

# A C# 2008 és a .NET 3.5

Első kötet (1-19. fejezet)

Andrew Troelsen



2009

A C# 2008 és a .NET 3.5 – Első kötet (1-19. fejezet)  
A negyedik kiadás magyarul, két kötetben  
Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition  
Andrew Troelsen

Copyright 2008 by Apress.  
Original English language edition published by Apress. Co Sound View Court 3B,  
Greenwich CT 06830 USA. Copyright © 2008 by Apress Co.. Hungarian language  
edition copyright © 2009 by SZAK Kiadó. All rights reserved.

Magyar fordítás (Hungarian translation) © SZAK Kiadó 2009.  
Fordította a SZAK Kiadó fordítócsoportja: Domoszlai László, Erős Melinda, Erős  
Szilvia, Iváncsy Renáta, Iváncsy Tamás, Lucza Mónika Tiber Melinda, Varga Péter  
Lektor: Dávid Zoltán, MVP és Gincsei Gábor, MVP  
Magyar nyelvi gondozás: Laczkó Krisztina

ISBN 978-963-9863-08-8

A könyv fordítása a Kilgray Kft. MemoQ (<http://www.memoqtm.com>) programjával készült, a szöveg helyességét és az elválasztásokat a MorphoLogic He-lyesek nevű programjával ellenőriztük.

Minden jog fenntartva. Jelen könyvet, illetve annak részeit a kiadó engedélye nélkül tilos reprodukálni, adatrendszerben tárolni, bármilyen formában vagy eszközzel elektronikusan úton vagy más módon közölni.



SZAK Kiadó Kft. ■ Az 1795-ben alapított Magyar Könyvkiadók és Könyvterjesztők Egyesülésének a tagja ■ 2060 Bicske, Diófa u. 3. ■ Tel.: 36-22-350-209 ■ Fax: 36-22-565-311 ■ [www.szak.hu](http://www.szak.hu) ■ e-mail: [info@szak.hu](mailto:info@szak.hu) ■ Kiadóvezető: Kis Ádám, e-mail: [adam.kis@szak.hu](mailto:adam.kis@szak.hu) ■ Főszerkesztő: Kis Balázs MCSE, MCT, e-mail: [balazs.kis@szak.hu](mailto:balazs.kis@szak.hu)



Ezt a kiadást Milkónak, a csodamacsának, a 412-es szám alatt elítöltött  
éveknek és csodálatos feleségemnek, Amandának ajánlom, aki türelmesen  
várta, hogy elkészüljek újabb könyvem megírásával.



# Tartalomjegyzék

|                                                                              |         |
|------------------------------------------------------------------------------|---------|
| Köszönetnyilvánítás.....                                                     | xxix    |
| Bevezetés.....                                                               | xxxi    |
| Mi egy csapat vagyunk, az Olvasó és én.....                                  | xxxii   |
| A könyv áttekintése.....                                                     | xxxiii  |
| Első kötet.....                                                              | xxxiii  |
| 1. rész: Bevezetés a C#-ba és a .NET platformba.....                         | xxxiii  |
| 1. fejezet: A .NET filozófiája.....                                          | xxxiii  |
| 2. fejezet: C#-alkalmazások készítése.....                                   | xxxiv   |
| 2. rész: A C# alapvető építőelemei.....                                      | xxxiv   |
| 3. fejezet: A C# alapvető építőelemei, I. rész.....                          | xxxiv   |
| 4. fejezet: A C# alapvető építőelemei, II. rész.....                         | xxxiv   |
| 5. fejezet: Egységbe zárt osztálytípusok definiálása.....                    | xxxv    |
| 6. fejezet: A származtatás és a polimorfizmus.....                           | xxxv    |
| 7. fejezet: Strukturált hibakezelés.....                                     | xxxv    |
| 8. fejezet: Az objektumok életciklusa.....                                   | xxxv    |
| 3. rész: Haladó programozási szerkezetek a C#-ban.....                       | xxxvi   |
| 9. fejezet: Interfészek használata.....                                      | xxxvi   |
| 10. fejezet: Gyűjtemények és generikus típusok.....                          | xxxvi   |
| 11. fejezet: Metodusreferenciák, események és lambdák.....                   | xxxvi   |
| 12. fejezet: Indexerek, operátorok és mutatók.....                           | xxxvii  |
| 13. fejezet: C# 2008 nyelvi újonságai.....                                   | xxxvii  |
| 14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (LINQ).....         | xxxvii  |
| 4. rész: Programozás .NET-szerelvényekkel.....                               | xxxvii  |
| 15. fejezet: A .NET-szerelvények.....                                        | xxxviii |
| 16. fejezet: Típusreflexió, késői kötés és attrribútumalapú programozás..... | xxxviii |
| 17. fejezet: Folyamatok, alkalmazástartományok és objektumkönyvezetek.....   | xxxviii |
| 18. fejezet: Többszálú alkalmazások készítése.....                           | xxxviii |
| 19. fejezet: A köztes nyelv (CIL) és a dinamikus szerelvények.....           | xxxix   |

