandelen 54
Runtime-eigenschappen aanpassen 56
Eigenschap Name 58

Activiteiten vaste schijf **58**
Programma draaien zonder Visual C# **59**
Samenvatting **61**
Vragen en antwoorden **61**
Oefeningen **62**

3 Basiscursus C#: dataverwerking 63

Consoletoepassingen **64**
Toepassingsskelet 65
Consoletoepassingen in Visual C# 65
Consoletoepassingen uitvoeren zonder Visual C# 68
Datatypen en variabelen **69**
Waarde van variabelen 72
Variabelen tijdens de declaratie initialiseren 73
Waarde van variabelen uitlezen 73
Elementaire datatype 74
Strings 76
Typeconversie 83
C# voor pietje-precies 89
Variabelen versus constanten **90**
Operatoren **91**
Verschillende operatoren 92
Goniometrische methoden gebruiken 94
Deling 97
Objecten en klassen **98**
C# voor filosofen 98
Klassen declareren 100
Programmeren met klassen 107
Arrays **112**
Arrays declareren 112
Array-elementen benaderen 113
Arrays initialiseren 114
Kant-en-klare klassen en namespaces **115**
Samenvatting **117**
Vragen en antwoorden **117**
Oefeningen **118**

4 Basiscursus C#: modulair werken en programmasturing 121

Modulair werken met klassen en methoden **122**
Deeltaken implementeren in methoden I 124
Deeltaken implementeren in methoden II 125
Deeltaken implementeren in klassen 126
Eigen bibliotheken 132

Controlestructuren **138**
Selection statement: if-else 138
Booleaanse expressies 139
Selection statement: switch 142
Loops: for, while en foreach 143
Loopvariabelen en afbreekvoorwaarden 145
Loops en arrays 146
Toveren met lussen, getallen en konijntjes 148
Veranderen van de programmastroom 149
Foutafhandeling met exceptions **150**
Exceptions afvangen 151
Verschillende catch-blokken 154
De exception-parameter 156
Eigen exceptions 157
Samenvatting **157**
Vragen en antwoorden **158**
Oefeningen **159**

5 Basiscursus C#: OOP-verdieping 161

Statische en niet-statische class members **162**
Niet-statische klasseleden 162
Statische elementen 165
Methoden **167**
Declaratie van methoden 167
Teruggegeven waarde van methoden – return 168
Parameters en argumenten 169
Overloading 173
Toegang en toegangsbeperking **175**
Geldigheid 175
Lokale variabelen en name hiding 176
Access modifiers 177
Eigenschappen **179**
Overerving **184**
Fundamenteel mechanisme 184
Access modifier protected 191
Aanroep constructor basisklasse 194
Hiding en overriding 196
Samenvatting **200**
Vragen en antwoorden **200**
Oefeningen **201**

6 Basiscursus C#: OOP-overzicht 203

Polymorfisme 204
Object 208
Override ToString() 209
Boxing 211
Interfaces 212
Interfaces declareren 212
Interfaces implementeren 213
Samenvatting 216
Vragen en antwoorden 216
Oefeningen 217

7 Basiscursus C#: I/O en bestanden 219

Schrijven naar de console 220
Write en WriteLine 220
Uitvoer formatteren 221
Schrijven in bestanden 223
Lezen van het toetsenbord 225
Lezen uit bestanden 227
Argumenten op de opdrachtprompt 229
Samenvatting 232
Vragen en antwoorden 232
Oefeningen 233

8 Basiscursus C#: nuttige klassen in .NET 235

Datum en tijd 236
Datum en tijd opvragen 236
Datum en tijd manipuleren 237
Datum en tijd uitvoeren 237
Tijdsverloop meten 239
Toevalsgetallen 240
Collection classes 242
De geschiedenis 243
Collections vergeleken 244
ArrayList 245
List 248
Dictionary<TKey,TValue> 251
Stack 252
Samenvatting 254
Vragen en antwoorden 254
Oefeningen 255

9 Rondleiding door Visual C# 257

Start **258**
Projectbeheer **258**
Projectbeheer gebruiken 258
Project maken 259
Projecten opslaan, sluiten en openen 259
Met projecten werken 260
Projecteigenschappen 262
Solutions 265
Editor **269**
Syntaxis accentueren 270
Automatisch inspringen 270
IntelliSense 271
Samenwerking met compiler en debugger 273
Uitgebreide functies 273
Windows Forms Designer **273**
Componenten toevoegen en wissen 273
Componenten selecteren 274
Componenten dimensioneren 274
Componenten uitlijnen 275
Componenten kopiëren 275
Venster Properties 276
Compiler **277**
Debugger **279**
Fouten en debugging 279
Verloop debuggingsessie 280
Voorbereidingen debuggen 281
Programma in debugger laden en starten 281
Programma stoppen 282
Programma stap-voor-stap uitvoeren 284
Vensters van Debug 284
Help **286**
Online hulp 287
Lokale hulp 288
Configuratie **290**
Samenvatting **290**
Vragen en antwoorden **290**
Oefeningen **291**

10 Windows-toepassingen: forms en controls 293

Windows-toepassingen 294
Venster, hoofdvenster en forms 296
Wat is een venster? 297
Venster configureren 298
Programmapictogram 304
Controls 307
Programmeren met controls 307
Opschrift 308
Knoppen 310
Checkbox 312
Radiobuttons en GroupBox 313
Invoervak 314
Keuzelijst 315
Keuzelijst met invoervak 317
Meer controls en informatie 318
Eventafhandeling 318
Eventafhandeling opzetten 319
Welke events afhandelen 320
Samenvatting 322
Vragen en antwoorden 322
Oefeningen 323

11 Windows-toepassingen: menu's en werkbalken 325

Menubalken 326
Opbouw van een menubalk 326
Eventafhandeling voor menuonderdelen 330
Menuonderdelen configureren 332
Werkbalken 332
Snelmenu's 334
Samenvatting 335
Vragen en antwoorden 336
Oefeningen 336

12 Windows-toepassingen: dialogen 337

Dialoogvensters **338**
Dialoogvensters opbouwen en configureren **338**
Dialoogvensters maken en tonen **341**
Instellingen in dialoogvensters uitlezen **344**
Standaarddialoogvensters **346**
Meldingsvenster 346
Bestanden openen 347
Samenvatting **349**
Vragen en antwoorden **349**
Oefeningen **349**

13 Windows-toepassingen: grafisch talent 351

Gereedschap voor kunstenaars **352**
Tekst tekenen 352
Tekeningen reconstrueren – Paint-event 354
Tekenmethoden – Graphics 355
Penseel, potlood en lettertype 357
Tekenen in panels **358**
Gebruikersinterface 359
Functies kiezen 360
Functies plotten 360
Tekenen uit de vrije hand **362**
Concept tekenen uit de vrije hand 363
Een Graphics-object maken 364
Afbeeldingen tonen **367**
Afbeeldingsbestand laden 367
Afbeeldingen tonen 368
Bitmapviewer 368
Samenvatting **371**
Vragen en antwoorden **371**
Oefeningen **372**