|                                                                    |        |
|--------------------------------------------------------------------|--------|
| Második kötet.....                                                 | xxxxix |
| 5. rész: Bevezetés a .NET alaposztálykönyvtárába .....             | xxxxix |
| 20. fejezet: Fájlműveletek és elkülönített tárolás .....           | xxxix  |
| 21. fejezet: Bevezetés az objektumsorostás világába .....          | xl     |
| 22. fejezet: ADO.NET 1. rész: Az elő kapcsolat .....               | xl     |
| 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat .....            | xl     |
| 24. fejezet: A LINQ API programozása .....                         | xl     |
| 25. fejezet: A WCF .....                                           | xli    |
| 26. fejezet: A WF .....                                            | xli    |
| 6. rész: Felhasználói felületek.....                               | xli    |
| 27. fejezet: Windows Forms-programozás.....                        | xli    |
| 28. fejezet: A WPF és az XAML .....                                | xlii   |
| 29. fejezet: Programozás WPF-vezérlőelemekkel.....                 | xlii   |
| 30. fejezet: WPF 2D grafikus renderelés, erőforrások és témák..... | xlii   |
| 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével.....  | xliii  |
| 31. fejezet: ASP.NET weboldalak készítése .....                    | xliii  |
| 32. fejezet: ASP.NET-vezérlőelemek, témák és mesteroldalak .....   | xliii  |
| 33. fejezet: ASP.NET állapotkezelési technikák.....                | xliii  |
| 8. rész: Függelék.....                                             | xliiv  |
| A függelék: A COM és a .NET együttműködése .....                   | xliiv  |
| B függelék: Platformfüggetlen .NET-fejlesztés a Mondóval.....      | xliiv  |
| Ct szabadon letölthető fejezet — még több információ .....         | xliiv  |
| A könyv forráskódjainak igénylése.....                             | xliv   |
| A lehetséges javítások .....                                       | xlivi  |
| Elérhetőség.....                                                   | xlivi  |
| 1. rész: Bevezetés a C#-ba és a .NET platformba                    |        |
| 1. A .NET filozófiája.....                                         | 3      |
| Az előzmények áttekintése.....                                     | 4      |
| Egy C/Win32 API programozó élete .....                             | 4      |
| Egy C++/MFC-programozó élete .....                                 | 4      |
| Egy Visual Basic 6.0 programozó élete .....                        | 5      |
| Egy Java/J2EE-programozó élete .....                               | 5      |
| Egy COM-fejlesztő élete.....                                       | 6      |
| Egy Windows DNA programozó élete.....                              | 7      |

|                                                                     |    |
|---------------------------------------------------------------------|----|
| A .NET-megoldás .....                                               | 8  |
| Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS) .....     | 9  |
| Az alaposztálykönyvtárak szerepe .....                              | 10 |
| A C# előnyei .....                                                  | 11 |
| További .NET-alapú programozási nyelvek .....                       | 13 |
| Élet egy többnyelvű világban .....                                  | 14 |
| A .NET-szerelvények áttekintése .....                               | 15 |
| Egyfajlos és többfajlos szerelvények .....                          | 17 |
| A közös köztes nyelv (CIL) szerepe .....                            | 18 |
| A közös nyelv előnyei .....                                         | 20 |
| A közös nyelv fordítása platformfüggetlen utasítássá .....          | 21 |
| A .NET-típusmetaadat szerepe .....                                  | 22 |
| A szerelvény metaadatoknak szerepe .....                            | 23 |
| A közös típusrendszer (CTS) .....                                   | 23 |
| Az egysegű típusrendszer osztálytípusai .....                       | 24 |
| Az egysegű típusrendszer interfész típusai .....                    | 25 |
| Az egysegű típusrendszer struktúrátípusai .....                     | 25 |
| Az egysegű típusrendszer felsorolt típusai .....                    | 26 |
| Az egysegű típusrendszer metódusreferencia-típusai .....            | 27 |
| Az egysegű típusrendszer típusragjai .....                          | 27 |
| Valódi CTS-adattípusok .....                                        | 28 |
| A közös nyelvi specifikáció (CLS) .....                             | 29 |
| A CLS-megfeleltetés biztosítása .....                               | 31 |
| A közös nyelvi futtatórendszer (CLR) .....                          | 32 |
| A szerelvény, a névter és a típus megkülönböztetése .....           | 33 |
| A Microsoft-névterek szerepe .....                                  | 37 |
| Programozott hozzáférés a névterekhez .....                         | 38 |
| Hivatkozás külső szerelvényekre .....                               | 39 |
| Szerelvények feltárása az ildasm.exe segítségével .....             | 40 |
| A köztes nyelvi kód megtekintése .....                              | 41 |
| A típusmetaadat megtekintése .....                                  | 42 |
| Szerelvény metaadatainak (manifesztumának) megtekintése .....       | 42 |
| Szerelvények feltárása a Lutz Roeder's Reflector segítségével ..... | 43 |
| A .NET-futtatókörnyezet telepítése .....                            | 44 |
| A .NET platformfüggetlen természete .....                           | 44 |
| Összefoglalás .....                                                 | 46 |

|                                                                                   |           |
|-----------------------------------------------------------------------------------|-----------|
| <b>2. C#-alkalmazások fordítása .....</b>                                         | <b>49</b> |
| A .NET 3.5 keretrendszer fejlesztőkészletének szerepe .....                       | 49        |
| A Visual Studio 2008 parancssor .....                                             | 50        |
| C#-alkalmazások fordítása csc.exe használatával .....                             | 51        |
| Bemeneti és kimeneti célok megadása .....                                         | 52        |
| Hivatkozás külső szereplvényekre .....                                            | 54        |
| Hivatkozás több külső szereplvényre .....                                         | 55        |
| Több forrásfájl fordítása .....                                                   | 55        |
| A C#-reakciófájlok használata .....                                               | 56        |
| .NET-alkalmazás készítése TextPad használatával .....                             | 59        |
| A C#-kulcsszószínezés bekapcsolása .....                                          | 59        |
| A *.cs fájlszűrő konfigurálása .....                                              | 60        |
| A csc.exe integrálása .....                                                       | 61        |
| Futtatási parancsok társítása menüelemekhez .....                                 | 62        |
| .NET-alkalmazás készítése Notepad++ használatával .....                           | 64        |
| Az automatikus kiegészítési lista testreszabása .....                             | 65        |
| .NET-alkalmazás készítése SharpDevelop használatával .....                        | 66        |
| Egy egyszerű tesztprojekttel létrehozása .....                                    | 67        |
| .NET-alkalmazás készítése Visual C# 2008 Express használatával .....              | 69        |
| A Visual C# Express néhány jellegetessége .....                                   | 70        |
| .NET-alkalmazás készítése Visual Studio 2008 használatával .....                  | 71        |
| A Visual Studio 2008 néhány jellegetessége .....                                  | 72        |
| A .NET-keretrendszer kiválasztása a New Project párbeszédablak segítségével ..... | 72        |
| A Solution Explorer használata .....                                              | 73        |
| Hivatkozás külső szereplvényekre .....                                            | 74        |
| Projektitulajdonosságok megtekintése .....                                        | 74        |
| A Class View .....                                                                | 75        |
| Az Object Browser .....                                                           | 76        |
| Integrált támogatás a kódújratervezéshez .....                                    | 76        |
| Kódbovítések és a környezetbe foglalt technológiája .....                         | 79        |
| A Visual Class Designer .....                                                     | 81        |
| Object Test Bench .....                                                           | 84        |
| Az integrált .NET Framework 3.5 dokumentációs rendszer .....                      | 85        |
| További .NET-fejlesztőeszközök .....                                              | 87        |
| Összefoglalás .....                                                               | 88        |