14 Databases – ADO.NET 373

Relationele databases en SQL 374
SQL – de basis 376
Toegang tot een SQL Server-database 379
Database 379
Toepassing 385
Toegang tot een Microsoft Access-database 391
Project kopiëren 391
Database maken 392
Brontekst voor toegang database aanpassen 392
Databasetoepassingen met volle Visual C#-ondersteuning 393
Zelfgebouwde databasetoepassingen 395
Basisproject 396
Samenspel van databaseklassen 397
Code 398
Samenvatting 402
Vragen en antwoorden 403
Oefeningen 403

15 Screensaver 405

Bijzondere Windows-toepassing 406
Ticker als screensaver 407
Configuratie van het venster 407
Sluiten bij muisklik 407
Startargumenten toekennen 408
Animatie 410
Screensaver inrichten 413
Samenvatting 413
Vragen en antwoorden 414
Oefeningen 414

16 Hoe gaat het verder? 415

A Oplossingen 417

Antwoorden oefeningen 418

B Probeerversie Visual C# 429

Visual StudioExpress 2012 **430**
Systeemeisen 430
Downloaden 430
Voorbeelden **431**

C Unicode-tabel 433

Unicode-tekenset **434**

D Syntaxisoverzicht 437

Keywords **438**
Elementaire datatypen **438**
Strings **439**
Formattering met ToString() 440
Operatoren **441**
Loops en vertakkingen **443**
Vertakkingen 443
Loops 443
Goto-sprongen 444
Exceptions **444**
Enumeraties **445**
Arrays **446**
Eendimensionale arrays 446
Meerdimensionale arrays 446
Arrays van arrays 446
Programmering 446
Interface **447**
Delegates **447**
Events **448**
Structs **449**
Klassen **449**
Definitie 449
Overerving 451
Partial-klassedeclaratie 452
Generics **452**
Generic class 452
Generic-methoden 453

E Verklarende woordenlijst 455

Index 467

Hoofdstuk 1

Een beetje theorie vooraf

Dit hoofdstuk geeft u een goede basis waarop u in de volgende hoofdstukken verder bouwt. Hebt u nog nooit geprogrammeerd en weet u van C# (spreek uit: sie-sharp) niets meer dan de naam, dan vindt u hier enkele belangrijke begrippen die u nodig hebt. Hebt u al programmeerervaring met andere talen opgedaan, neem dan ten minste de onderdelen over C#, .NET Framework en Visual C# door. Maar jeuken uw handen en wilt u liever direct beginnen met programmeren, ga dan naar het volgende hoofdstuk en keer terug naar dit hoofdstuk wanneer u meer over de achtergronden van C# en .NET wilt weten.

U leert in dit hoofdstuk:

Wat een programma is.

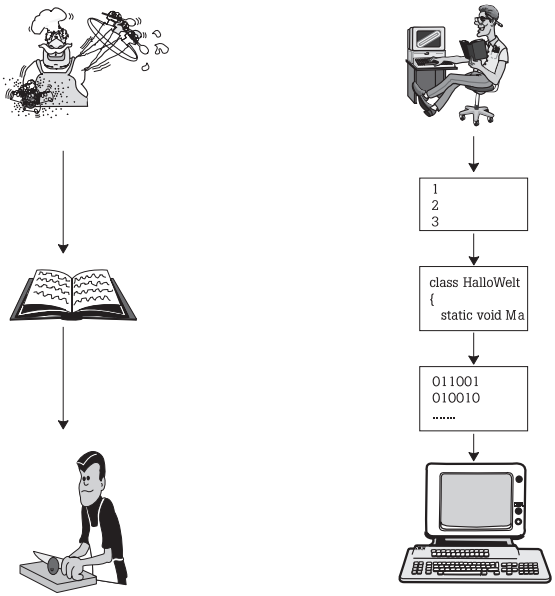
Hoe een C#-brontekst verandert in een uitvoerbaar C#-programma.

Wat precies het .NET Framework is en waarom het onmisbaar is.

Welke rol Visual C# speelt bij het tot stand komen van C#-programma's.

Wat is een programma?

Een programma is niets anders dan een reeks opdrachten die de computer moet uitvoeren. Vergelijk het maar met een recept in een kookboek. Een recept is een lijst met ingrediënten en handelingen die noodzakelijk zijn om een lekker gerecht te maken. De kok schrijft nauwkeurig op wat nodig is, wat u moet doen en in welke volgorde. Is het recept geschreven, dan kan elke kookliefhebber het gerecht klaarmaken door het recept te volgen, wanneer en hoe vaak hij maar wil. In deze analogie is de programmeur de kok, het recept is het programma en de kookliefhebber is de computer.



Afbeelding 1.1 *Analogie tussen een recept uit een kookboek en een computerprogramma.*

Dat is de theorie, maar er zijn – natuurlijk – wel wat complicaties. Voor Nederlandse kookliefhebbers levert het volgen van een Nederlandstalig recept geen enkel probleem op; de kok en kookliefhebbers spreken dezelfde taal. Maar hoe zit dat met programmeurs en computers? Wel, de programmeur spreekt Nederlands en de computer begrijpt daar geen woord van. De computer – of beter, de processor – verstaat slechts een zeer beperkte set van elementaire opdrachten, de zogenoemde machinetaal die ook nog eens binair gecodeerd is en dus bestaat uit een reeks van nullen en enen.

Wilt u een programma schrijven dat de processor begrijpt, dan bestaat dat geheel uit lange reeksen enen en nullen. Het probleem daarmee is dat het voor de meeste mensen volslagen onleesbaar is. Erger nog, als er bij het invoeren van dat programma een foutje wordt gemaakt – zoals een 1 in plaats van een 0 – dan is het niet eenvoudig om dat foutje op te sporen. Aangezien het onwaarschijnlijk is dat de processor op korte termijn natuurlijke talen – zoals het Nederlands – leert begrijpen, zult u de computer wat tegemoet moeten komen.

Terug naar de recepten, stel dat een Chinese meesterkok een kookboek schrijft dat in Nederland moet verschijnen. De Chinese kok is niet in staat om Nederlands te schrijven, maar zijn kennis van de Engelse taal is voldoende om het boek te schrijven, waarna een vertaler het boek uit het Engels vertaalt in het Nederlands. Iets dergelijks gebeurt bij het schrijven van een programma. In plaats van het programma in het Nederlands te schrijven, gebruikt u een programmeertaal (zoals C#, C++, Java, Pascal, Visual Basic enzovoort) waarvoor een passende vertaler bestaat – ook wel compiler genoemd – die de opdrachten in machinetaal kan omzetten.

Maar dan rijst de vraag: wat is dan precies het programma? De nog in het Nederlands geformuleerde opdrachten, de in C# geschreven regels of de binair gecodeerde opdrachten in machinetaal? In de breedste zin van het woord zijn ze allemaal op te vatten als programma, maar dat is geen werkbare definitie. Daarom noemen we de in het Nederlands geformuleerde opdrachten het algoritme, de in C# geschreven versie noemen we de broncode en de door de compiler geproduceerde machinecode is het uitvoerbare programma.

Hoe maakt u een programma?

Een programma komt tot stand volgens een aantal vaste stappen:

1. **Formulering probleem** U hebt een probleem of een taak die u met behulp van een computer wilt aanpakken.
2. **Opstellen algoritme** In het Nederlands opstellen van de stappen die nodig zijn om de taak uit te voeren. Grotere problemen worden daarbij opgedeeld in kleinere deelproblemen.
3. **Implementatie algoritme** Schrijf het algoritme in voor de computer begrijpelijke opdrachten in een programmeertaal, dit resulteert in de broncode.
4. **Compilatie programma** De compiler (een speciaal programma) vertaalt de broncode naar binaire code (machinetaal) die de processor begrijpt en kan uitvoeren.
5. **Uitvoeren programma** De computer laadt het programma in het werkgeheugen en de processor voert het programma uit.

Op papier of niet?

Of het algoritme daadwerkelijk op papier wordt gezet of alleen in het hoofd van de programmeur wordt ontwikkeld, hangt af van de complexiteit van de opgave en de genialiteit van de programmeur.

Voorbeeld

1. U rekent iets af en u wilt weten hoeveel btw u over dat bedrag hebt betaald.
2. Het algoritme is heel eenvoudig en bestaat uit vier deelopgaven:
   - het totaalbedrag aan de gebruiker vragen;
   - het bedrag in het programma inlezen;
   - het aandeel btw berekenen;
   - het resultaat weergeven.
3. Het algoritme wordt in voor de computer begrijpelijke opdrachten van een programmeertaal omgezet, dit resulteert in de brontekst of broncode.

De brontekst voor ons voorbeeld zou er ongeveer zo kunnen uitzien:

```
using System;
namespace BTW_berekening
{
  class Program
  {
    static void Main(string[] args)
    {
      double prijs;
      double btw;
      // 1. Invoer van gebruiker vragen
      Console.Write(" Voer het bedrag in: ");
      // 2. Invoer inlezen
      string invoer = Console.ReadLine();
      prijs = Convert.ToDouble(invoer);
      // 3. BTW-bedrag berekenen
      btw = 0.21 * prijs / 1.21;
      // 4. Berekende BTW-bedrag weergeven
      Console.WriteLine(" In de prijs is {0:#.00} "
      + "Euro BTW inbegrepen \n", btw);
    }
  }
}
```

Kennismaking

Lijkt deze code onbegrijpelijk, dan hoeft u zich daarover geen zorgen te maken. Ten eerste is deze code al vrij complex, ten tweede is het de bedoeling u te laten kennismaken met een stukje C#-brontekst, niet dat u de code volledig doorgrondt.

Bij sommige programmeertalen zijn de stappen compilatie en uitvoer van het programma samengevoegd: in plaats van het hele programma te vertalen en dan uit te voeren, wordt een interpreter (tolk) gebruikt. De interpreter leest een regel van de brontekst en vertaalt deze, waarna de processor de regel uitvoert, waarna de interpreter het hele proces herhaalt met de volgende regel. Een geïnterpreteerde computertaal slaat een programma dus op als tekstbestand en niet als uitvoerbaar programma (machinecode).

Twee grote nadelen van interpreters:

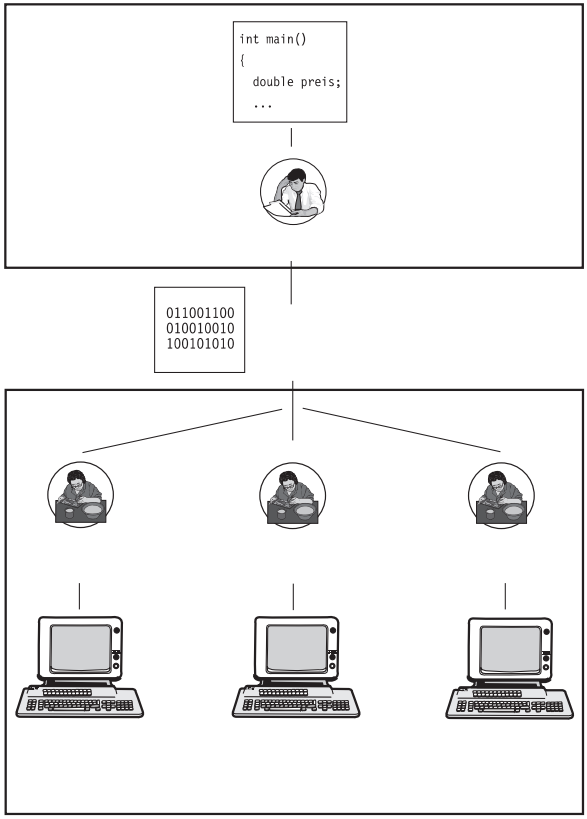
- De omzetting in machinecode vindt pas tijdens de uitvoering van het programma plaats, waardoor zulke programma's duidelijk langzamer draaien dan gecompileerde programma's.
- Deze programma's worden als brontekst verspreid en zijn dus leesbaar met elke teksteditor, zodat oneigenlijk gebruik van de broncode en de voor het programma ontwikkelde algoritmen niet of nauwelijks valt te voorkomen.

Aan de andere kant hebben deze programma's het voordeel dat ze makkelijk te exporteren zijn, dat wil zeggen, de vertaling naar andere computersystemen is eenvoudig.

Bekende compilertalen zijn C, C++ en Pascal. Voorbeelden van geïnterpreteerde computertalen zijn Basic, Perl en scripttalen. En C#?

IL-code en JIT-compiler

C# is zowel een interpreter als een compilertaal. Verwarrend? Nee, niet echt. De compiler *csc* vertaalt C#-brontekst naar een tussencode – *Intermediate Language Code* – ofwel IL-code. U zou kunnen zeggen dat IL-code de machinecode is voor een niet-bestaande (virtuele) processor, maar om de IL-code te kunnen uitvoeren op een echte computer, vertaalt een interpreter de IL-code tijdens het draaien van het programma. Deze interpreter is platformafhankelijk, dat wil zeggen, deze maakt de vertaalslag naar de machinetaal van de in de computer aanwezige processor.