## 2. rész: A C# alapvető építőelemei

### 3. A C# alapvető építőelemei, I. rész ..... 91

|                                                                        |     |
|------------------------------------------------------------------------|-----|
| Egy egyszerű C#-program anatómiája.....                                | 91  |
| Variációk Main() metódusra.....                                        | 93  |
| Alkalmazás-hibakód megadása.....                                       | 94  |
| A parancssori argumentumok feldolgozása.....                           | 96  |
| Parancssori paraméterek megadása Visual Studio 2008-ban.....           | 98  |
| Érdekes kitérő: a System.Environment osztály néhány további tagja..... | 99  |
| A System.Console osztály.....                                          | 100 |
| A Console osztály alapvető bemenete és kimenete.....                   | 101 |
| A konzolkimenet formázása.....                                         | 102 |
| Numerikus adatok formázása.....                                        | 103 |
| Numerikus adatok formázása a parancssori alkalmazásokon túl.....       | 105 |
| A System-adattípusok és gyorsazonosítók a C#-ban.....                  | 106 |
| Valtózódeklarálás és inicializálás.....                                | 108 |
| A beépített adattípusok példányosítása.....                            | 110 |
| Az adattípusok hierarchiája.....                                       | 110 |
| A numerikus adattípusok tagjai.....                                    | 112 |
| A System.Boolean tagjai.....                                           | 113 |
| A System.Char tagjai.....                                              | 114 |
| Értékek értelmezése sztringadatokból.....                              | 114 |
| A System.String típus.....                                             | 115 |
| Egyszerű sztringkezelés.....                                           | 116 |
| Sztringösszeftűzés.....                                                | 117 |
| Vezérlőkéarakterek.....                                                | 118 |
| Szó szerinti sztringek definiálása.....                                | 119 |
| A sztringek és az egyenlőség.....                                      | 120 |
| A sztring megvalósíthatatlan.....                                      | 121 |
| A System.Text.StringBuilder típus.....                                 | 123 |
| System.DateTime és System.TimeSpan.....                                | 125 |
| Szűkítő és bővítő adattípus-konverziók.....                            | 126 |
| Hibacsapda állítása a szűkítő adatkonverzióknak.....                   | 128 |
| Projektszintű túlcsoordulás-ellenőrzés beállítása.....                 | 131 |
| Az unchecked kulcsszó.....                                             | 132 |
| A System.Convert szerepe.....                                          | 132 |

#### 4. A C# alapvető építőelemei, II. rész..... 141

|                                                             |     |
|-------------------------------------------------------------|-----|
| Ciklus készítése a C#-ban.....                              | 133 |
| A for ciklus.....                                           | 133 |
| A foreach ciklus.....                                       | 134 |
| A while és a do/while ciklusszerkezetek.....                | 135 |
| Elágazó szerkezetek és a reláció/egyenlőség-operátorok..... | 136 |
| Az if/else utasítás.....                                    | 136 |
| A switch utasítás.....                                      | 138 |
| Osszefoglalás.....                                          | 139 |

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| Metódusok és paramétermódosítók.....                            | 141 |
| Az alapértelmezett paraméterátadási viselkedés.....             | 142 |
| Az out módosító.....                                            | 144 |
| A ref módosító.....                                             | 145 |
| A params módosító.....                                          | 147 |
| A tag túlterhelésének megértése.....                            | 149 |
| Tömbök kezelése a C#-ban.....                                   | 151 |
| A C# tömbinicializáló szintaxisa.....                           | 152 |
| Objektumtömb definíciója.....                                   | 153 |
| Munka a többdimenziós tömbökkel.....                            | 154 |
| Tömbök mint paraméterek (és visszatérési értékek).....          | 156 |
| A System.Array összetétel.....                                  | 156 |
| Az Enum típus.....                                              | 159 |
| Az enum alapjául szolgáló tároló vezérlése.....                 | 160 |
| Felsorolások deklarálása és használata.....                     | 160 |
| A System.Enum típus.....                                        | 162 |
| Felsorolás név/érték pártjának dinamikus feltárása.....         | 162 |
| A struktúrátípus.....                                           | 165 |
| Struktúrávaltozók létrehozása.....                              | 167 |
| Érték- és referenciátípusok.....                                | 169 |
| Érték- és referenciátípusok, valamint az értékadó operátor..... | 170 |
| Referenciátípusokat tartalmazó értéktípusok.....                | 172 |
| Referenciátípusok átadása értékként.....                        | 174 |
| Referenciátípusok átadása referenciaként.....                   | 176 |
| Érték- és referenciátípusok: további részletek.....             | 177 |
| Nullázható típusok a C#-ban.....                                | 179 |
| Munka a nullázható típusokkal.....                              | 180 |
| A ?? operátor.....                                              | 181 |
| Osszefoglalás.....                                              | 182 |