Afbeelding 1.2 *Een C#-programma wordt gecompileerd en geïnterpreteerd.*

Om een C#-programma te kunnen uitvoeren hebt u dus zowel een compiler als een interpreter nodig. Deze combinatie van compiler en interpreter biedt de voordelen van beide systemen, maar met minder nadelen.

Voordelen combinatie compiler en interpreter

Beschermde code

Aangezien u als C#-programmeur uw programma's verspreidt als gecompileerde IL-code, is het risico kleiner dat piraten uw programmacode en algoritmen zonder uw toestemming gebruiken voor andere programma's.

Geldwolven?

Ik wil niet de indruk wekken dat programmeren alleen over geld verdienen gaat – ook wanneer de media en de hernieuwde opleving van de IT-industrie dit wellicht suggereren. Velen programmeren louter voor het plezier, uit nieuwsgierigheid of enthousiasme. Mijn ervaring is dat de meeste mensen uit interesse met programmeren beginnen en er pas later hun beroep van maken. Ook zijn er veel programmeurs die uit idealisme hun programma's beschikbaar maken op internet, waar het gratis als freeware is te downloaden – vaak zelfs inclusief brontekst zodat anderen daarvan kunnen leren. Het punt is, dat u met een gecompileerde of voorgecompileerde taal zelf kunt kiezen of u alleen het binair gecodeerde programma – in geval van C# ook de IL-code – of ook de brontekst aan de gebruiker ter beschikking wilt stellen.

Snelheid

De interpretatie van voorgecompileerde IL-code is veel sneller dan van pure brontekst. De interpreter voor IL-code is zelfs zo snel dat deze wordt aangeduid als Just-in-Time-Compiler. De Just-in-Time-Compiler, ook JIT-compiler of Jitter genoemd, verwerkt de IL-code van het programma in delen en naar behoefte. Ter verduidelijking: een compiler zou de gehele IL-code in het geheugen inlezen, vertalen en uitvoeren. Een groot programma wordt dan snel uitgevoerd, maar het laden duur lang. Een typische interpreter voert de broncode regel voor regel uit, bij een groot programma gaat het inlezen snel, maar de uitvoering is tergend langzaam omdat de uitvoering steeds wordt onderbroken voor de vertaling van de volgende opdrachten. De Just-in-Time-Compiler lost dit probleem op met een intelligent compromis. Hij vertaalt de IL-code in delen, waarbij hij steeds dat deel van de code opzoekt dat de gebruiker nodig heeft. Wanneer de gebruiker bijvoorbeeld een in C# geschreven tekstverwerkingsprogramma start om een tekstbestand te bewerken, vertaalt de Jitter de code om het programma te starten en ook voor het laden en bewerken van tekst, maar niet de functies voor spellingcontrole, zoek en vervang en dergelijke. Pas als de gebruiker spellingcontrole voor de eerste keer aanroept, wordt deze code vertaald en uitgevoerd. Na de eerste aanroep blijft de vertaalde machinecode in het werkgeheugen van de computer. Bij de volgende aanroep van spellingcontrole hoeft de Jitter dus niet meer in te grijpen en is de spellingcontrole direct beschikbaar.

Methoden

Voor wie bekend is met objectgeoriënteerd programmeren, bij de hierboven vermelde codedelen die de Jitter naar behoefte vertaalt, gaat het om de *methoden* in de *C#-klassen*.

Platformonafhankelijkheid

IL-code is platformonafhankelijk. U vertaalt de programmabrontekst op uw computer met behulp van de C#-compiler en het resulterende programma is bruikbaar op alle computers die een JIT-compiler geïnstalleerd hebben. De JIT-compiler is een vast onderdeel van het .NET Framework. Het .NET Framework maakt onderdeel uit van alle Windows-besturingssystemen vanaf Windows XP (Service Pack 3), zodat praktisch elke Windows-gebruiker, of hij zich daarvan bewust is of niet, beschikt over een geïntegreerde JIT-compiler voor de uitvoering van IL-programma's.

Hoe komt u in het bezit van een JIT-compiler?

Op de huidige Windows-systemen kunt u .NET Framework zelf installeren – download het .NET Framework van de Microsoft-website **msdn.microsoft.com/netframework** of **www.microsoft.com/downloads** of installeer het samen met op .NET gebaseerde software, bijvoorbeeld Visual Studio Express 2012.

Gebruiksvriendelijk

De JIT-compiler heeft nog meer te bieden. Om een geïnterpreteerd programma uit te voeren, is het nodig dat de gebruiker vanaf de opdrachtprompt de interpreter start en de brontekst aanbiedt:

```
Perl mijnScript.pl     //starten van een Perl-programma
Java mijnProgramma     //starten van een Java-programma
```

Voor de gebruiker die gewend is een programma te starten met een dubbelklik op het programmapictogram of met een dubbelklik op een te bewerken document, is het starten van een interpreter een omslachtige handeling. C#-programma's laten zich net als gecompileerde programma's starten op de voor de gebruiker vertrouwde wijze. Het besturingssysteem – of beter, het .NET Framework, herkent dat het uitvoerbare bestand uit IL-code bestaat en start automatisch en ongemerkt de JIT-compiler.

Veilig en elegant

IL-code draait in de runtimeomgeving van het .NET Framework, hetgeen grote voordelen biedt. Niet alleen heeft de programmeur een grote bibliotheek met klassen en objecten tot zijn beschikking die hij direct in zijn programma's kan gebruiken, maar ook zaken als programma-uitvoering, versiebeheer en het beheer van systeembronnen voor het programma – zoals het reserveren en weer vrijgeven van werkgeheugen, bestanden en netwerkverbindingen – neemt .NET voor zijn rekening. Dit waarborgt de systeemintegriteit: de computer crasht niet als de programmeur een foutje maakt met de afhandeling van systeembronnen. Bovendien neemt dit de programmeur werk uit handen.

Ervaren programmeurs kennen dit wellicht al van Java, voor de beginnende programmeur betekent dit dat het programmeren eenvoudiger is.

Taalonafhankelijkheid

Tot slot is de IL-code onafhankelijk van de gebruikte programmeertaal. De JIT-compiler vertaalt IL-code en voert deze uit in de .NET-omgeving, waarbij het niet uitmaakt waar de IL-code vandaan komt en in welke programmeertaal de brontekst was geschreven. Als een brontekst in correcte IL-code is vertaald, dan draait het programma op elke computer waarop .NET is geïnstalleerd, ongeacht of het daarbij om een van oorsprong C#-, Visual Basic- of C++-programma gaat. In de .NET-omgeving kunt u iedere programmamodule in IL-code opnemen en gebruiken. Dus als u een C#-klasse voor het beheer van adressen hebt geschreven die u wilt aanbieden, dan kan een programma in elke .NET omgeving waar uw klasse is geïnstalleerd de functionaliteit van uw klasse gebruiken. Omgekeerd kunt u ook in uw programma's modulen van andere programmeurs gebruiken – onafhankelijk van de programmeertaal waarin deze modulen zijn geschreven. Het is mogelijk om in een C#-programma de functionaliteit van bijvoorbeeld een bestaande Visual Basic-klasse te gebruiken en u kunt zelfs uw eigen C#-klasse daarvan afleiden.

Het .NET Framework

Simpel gesteld is het .NET Framework een JIT-compiler met een hoogwaardige codebibliotheek, of beter, het .NET Framework is niets anders dan een kader waarbinnen C#-programma's – of meer algemeen: .NET-toepassingen – worden uitgevoerd.

.NET en Java

De introductie in 1996 van SUNs nieuwe programmeertaal Java wekte de belangstelling van programmeurs en ontwikkelaars: een geïnterpreteerde programmeertaal voor het schrijven van professionele gebruikerssoftware. Java is syntactisch sterk verwant aan het veelgebruikte C++ – wat de overgang vergemakkelijkte – maar is geheel herzien, ontdaan van tierelantijnen en gemoderniseerd. Een compiler vertaalt Java-programma's in een virtuele tussencode, zodat eenmaal geschreven Java-toepassingen op iedere computer werken waarop de Java Virtual Machine is geïnstalleerd. De Java Virtual Machine bestaat uit een passende interpreter, een omvangrijke runtimebibliotheek en enkele andere tools.

Hier kon Microsoft natuurlijk niet werkloos toekijken, dus zag het .NET-initiatief het daglicht in de zomer van 2000. Het .NET Framework is niets anders dan de Microsoft-tegenhanger van de Java Virtual Machine, waarbij het .NET Framework natuurlijk geen Java-tussencode uitvoert, maar de door Microsoft gedefinieerde IL-code die bijvoorbeeld door compilatie van een C#-programma is ontstaan. De taal C# werd overigens speciaal voor .NET en het .NET Framework ontwikkeld en doet sterk denken aan Java. Het is in elk geval niet de enige taal die in IL-code kan worden gecompileerd.

.NET, COM en de Babylonische spraakverwarring

Microsoft en de ontwikkelaars voor het Microsoft-platform dromen al jaren van universeel te gebruiken binair gecodeerde softwarebouwstenen waarmee efficiënt en snel software kan worden ontwikkeld. Efficiënt software ontwikkelen kan tegenwoordig alleen als voor deelproblemen al oplossingen beschikbaar zijn die eenvoudig in het programma kunnen worden opgenomen. Veel softwareontwikkelaars stellen oplossingen beschikbaar die ze voor interessante problemen hebben gevonden, net als algemeen bruikbare programmabouwstenen – het maakt niet uit of dat nu code is om sudokupuzzels te maken, een digitaal klokje, een rentecalculator of een verzameling statistische functies. Andere ontwikkelaars hoeven het wiel niet opnieuw uit te vinden en gebruiken deze bouwstenen snel en comfortabel in hun programma's. Sommige programmeurs maken hun oplossing beschikbaar als brontekst, anderen compileren naar machinetaal en verkopen hun werk als deel van een softwarebibliotheek of als zelfstandige componenten.

Het nadeel is dat die bronteksten en bibliotheken alleen bruikbaar zijn in programma's die in dezelfde taal zijn geschreven . Nu zijn er voor het Windows-platform veel verschillende programmeertalen beschikbaar – en in gebruik – voor de ontwikkeling van toepassingen. En in dat *Babel der babbels* is het vinden van geschikte modulen in de juiste taal niet makkelijk. Vandaar dat Microsoft al vroeg een algemene standaard heeft opgesteld die beschrijft hoe componenten moeten zijn opgebouwd , zodat ze door willekeurige toepassingen gebruikt kunnen worden. Deze standaard was COM.

COM maakte het mogelijk dat bijvoorbeeld de bouwstenen van een Visual Basic-programmeur ook bruikbaar waren in een C++-programma. Sterker nog, dankzij COM konden toepassingen geschreven worden die in staat waren willekeurige COM-bouwstenen uit te voeren waarvan niemand zich realiseerde dat het COM-bouwstenen zijn. En de uitbreidingen van de COM-standaard zoals DCOM, OCX en ActiveX maakten een uitwisseling van COM-bouwstenen over netwerken en internet mogelijk.

Klinkt dat te mooi om waar te zijn? Tja, jammer genoeg is de COM-specificatie zeer formeel, omslachtig en gecompliceerd, zodat het schrijven van COM-bouwstenen voor menig programmeur een nachtmerrie bleek. Een nachtmerrie waaruit we nu kunnen ontwaken. Want het .NET Framework vervult de oude droom van taalonafhankelijke uitwisseling van binaire componenten. De truc is dat de compiler van iedere .NET-taal de brontekst vertaalt in IL-code. Tegelijkertijd organiseert de compiler de code in zogenoemde *assemblies* (bestanden die naast de IL-code ook aanvullende informatie bevatten) die het andere toepassingen mogelijk maakt de IL-code uit te voeren.

- Als u dus in C# een verzameling van koersanalysefuncties hebt geschreven die u als bibliotheek wilt verkopen, geen probleem! Compileer de code in een bibliotheekassembly en maak deze beschikbaar als download. De bibliotheek kan dan door alle programmeurs worden gebruikt die ook voor het .NET Framework ontwikkelen, ongeacht welke programmeertaal ze daarvoor gebruiken. En als u in een toepassing die u in C# schrijft een bepaalde component – bijvoorbeeld een digitaal klokje – wilt gebruiken die iemand in Visual Basic.NET heeft geschreven, geen probleem. Haal de assembly op, meld de assembly aan in uw toepassing en gebruik hem. En mocht de assembly niet op uw computer staan maar alleen op een bepaalde computer in uw netwerk of op internet, dan is dat ook geen probleem, want het .NET Framework laat de grenzen tussen uw computer en het netwerk of internet verdwijnen.

.NET, het netwerk en internet

Waren computers in het verleden veelal geïsoleerde werkstations, tegenwoordig zijn ze meestal in netwerken verbonden of ze staan tenminste via een telefoonlijn in verbinding met internet. Navenant zijn er steeds meer toepassingen waarvan de code over meerdere computers verdeeld is of die de functionaliteit gebruiken van andere computers (van servers tot webdiensten).

Met het .NET Framework vervagen de grenzen tussen de lokale computer en het net, zoals uit het volgende voorbeeld duidelijk wordt. Als u een traditionele toepassing uitvoert, zorgt het besturingssysteem op de computer ervoor dat de toepassing correct wordt gestart. Daarbij zoekt het besturingssysteem alle componenten bij elkaar die voor de uitvoering nodig zijn en stopt de uitvoering wanneer het niet alle componenten op de computer kan vinden. .NET-toepassingen worden in de omgeving van het .NET Framework uitgevoerd en deze staat toe dat programmacomponenten – de eerder vermelde assemblies – van verschillende plaatsen op het netwerk of internet worden opgehaald.