|                                                                           |            |
|---------------------------------------------------------------------------|------------|
| <b>5. Beágyazott osztálytípusok definiálása .....</b>                     | <b>183</b> |
| Bevezetés a C# osztálytípusaiba.....                                      | 183        |
| Objektumok memóriaterületének lefoglalása                                 |            |
| a new kulcsszóval .....                                                   | 186        |
| A konstruktor .....                                                       | 187        |
| Az alapértelmezett konstruktor szerepe.....                               | 188        |
| Egyedi konstruktor definiálása .....                                      | 189        |
| Az újradefiniált alapértelmezett konstruktor .....                        | 190        |
| A this kulcsszó szerepe .....                                             | 191        |
| Konstruktorhívások láncolása a this kulcsszóval .....                     | 193        |
| A konstruktorok lefutásának sorrendje .....                               | 196        |
| A static kulcsszó .....                                                   | 198        |
| Statikus metódusok (és mezők) definiálása .....                           | 199        |
| Statikus adatok definiálása .....                                         | 200        |
| Statikus konstruktorok definiálása .....                                  | 203        |
| Statikus osztályok definiálása .....                                      | 205        |
| Az objektumorientált programozás alappillérei .....                       | 207        |
| Az egységbe záras .....                                                   | 208        |
| A származtatás .....                                                      | 209        |
| A polimorfizmus .....                                                     | 210        |
| Hozzáférés-módosítók a C#-ban .....                                       | 212        |
| Az alapértelmezett hozzáférés-módosítók .....                             | 213        |
| Hozzáférés-módosítók és beágyazott típusok .....                          | 214        |
| Az első pillér: az egységbe záras a C#-ban .....                          | 215        |
| Egységbe záras tradicionális lekérdezők és módosítók használataival ..... | 216        |
| Egységbe záras típusutalajdonságok használataival .....                   | 218        |
| A tulajdonságok belső reprezentációja .....                               | 220        |
| A get/set utasítástulajdonságok láthatósági szintjeinek vezérlése .....   | 222        |
| Csak olvasható és csak írható tulajdonságok .....                         | 223        |
| Statikus tulajdonságok .....                                              | 224        |
| Konstans adatok .....                                                     | 226        |
| Írásvédett mezők .....                                                    | 227        |
| Statikus írásvédett mezők .....                                           | 228        |
| Részleges típusok .....                                                   | 229        |

|                                                                      |     |
|----------------------------------------------------------------------|-----|
| A C#-forráskód dokumentálása XML segítségével.....                   | 231 |
| Az XML-fájl generálása.....                                          | 234 |
| XML-megjegyzések átalakítása az NDoc-on keresztül.....               | 235 |
| Az elvégzett munka eredményének láthatóvá tétele.....                | 237 |
| Összefoglalás.....                                                   | 238 |
| <b>6. A származtatás és a polimorfizmus..... 239</b>                 |     |
| A származtatás alapvető szerkezete.....                              | 239 |
| Az osztálytípus szülőosztálya.....                                   | 240 |
| Több osztály.....                                                    | 242 |
| A sealed kulcsszó.....                                               | 243 |
| Osztálydiagramok készítése a Visual Studio 2008-ban.....             | 245 |
| A második pillér: a származtatás.....                                | 246 |
| Osztály konstruktorának meghívása a base kulcsszóval.....            | 249 |
| Családi titok megtartása: a protected kulcsszó.....                  | 251 |
| Lezárt osztály hozzáadása.....                                       | 252 |
| A tartalmazás és a deleghálás.....                                   | 254 |
| A beágyazott típus definiálása.....                                  | 255 |
| A harmadik pillér: a polimorfizmus támogatása.....                   | 258 |
| A virtual és override kulcsszavak.....                               | 259 |
| Virtuális tagok felülbírálása a Visual Studio 2008 segítségével..... | 261 |
| A virtuális tagok lezárása.....                                      | 262 |
| Az absztrakt osztályok.....                                          | 263 |
| Polimorfikus interfész építése.....                                  | 265 |
| Tagok eltávolítása.....                                              | 270 |
| Az osztály, a lezárazott osztály és a típusmódosítási szabályok..... | 272 |
| Az as kulcsszó.....                                                  | 274 |
| Az is kulcsszó.....                                                  | 274 |
| A fő szülőosztály: a System.Object.....                              | 275 |
| A System.Object.ToString() felüldefiniálása.....                     | 279 |
| A System.Object.Equals() felüldefiniálása.....                       | 280 |
| A System.Object.GetHashCode() felüldefiniálása.....                  | 281 |
| A módosított Person osztály tesztelése.....                          | 282 |
| A System.Object statikus tagjai.....                                 | 283 |
| Összefoglalás.....                                                   | 284 |

|                                                                       |            |
|-----------------------------------------------------------------------|------------|
| <b>7. A strukturált kivételkezelés .....</b>                          | <b>285</b> |
| Hibák, programhibák és kivételek .....                                | 285        |
| A kivételkezelés szerepe a .NET-ben .....                             | 286        |
| A .NET-kivételkezelés .....                                           | 288        |
| A System.Exception osztály .....                                      | 288        |
| A lehető legegyszerűbb példa .....                                    | 290        |
| Altalanos kivétel dobása .....                                        | 292        |
| Kivételek elkapása .....                                              | 294        |
| A kivétel tulajdonságainak beállítása .....                           | 295        |
| A TargetSite tulajdonság .....                                        | 295        |
| A StackTrace tulajdonság .....                                        | 297        |
| A HelpLink tulajdonság .....                                          | 297        |
| A Data tulajdonság .....                                              | 298        |
| Rendszerszintű kivételek (System.SystemException) .....               | 301        |
| Alkalmazásszintű kivételek (System.ApplicationException) .....        | 302        |
| Egyedi kivétel készítése, 1. rész .....                               | 302        |
| Egyedi kivétel készítése, 2. rész .....                               | 305        |
| Egyedi kivétel készítése, 3. rész .....                               | 306        |
| Többszörös kivételek feldolgozása .....                               | 308        |
| Altalanos catch utasítások .....                                      | 310        |
| Kivételek továbbdobása .....                                          | 311        |
| Belső kivételek .....                                                 | 311        |
| A finally blokk .....                                                 | 313        |
| Ki mit dob? .....                                                     | 314        |
| A kezeletlen kivételek eredménye .....                                | 315        |
| Kezeletlen kivételek hibakeresése a Visual Studio használatával ..... | 315        |
| Összefoglalás .....                                                   | 317        |
| <b>8. Az objektumok élettartama.....</b>                              | <b>319</b> |
| Osztályok, objektumok és referenciák .....                            | 319        |
| Az objektumok élettartama .....                                       | 321        |
| A new kulcsszó köztes nyelve .....                                    | 322        |
| Objektumreferenciák null értékre állítása .....                       | 323        |
| Az alkalmazás gyökerelemének szerepe .....                            | 325        |
| Az objektumgenerációk .....                                           | 327        |
| A System.GC típus .....                                               | 328        |
| A személggyűjtés kikényszerítése .....                                | 330        |

### 3. rész: Haladó programozási szerkezetek a C#-ban

## 9. Interfészek használata..... 349

|                                                                   |     |
|-------------------------------------------------------------------|-----|
| Az Interface típus .....                                          | 349 |
| Az interfész típusok és absztrakt osztályok .....                 | 351 |
| Összehasonlítása .....                                            | 354 |
| Saját interfészek definiálása .....                               | 356 |
| Az interfész implementálása .....                                 | 358 |
| Interfésztagek hívása az objektumon .....                         | 360 |
| Az interfészreferenciák megszerzése: az as kulcsszó .....         | 360 |
| Az interfészreferenciák megszerzése: az is kulcsszó .....         | 361 |
| Interfészek mint paraméterek .....                                | 364 |
| Interfészek mint visszatérési értékek .....                       | 364 |
| Interfésztípusok tömbje .....                                     | 366 |
| Interfészimplementálás a Visual Studio 2008 segítségével .....    | 367 |
| Névutközés feloldása explicit interfészimplementációval .....     | 371 |
| Interfészhierarchia tervezése .....                               | 372 |
| Többszörös örökletes interfésztípusokkal .....                    | 375 |
| Felsorolható típusok készítése (IEnumerable és IEnumerator) ..... | 379 |
| Iterátor metódus készítése a yield kulcsszóval .....              | 380 |
| Nevesített iterátor létrehozása .....                             | 381 |
| Az iterátormetódusok belső reprezentációja .....                  | 383 |
| Klónozható objektumok készítése (ICloneable) .....                | 385 |
| Egy összetettebb klónozási példa .....                            | 389 |
| Összehasonlítható objektumok készítése (IComparable) .....        | 392 |
| Többszörös rendezési sorrend megadása (IComparable) .....         | 394 |
| Egyedi tulajdonságok, egyedi rendezési típusok .....              | 395 |
| A Callback interfész .....                                        | 399 |
| Összefoglalás .....                                               | 399 |

**10. Gyűjtemények és generikus típusok ..... 401**

|                                                                |     |
|----------------------------------------------------------------|-----|
| A System.Collections névtér interfészei.....                   | 401 |
| Az ICollection interfész.....                                  | 403 |
| Az IDictionary interfész.....                                  | 403 |
| Az IDictionaryEnumerator interfész.....                        | 404 |
| Az IList interfész.....                                        | 404 |
| A System.Collections névtér osztálytípusai.....                | 405 |
| Az ArrayList típus használata.....                             | 406 |
| A Queue típus használata.....                                  | 408 |
| A Stack típus használata.....                                  | 409 |
| A System.Collections.Specialized névtér.....                   | 410 |
| A bedobozolás, kidobozolás és a System.Object kapcsolata.....  | 412 |
| A dobozolásról járó probléma.....                              | 415 |
| A típusbiztonság és az erősen típusos gyűjtemények.....        | 416 |
| Egyedi gyűjtemény készítése.....                               | 417 |
| Az erősen típusos gyűjtemények és a dobozolás.....             | 419 |
| A System.Collections.Generic névtér.....                       | 421 |
| A List<T> típus vizsgálata.....                                | 423 |
| Saját generikus metódusok létrehozása.....                     | 426 |
| A típusparaméterek kikövetkeztetése.....                       | 427 |
| Generikus struktúrák és osztályok létrehozása.....             | 429 |
| A default kulcsszó a generikus kódban.....                     | 430 |
| Saját generikus gyűjtemény készítése.....                      | 432 |
| A saját generikus gyűjtemény korlátai.....                     | 433 |
| Típusparaméter korlátozása a where kulcsszó használatával..... | 435 |
| Az operátorkorlátozások hiánya.....                            | 438 |
| Generikus ösztály létrehozása.....                             | 439 |
| Generikus interfész létrehozása.....                           | 440 |
| Összefoglalás.....                                             | 442 |

**11. Metódusreferenciák, események és lambdák ..... 443**

|                                                                  |     |
|------------------------------------------------------------------|-----|
| A.NET-metódusreferencia.....                                     | 443 |
| A metódusreferencia definiálása C#-ban.....                      | 445 |
| A System.MulticastDelegate és System.Delegate alaposztályok..... | 448 |
| A legegyszerűbb metódusreferencia.....                           | 449 |
| A metódusreferencia-objektum vizsgálata.....                     | 452 |

|                                                                        |            |
|------------------------------------------------------------------------|------------|
| A Car típus utólagos módosítása metódusreferenciákkal .....            | 453        |
| Többes küldés engedélyezése .....                                      | 457        |
| Cél eltávolítása a metódusreferencia hívási listájáról .....           | 459        |
| Egy összetettebb metódusreferencia .....                               | 460        |
| Metódusreferenciák mint paraméterek .....                              | 461        |
| Az átírányító kód elemzése .....                                       | 464        |
| Metódusreferenciák kovarianciája .....                                 | 465        |
| Generikus metódusreferencia készítése .....                            | 468        |
| A generikus metódusreferencia szimulációja .....                       | 469        |
| C#-események .....                                                     | 471        |
| Az event kulcsszó .....                                                | 472        |
| Eseménykezelés a felszín alatt .....                                   | 474        |
| A bejövő események figyelése .....                                     | 476        |
| Esemény beregisztrálásának egyszerűsítése .....                        | 477        |
| a Visual Studio 2008 használatával .....                               | 478        |
| Egy „korrekt” esemény .....                                            | 478        |
| A generikus EventHandler<T> metódusreferencia .....                    | 480        |
| Névtelen metódusok a C#-ban .....                                      | 481        |
| „Külső” változók elérése .....                                         | 483        |
| A metóduscsoport átalakítása .....                                     | 484        |
| A C# 2008 lambda operátora .....                                       | 486        |
| A lambda kifejezések részletezése .....                                | 489        |
| Argumentumok feldolgozása több utasításon keresztül .....              | 490        |
| A CarDelegate példa áttekintése lambda kifejezések használatával ..... | 492        |
| lambda kifejezések több paraméterrel (vagy paraméterek nélkül) .....   | 494        |
| Összefoglalás .....                                                    | 496        |
| <b>12. Indexelők, operátorok és mutatók .....</b>                      | <b>497</b> |
| Az indexelő metódus .....                                              | 497        |
| Objektumok indexelése sztringértékekkel .....                          | 500        |
| Az indexelő metódusok túlterhelése .....                               | 501        |
| Az indexelő metódusok belső ábrázolása .....                           | 502        |
| Többdimenziós indexelők .....                                          | 503        |
| Indexelő definíciók interfésztípusokon .....                           | 503        |