Een ander aspect is de integratie in toepassingen van delen van het .NET Framework als plug-in. Deze toepassingen zijn dan in staat .NET-code van internet te downloaden en direct uit te voeren.

Componenten van .NET Framework

De belangrijkste componenten van het .NET Framework zijn:

De Common Language Runtime, kortweg CLR

Dit is de omgeving die voor de uitvoering van toepassingen verantwoordelijk is. Steeds wanneer een .NET-toepassing wordt aangeroepen, neemt de CLR het over en zoekt de verschillende delen van de uit te voeren toepassing bij elkaar (de assemblies met de IL-code, maar ook optionele bronbestanden zoals beeld, geluid enzovoort) en laadt de inhoud in het werkgeheugen. De geïntegreerde JIT-compiler vertaalt de IL-code in machinecode en geeft deze voor verwerking door aan de processor.

Maar de CLR presteert meer. Terwijl de toepassing wordt uitgevoerd, is de CLR ook paraat om:

- onverwachte fouten af te vangen en aan de gebruiker te melden;
- beveiligingsvoorschriften in de code te handhaven, zoals beperkte toegang tot bestanden;
- geheugen voor het programma te reserveren en na gebruik automatisch weer vrij te geven.

Met andere woorden: de CLR voert de toepassing niet alleen uit, maar fungeert tegelijkertijd ook als bemiddelaar tussen de toepassing en het systeem en bewaakt daarbij de belangen van de toepassingen en waarborgt tevens de veiligheid en integriteit van het systeem.

De bibliotheken

Bijna nog belangrijker dan de CLR – waarbij u helemaal automatisch van de diensten profiteert – zijn de bibliotheken die deel uitmaken van het .NET Framework. U wilt in uw programma twee tekstpassages aan elkaar plakken, de sinus van 0.45 berekenen of een bestand op de vaste schijf openen? Geen probleem, de kant-en-klare IL-code voor zulke taken zit al in de .NET Framework-bibliotheken. Beschouw de .NET-bibliotheken voorlopig als een rijke verzameling programmamodulen waarvan u naar hartenlust gebruik mag maken in uw eigen toepassingen.

Objectgeoriënteerd

Voor wie al wat programmeerervaring heeft: de .NET-bibliotheken zijn objectgeoriënteerd en de functionaliteit is in klassen georganiseerd.

Belangrijke onderdelen .NET-bibliotheken

De zogenoemde Base Class Library, kortweg BCL is de basisbibliotheek met klassen voor in- en uitvoer, tekstverwerking, *threads* enzovoort. Verder zijn er klassen voor programmering van Windows-toepassingen en webdiensten en klassen voor het programmeren met bestanden, databases en XML.

Verderop in dit boek leert u hoe u bibliotheken kunt gebruiken en maakt u kennis met verschillende klassen uit de BCL en de bibliotheek voor programmering van Windows-toepassingen.

Framework, SDK en bronnen

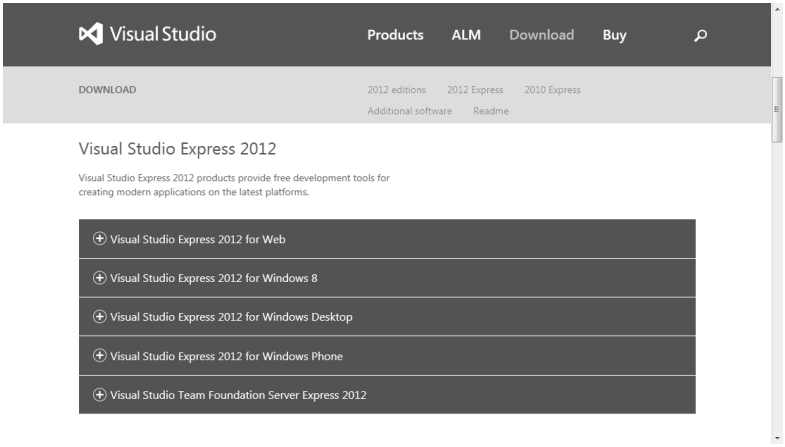
Programma's die u voor het .NET Framework hebt geschreven, draaien op elke computer waarop het .NET Framework is geïnstalleerd. In de huidige Windows-besturingssystemen is het .NET Framework stevig geïntegreerd. Ontbreekt .NET Framework, bijvoorbeeld omdat het is verwijderd of op oudere Windows-systemen, dan valt dit eenvoudig te installeren. U kunt .Net Framework downloaden op **http://msdn.microsoft.com/netframework/downloads** of installeer een softwarepakket dat op .NET is gebaseerd, zoals Visual Studio 2012. Ook toekomstige versies van Windows hebben het .NET Framework standaard aan boord, zodat praktisch elke Windows-gebruiker IL-programma's kan uitvoeren.

Acceptatie

Een belangrijk punt voor de acceptatie van C# is dat het .NET Framework – en daarmee de JIT-compiler – in de praktijk tot de standaarduitrusting van vrijwel iedere Windows-pc behoort en dus voor C#-toepassingen beschikbaar is. Programmeurs zullen immers niet graag programma's ontwikkelen waarvoor slechts weinig gebruikers zijn, omgekeerd zal een gebruiker ook niet snel investeren in een programma dat niet werkt op zijn computer.

Visual C#

Alles wat u nodig hebt om een C#-toepassing te schrijven is een teksteditor, zoals Kladblok (Notepad) of een tekstverwerker die zuivere, ongeformatteerde ASCII-tekst kan opslaan, een csc-compiler voor de vertaling en een Jitter om het programma te kunnen draaien. De teksteditor is een vast onderdeel van het besturingssysteem, de csc- en JIT-compiler zijn een onderdeel van het .NET Framework SDK en kunnen op de website van Microsoft gratis worden gedownload. Dus waarom zou u Visual C# gebruiken?



Afbeelding 1.3 *De Express Edition van Visual Studio is gratis en onbeperkt te gebruiken. Gewoon even downloaden bij Microsoft.*

Hoewel een eenvoudige teksteditor en de compilertools volstaan om programma's te ontwikkelen, wil dat niet zeggen dat het comfortabel werken is.

- Eerst stelt u in de editor de brontekst op.
- Dan roept u de C#-compiler csc aan die de brontekst in IL-code vertaalt en als programmabestand met de extensie .exe opslaat.
- Vervolgens kunt u het programma draaien en testen.
- Als alles goed gaat, bent u klaar. Zo niet – dan terug naar de editor, de fout zoeken en verbeteren, opnieuw compileren en testen. Bij hardnekkige fouten kunt u wel wat hulp gebruiken om de fouten te lokaliseren, zoals de debugger, maar dit is geen voorbeeld van gebruiksvriendelijkheid.