|                                                            |     |
|------------------------------------------------------------|-----|
| Az operátor-tülterheles.....                               | 504 |
| Kétooperandusú operátorok tülterhelese.....                | 506 |
| Mi a helyzet a += és -= operátorokkal?.....                | 508 |
| Egyoperandusú operátorok tülterhelese.....                 | 509 |
| Az egyenlőségvizsgáló operátorok tülterhelese.....         | 510 |
| Összehasonlító operátorok tülterhelese.....                | 511 |
| A tülterhelt operátorok belső ábrázolása.....              | 512 |
| Zárzó az operátor-tülterheléstől.....                      | 514 |
| Egyedi típuskonverziók.....                                | 515 |
| A numerikus konverzió.....                                 | 515 |
| Konverzió az egymásból származó osztálytípusok között..... | 516 |
| Egyedi konverziós rutinok készítése.....                   | 517 |
| A Square típus további explicit konverziói.....            | 520 |
| Implicit konverziós rutinok definíciója.....               | 521 |
| Az egyedi konverziós rutinok belső ábrázolása.....         | 523 |
| Mutatótípusok használata.....                              | 524 |
| Az unsafe kulcsszó.....                                    | 526 |
| A * és az & operátor.....                                  | 528 |
| A nem felügyelt (és felügyelt) cseréltűgvény.....          | 528 |
| Mezőleíres mutatókkal (a -> operátor).....                 | 530 |
| A stackalloc kulcsszó.....                                 | 530 |
| Típuspeldány rögzítése a fixed kulcsszóval.....            | 531 |
| A sizeof kulcsszó.....                                     | 532 |
| A C# előfordítói direktívái.....                           | 533 |
| Kódregiók megadása.....                                    | 534 |
| Feltételes kódfordítás.....                                | 535 |
| Összefoglalás.....                                         | 537 |

### 13. A C# 2008 nyelv szolgáltatásai ..... 539

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| Implicit típusú lokális változók.....                           | 539 |
| A var használata foreach szerkezetekben.....                    | 541 |
| Az implicit típusú változókra vonatkozó korlátozások.....       | 542 |
| Implicit típusú lokális tömbök.....                             | 543 |
| Az implicit típusú adatok erősen tipizáltak.....                | 544 |
| Az implicit típusú lokális változók előnye.....                 | 545 |
| Automatikus tulajdonságok.....                                  | 546 |
| Interakció automatikus tulajdonságokkal.....                    | 548 |
| Az automatikus tulajdonságok elérésének korlátozása.....        | 549 |
| Az automatikus tulajdonságok és az alapértelmezett értékek..... | 549 |

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| Bővítő metódusok.....                                             | 551        |
| A bővítő metódusok definiálása .....                              | 552        |
| A bővítő metódusok meghívása példányszinten .....                 | 553        |
| A bővítő metódusok statikus meghívása .....                       | 555        |
| A bővítő metódus hatóköre .....                                   | 555        |
| A bővítő metódusokat definiáló típusok importálása .....          | 557        |
| IntelliSense a bővítő metódusokhoz.....                           | 558        |
| Bővítő könyvtárak létrehozása és használata .....                 | 559        |
| Interfész típusok kiterjesztése bővítő metódusokkal .....         | 560        |
| Részleges metódusok .....                                         | 562        |
| A részleges metódusok első pillantásra .....                      | 563        |
| A részleges metódusok használata .....                            | 565        |
| Az objektuminitializáló szintaxis.....                            | 566        |
| Egyedi konstruktor meghívása az initializációs szintaxissal ..... | 568        |
| A belső típusok inicializálása .....                              | 569        |
| Gyűjtemény inicializálók .....                                    | 571        |
| Névtelen típusok .....                                            | 572        |
| A névtelen típusok belső ábrázolása .....                         | 573        |
| A ToString() és a GetHashCode() megvalósítása .....               | 575        |
| Az egyenlőség szemantikája névtelen típusoknál .....              | 576        |
| Névtelen típusokat tartalmazó névtelen típusok .....              | 578        |
| Összefoglalás.....                                                | 579        |
| <b>14. A LINQ. Bevezetés .....</b>                                | <b>581</b> |
| A LINQ szerepe .....                                              | 581        |
| A LINQ-kifejezések jellemzői .....                                | 583        |
| (típzáltsággal és kiterjeszthetőséggel) .....                     | 583        |
| Alapvető LINQ-szerelvények .....                                  | 583        |
| Bevezetés a LINQ lekérdező kifejezésekbe .....                    | 585        |
| A LINQ és az implicit típusú lokális változók .....               | 588        |
| A LINQ és a bővítő metódusok .....                                | 590        |
| A késleltetett futtatás szerepe.....                              | 591        |
| Az azonnali futtatás szerepe .....                                | 592        |
| A LINQ és a generikus gyűjtemények.....                           | 593        |
| A LINQ-kifejezés alkalmazása .....                                | 594        |
| A LINQ és a nem generikus gyűjtemények .....                      | 595        |
| Adatok szűrése az OFType<T>() használatával.....                  | 596        |



|                                                                                                         |     |
|---------------------------------------------------------------------------------------------------------|-----|
| A LINQ-lekérdezésoперátorok belső ábrázolása .....                                                      | 597 |
| Lekérdező kifejezések létrehozása lekérdézésoперátorok segítségével. Ismételtes .....                   | 598 |
| Lekérdező kifejezések létrehozása az Enumerable típus és a lambdák használatával .....                  | 599 |
| Lekérdező kifejezések létrehozása az Enumerable típus és a névtelen metódusok használatával .....       | 601 |
| Lekérdező kifejezések létrehozása az Enumerable típus és a nyers metódusreferenciák használatával ..... | 602 |
| A LINQ-lekérdezésoперátorok .....                                                                       | 603 |
| Elemek számának meghatározása .....                                                                     | 604 |
| az Enumerable segítségével .....                                                                        | 604 |
| Egyszerű tesztprojekekre létrehozása .....                                                              | 605 |
| A kiválasztó szintaxis alapjai .....                                                                    | 606 |
| Adatok egy részhalmozának kinyerése .....                                                               | 608 |
| Új adattípusok leképezése .....                                                                         | 609 |
| Az eredményhalmoz-elemek sorrendjének megfordítása .....                                                | 610 |
| Kifejezések rendezése .....                                                                             | 610 |
| Különbségek keresése .....                                                                              | 611 |
| LINQ-lekérdezések: szigetek a szigeteken? .....                                                         | 612 |
| Lekérdezésoперátorok transzformálása Array típusa .....                                                 | 614 |
| Osszefoglalás .....                                                                                     | 615 |

#### 4. rész: Programozás .NET-szerelvényekkel

|                                                       |     |
|-------------------------------------------------------|-----|
| 15. A .NET-szerelvények .....                         | 619 |
| Egyedi névtér definíciója .....                       | 619 |
| A típus teljesen meghatározott neve .....             | 621 |
| Definíció alnevek használata .....                    | 623 |
| Begyazott névtér létrehozása .....                    | 624 |
| A Visual Studio 2008 „alapértelmezett névtér” .....   | 625 |
| A .NET-szerelvények szerepe .....                     | 626 |
| A szerelvények a kód újrafelhasználását segítik ..... | 626 |
| A szerelvények típushatárokat definiálnak .....       | 627 |
| A szerelvények verzióval rendelkező egységek .....    | 627 |
| A szerelvények önleírók .....                         | 627 |
| A szerelvények konfigurálhatók .....                  | 628 |

|                                                                     |     |
|---------------------------------------------------------------------|-----|
| A .NET-szerelvények felépítése.....                                 | 628 |
| A Win32-fejlec.....                                                 | 629 |
| A CLR-fejlec.....                                                   | 630 |
| A köztes nyelvi kód, a típusmetaadat és a szerelvénymanifestum..... | 631 |
| Optionális szerelvény-erőforrások.....                              | 631 |
| Egyfájlos és többfájlos szerelvények.....                           | 632 |
| Egy allományból álló szerelvény készítése és használata.....        | 634 |
| A manifestum.....                                                   | 638 |
| A köztes nyelvi kód.....                                            | 639 |
| A típusmetaadat.....                                                | 640 |
| C#-ügyfelalkalmazás készítése.....                                  | 641 |
| Visual Basic ügyfelalkalmazás készítése.....                        | 643 |
| A nyelvek közti származtatás.....                                   | 644 |
| Több allományból álló szerelvény készítése és használata.....       | 645 |
| Az ufo.netmodule fájl.....                                          | 647 |
| Az airvehicles.dll fájl.....                                        | 647 |
| A többfájlos szerelvény használata.....                             | 648 |
| Privát szerelvények.....                                            | 649 |
| A privát szerelvény azonosítója.....                                | 650 |
| A keresési sorrend (probing) megadása.....                          | 651 |
| Privát szerelvények beállítása.....                                 | 651 |
| Konfigurációs fájlok és a Visual Studio 2008.....                   | 654 |
| A .NET Framework konfiguráló segédprogramja.....                    | 655 |
| Megosztott szerelvények.....                                        | 657 |
| Erős nevek megértése.....                                           | 658 |
| A CarLibrary.dll ellátása erős névvel.....                          | 660 |
| Erős nevek hozzárendelése a Visual Studio 2008 segítségével.....    | 662 |
| Megosztott szerelvények telepítése/eltávolítása.....                | 662 |
| a globális szerelvénytárba.....                                     | 662 |
| A megosztott szerelvények felhasználása.....                        | 664 |
| A SharedCarLibClient manifestuma.....                               | 665 |
| A megosztott szerelvények konfigurálása.....                        | 666 |
| Az aktuális megosztott szerelvény befagyaszttása.....               | 667 |
| A megosztott szerelvény 2.0.0.0 verziójának elkészítése.....        | 668 |
| Dinamikus átírányítás a megosztott szerelvények.....                | 670 |
| meghatározott verziókba.....                                        | 670 |
| A .NET Framework konfiguráló programja – ismét.....                 | 672 |

|                                                   |     |
|---------------------------------------------------|-----|
| A GAC belső felépítése .....                      | 673 |
| Kibocsátó házirenddel rendelkező szerelvény ..... | 674 |
| A kibocsátó házirend leltára .....                | 676 |
| A <codeBase> elem .....                           | 677 |
| A System.Configuration névtér .....               | 679 |
| A gépi szintű konfigurációs fájl .....            | 680 |
| Összefoglalás .....                               | 681 |

## 16. Típusreflexió, késői kötés és attribútumalapú

### 683 programozás .....

|                                                             |     |
|-------------------------------------------------------------|-----|
| A metadatok szükségessége .....                             | 683 |
| Az EngineState felsorolás (részleges) metadata .....        | 685 |
| A Car típus (részleges) metadata .....                      | 686 |
| A TypeRef .....                                             | 687 |
| A definiáló szerelvény leírása .....                        | 687 |
| A hivatkozott szerelvények leírása .....                    | 688 |
| Sztringliterálok leírása .....                              | 688 |
| A reflexió .....                                            | 689 |
| A System.Type osztály .....                                 | 690 |
| Típusreferencia megszerzése a System.Object.GetType()       | 691 |
| segítségével .....                                          |     |
| Típusreferencia megszerzése a System.Type.GetType()         | 692 |
| segítségével .....                                          |     |
| Típusreferencia készítése a typeOf() segítségével .....     | 693 |
| Egyedi metaadat-nézegető létrehozása .....                  | 693 |
| A metódusok reflexiója .....                                | 693 |
| Mezők és tulajdonságok reflexiója .....                     | 694 |
| A típus által megvalósított interfészek reflexiója .....    | 695 |
| Egyéb érdekességek megjelölése .....                        | 695 |
| A Main() metódus megvalósítása .....                        | 695 |
| Generikus típusok reflexiója .....                          | 697 |
| Metódusparaméterek és visszatérési értékek reflexiója ..... | 698 |
| Dinamikusan betöltődő szerelvények .....                    | 700 |
| Megosztott szerelvények .....                               | 702 |
| A késői kötés .....                                         | 704 |
| A System.Activator osztály .....                            | 705 |
| Paraméter nélküli metódusok hívása .....                    | 706 |
| Paraméterekkel rendelkező metódusok hívása .....            | 707 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Programozás attribútumokkal.....                             | 708 |
| Attribútumok feldolgozása .....                              | 709 |
| Attribútumok alkalmazása a C# nyelvben.....                  | 710 |
| Attribútumok konstruktorparamétereinek meghatározása.....    | 712 |
| Az Obsolete attribútum működés közben.....                   | 713 |
| C#-attribútum rövidített formája.....                        | 713 |
| Egyedi attribútum készítése.....                             | 714 |
| Egyedi attribútumok alkalmazása.....                         | 715 |
| Nevesített tulajdonság szintaxisa .....                      | 716 |
| Az attribútumok használatainak korlátozása .....             | 716 |
| Alkalmazásszintű (és modulszintű) attribútumok.....          | 718 |
| A Visual Studio 2008 AssemblyInfo.cs fájlja.....             | 718 |
| Attribútumok korai kötés használatával.....                  | 720 |
| Attribútumok késői kötés használatával.....                  | 721 |
| Reflexió, késői kötés és egyedi attribútumok kifejtése ..... | 723 |
| Kiterjeszthető alkalmazás készítése .....                    | 724 |
| A CommonSnapppableTypes.dll létrehozása.....                 | 725 |
| C# beépülő modul létrehozása .....                           | 726 |
| Visual Basic beépülő modul készítése .....                   | 726 |
| Kiterjeszthető Windows Forms-alkalmazás készítése .....      | 727 |
| Összefoglalás.....                                           | 731 |