Het hele proces wordt een stuk aantrekkelijker met een handig gereedschap. En dat is het idee achter de geïntegreerde ontwikkelomgeving – IDE (Integrated Development Environment).

IDE

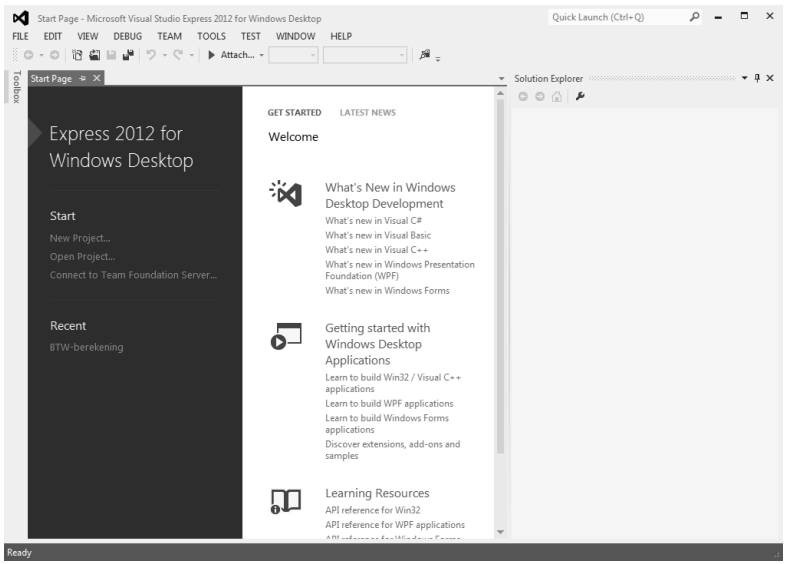
Een geïntegreerde ontwikkelomgeving heeft een gebruikersinterface van waaruit u alle voor de programmaontwikkeling benodigde stappen kunt uitvoeren:

- beheer van alle tot het project behorende bronteksten (projectbeheer);
- bewerking van de bronteksten (editor);
- vertaling van de bronteksten (compiler);
- uitvoeren en testen van het programma (JIT-compiler);
- foutzoeken (debugger).

En dat is precies wat Visual Studio u biedt. Als u Visual Studio start, verschijnt eerst de startpagina met onder het kopje Get started links naar nuttige informatie. Onder het kopje Latest news ziet u nieuws voor Visual Studio, hebt u een internetverbinding en klikt u op de RSS-feed, dan wordt het nieuws automatisch bijgewerkt.

Visual C# of Visual Studio?

Visual Studio is de geïntegreerde ontwikkelomgeving waarin u meestal uw Visual C#-programma's schrijft. In vorige versies van Visual Studio had u de mogelijkheid om slechts één programmeertaal te installeren. In de nieuwste versie krijgt u standaard de talen Visual C#, Visual Basic en C++ meegeleverd. U kiest bij het maken van een nieuw project welke programmeertaal u wilt gebruiken. In dit geval kiest u dan voor Visual C#.



Afbeelding 1.4 *Het startscherm van Visual Studio Express 2012.*

Opent u een project, dan ziet u het hoofdvenster van Visual Studio dat uit twee vensters bestaat met daarboven menu's en werkbalken. Het grootste linkervenster is het eigenlijke werkvenster, dat wordt gedeeld door de geïntegreerde editor en de formulierontwerper (Windows Forms Designer). Het rechtervenster is de projectverkenner, die voornamelijk dient voor de navigatie en het beheer van bestanden binnen het project. Bij de start is de projectverkenner nog leeg, omdat er geen project geopend is. De menu's bieden de mogelijkheid om uw bronbestanden te beheren, een editor te starten of een bestand te compileren.

Bepaalde programmeertools zijn een vast onderdeel van Visual Studio, zoals projectbeheer voor het organiseren van uw bronteksten of de editor die in vergelijking met een simpele teksteditor als Kladblok (Notepad) een hele lijst voordelen biedt, zoals syntaxis uitlijnen, automatisch aanvullen van code, automatisch inspringen en dergelijke. Andere programmeertools zijn externe programma's die vanuit de Visual Studio-IDE opgeroepen kunnen worden en vanuit het Visual Studio-venster zijn te bedienen. Hiertoe behoren de csc- en de JIT-compiler van het .NET Framework.

Verwar de IDE echter niet met een eenvoudig menusysteem voor het aanroepen van externe programma's. Zo'n menusysteem zou u in een uurtje zelf kunnen schrijven. De waarde en kracht van de IDE is juist de automatische informatie-uitwisseling tussen de verschillende elementen van de IDE. De twee volgende voorbeelden laten zien waarom.

- Als u met de C#-compiler csc werkt, moet u deze bij elke aanroep vertellen welke brontekst moet worden vertaald en in sommige gevallen moet u ook nog bepaalde compileropties meegeven voor de juiste vertaling. In Visual Studio regelt projectbeheer dit voor u. Wanneer u met behulp van Visual Studio een nieuw programma wilt ontwikkelen, legt u eerst een nieuw project aan. In dit project – beter gezegd, in het projectbestand dat Visual Studio intern aanlegt voor het project – legt Visual Studio vast welke brontekstbestanden tot het programma behoren en hoe deze gecompileerd moeten worden. Hebt u een Visual C#-project in Visual Studio geopend en de menuoptie voor het samenstellen van het project gekozen, dan roept Visual Studio op de achtergrond de C#-compiler op en geeft deze automatisch alle benodigde informatie voor het vertalen van de brontekstbestanden van het project. Wanneer de compiler waarschuwingen of zelfs foutmeldingen genereert, dan toont Visual Studio deze in het meldingenvenster.
- Een ander voorbeeld is de nauwe vervlechting tussen editor en debugger. Met de debugger kunt u een programma stap-voor-stap doorlopen, waarbij informatie over de toestand van het programma wordt getoond. Gebruikt u de debugger van Visual Studio, dan ziet u in de editor welke programma-

regel als volgende door de debugger wordt uitgevoerd. Verder kunt u in de editor aangeven op welke punten de debugger moet stoppen met het uitvoeren van de code – zogenoemde *breakpoints* – en kunt u in de editor de waarde op elk moment van de in het programma gebruikte variabelen bekijken. Meer over het werken met de debugger in hoofdstuk 9.

Express Edition

Visual Studio 2012 is in verschillende versies verkrijgbaar. Voor dit boek kunt u het beste Visual Studio Expres 2012 for Windows Desktop kiezen. Deze versie bevat drie programmeertalen, Visual Basic, Visual C# en Visual C++. De Express-editie van Visual Studio mag u gratis downloaden op de website van Microsoft (**www.microsoft.com/visualstudio/eng/downloads**). Er is dus eigenlijk geen enkele reden waarom u Visual Studio niet zou gebruiken.

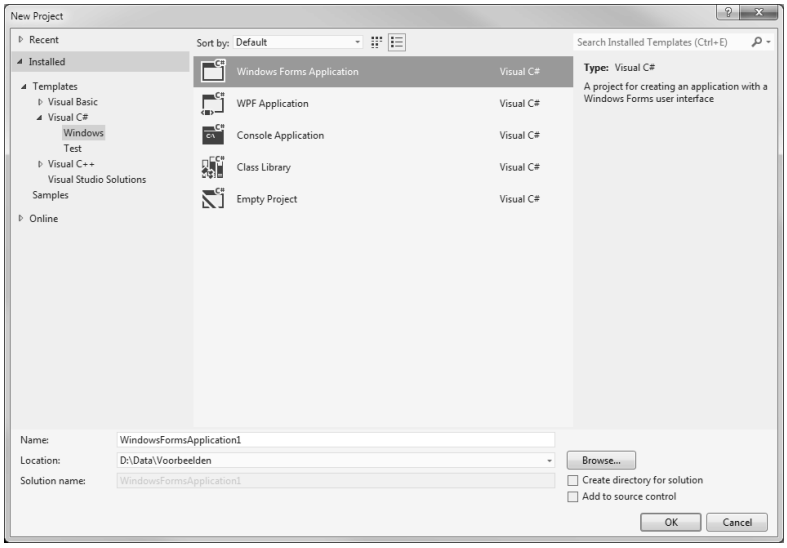
Visual C#

Visual C# was in vorige versies een versie van Visual Studio met uitsluitend de taal C#. Visual Studio 2012 kent dat onderscheid niet meer. Bij het maken van een nieuw project kiest u voor Visual C# en dan een template voor het nieuwe project. De term Visual C# duidt hierna steeds een Visual C#-project aan in de ontwikkelomgeving Visual Studio.

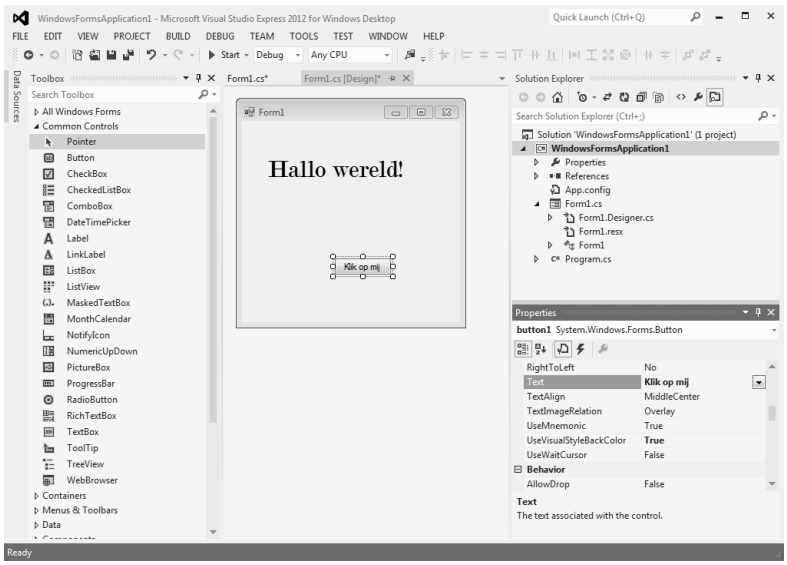
RAD

Visual C# is niet alleen een geïntegreerde ontwikkelomgeving, maar het is ook een hoogwaardig RAD-tool. RAD staat voor *Rapid Application Development* – snelle toepassingsontwikkeling – en gebruikt in het geval van Visual C# de op componenten gebaseerde programmering en de grafische gebruikersinterface.

Beginnen we met de grafische gebruikersinterface. Een groot deel van de programma's die vandaag de dag worden ontwikkeld, is bedoeld om te draaien onder speciale Windows-systemen (Microsoft Windows, X Window voor Linux) en om de gebruiker een foutloze grafische interface te bieden. Zo'n grafische gebruikersinterface bestaat in de regel uit een of meer vensters die beschikken over verschillende onderdelen, zoals menu's, knoppen, invoervelden enzovoort. De weergave van zo'n venster met zijn verschillende onderdelen is veel makkelijker en sneller te maken met een grafische editor die het venster laat zien zoals het er voor de gebruiker zal uitzien – WYSIWYG, *what you see is what you get* – dan door het typen van de bijbehorende C#-brontekst in een teksteditor. Daarom heeft Visual C# een speciale ontwerpmodus voor de grafische opbouw van vensters.



Afbeelding 1.5 *Start een nieuw project in Visual C# en kies een sjabloon (template) voor uw project. Gebruik Windows Forms Application om een Windows-toepassing te maken met vensters.*



Afbeelding 1.6 *WYSIWYG: een venster bewerken met de Designer in VisualC#.*

Tot de RAD-omgeving van Visual C# behoren onder andere:

- de *Windows Forms Designer* (kortweg *Designer*) voor de grafische opbouw van vensters;
- de *Toolbox* met beschikbare componenten;
- het *eigenschappenvenster* voor de configuratie van de toe te passen componenten.

Componenten

Bepaalde taken komen bij het programmeren steeds weer terug. Geen wonder dus, dat een hele reeks initiatieven ontstond om eenmaal ontwikkelde code tijd- en geldbesparend opnieuw te kunnen gebruiken. Een van de meest succesvolle initiatieven is de ontwikkeling van componenten. Deze componenten zijn min of meer complexe programmabouwstenen die eenvoudig opnieuw zijn te gebruiken , dat wil zeggen: in nieuwe programma's kunnen worden ingebouwd. Met behulp van componenten is het mogelijk om snel een programma samen te stellen, vergelijkbaar met het bouwen van een huis met legostenen.

Niet alleen C#

Componenten worden niet alleen in C# of Visual C# gebruikt, maar ook in andere ontwikkelomgevingen, zoals Visual Basic, Delphi of C++.

Visual C# heeft al een aantal componenten aan boord die u vrijelijk mag gebruiken. Het gelijkvormige constructieschema van componenten maakt de integratie van componenten in het RAD-systeem van Visual C# mogelijk. Voor programmeurs wil dat zeggen dat componenten eenzelfde opbouw hebben en eenvoudig zijn in te bouwen onder het motto: ken je er één, dan ken je ze allemaal. In het volgende hoofdstuk maakt u daar kennis mee.

Verschil moet er zijn

Hoewel componenten het leven van een programmeur makkelijk kunnen maken, wil dat niet zeggen dat het schrijven van componenten dat ook is. In tegendeel, het ontwikkelen van een component kan bijzonder complex zijn, terwijl het gebruik ervan meestal zeer eenvoudig is.