## 17. Folyamatok, alkalmazástartományok és objektumkörnyezetek..... 733

|                                                        |     |
|--------------------------------------------------------|-----|
| A tradicionális Win32-es folyamat áttekintése .....    | 733 |
| A szálak áttekintése .....                             | 734 |
| Folyamatok kezelése a .NET platformban .....           | 736 |
| A futó folyamatok listázása .....                      | 739 |
| Egy adott folyamat vizsgálata.....                     | 740 |
| Egy folyamat szálkészletének vizsgálata.....           | 741 |
| Egy folyamat modulkészletének vizsgálata.....          | 743 |
| Folyamatok programozott indítása és leállítása .....   | 745 |
| A .NET-alkalmazástartomány .....                       | 746 |
| Folyamat alkalmazástartományainak felsorolása .....    | 749 |
| Új alkalmazástartományok programozott létrehozása..... | 750 |
| Alkalmazástartományok programozott eltávolítása .....  | 752 |
| Az objektumkontextus határai.....                      | 754 |
| Kontextusváltó és kötött típusok.....                  | 755 |

## 18. Többszálú alkalmazások létrehozása ..... 761

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| A folyamat, az alkalmazástartomány, a kontextus és a szál kapcsolata ..... | 761 |
| A konkurencia problémája .....                                             | 763 |
| A szálszinkronizáció szerepe .....                                         | 764 |
| A .NET-metódusreferenciák rövid áttekintése .....                          | 764 |
| A metódusreferenciák aszinkron viselkedése .....                           | 767 |
| A BeginInvoke() és az EndInvoke() metódusok .....                          | 767 |
| A System.IAsyncResult interfész .....                                      | 768 |
| Metódushívás aszinkron módon .....                                         | 769 |
| A hívó szál szinkronizálása .....                                          | 770 |
| Az AsyncCallback metódusreferencia szerepe .....                           | 771 |
| Az AsyncResult osztály szerepe .....                                       | 773 |
| Egyedi állapotadatok átadása és fogadása .....                             | 774 |
| A System.Threading.Thread osztály .....                                    | 775 |
| Hozzáférés az aktuális szál statisztikáihoz .....                          | 778 |
| A Name tulajdonság .....                                                   | 779 |
| A Priority tulajdonság .....                                               | 780 |
| Második szál létrehozása kódból .....                                      | 781 |
| A ThreadStart metódusreferencia használata .....                           | 782 |
| A ParameterizedThreadStart metódusreferencia használata .....              | 784 |
| Előterben és háttérben futó szálak .....                                   | 786 |
| A konkurencia kérdése .....                                                | 787 |
| Szinkronizáció a C# lock kulcsszavával .....                               | 790 |
| Szinkronizáció a System.Threading.Monitor típussal .....                   | 792 |
| Szinkronizáció a System.Threading.Interlocked típussal .....               | 793 |
| Szinkronizáció a [Synchronization] attribútummal .....                     | 795 |
| Programozás időzített visszahívással .....                                 | 796 |
| A CLR ThreadPool .....                                                     | 798 |
| A BackgroundWorker komponens .....                                         | 800 |
| A BackgroundWorker típus használata .....                                  | 801 |
| Adatfeldolgozás a BackgroundWorker típussal .....                          | 803 |
| Összefoglalás .....                                                        | 805 |

|                                                                 |            |
|-----------------------------------------------------------------|------------|
| <b>19. A CIL és a dinamikus szerelevények szerepe .....</b>     | <b>807</b> |
| A CIL-programozás természete.....                               | 807        |
| CIL-direktívák, -attribútumok és -vezérlőkódok.....             | 808        |
| A CIL-direktívák szerepe.....                                   | 809        |
| A CIL-attribútumok szerepe .....                                | 810        |
| A CIL-vezérlőkódok szerepe.....                                 | 810        |
| A CIL-vezérlőkódok/CIL-mnemonikok megkülönböztetése .....       | 810        |
| Verem írása/olvasása: A CIL veremalapú természete.....          | 811        |
| A round-trip engineering.....                                   | 813        |
| A CIL-kódcímkek szerepe .....                                   | 817        |
| Fejlesztés a CIL-lel: egy *.il fájl módosítása .....            | 818        |
| A CIL-kód fordítása ilasm.exe használatával .....               | 820        |
| CIL-kód fordítása SharpDevelop használatával.....               | 821        |
| A peverify.exe szerepe .....                                    | 823        |
| CIL-direktívák és -attribútumok .....                           | 823        |
| Külsőleg hivatkozott szerelevények meghatározása CIL-ben.....   | 823        |
| Az aktuális szerelevény definiálása CIL-ben .....               | 824        |
| Névter definiálása CIL-ben.....                                 | 825        |
| Osztálytípusok definiálása CIL-ben .....                        | 826        |
| Interfészek definiálása és implementálása CIL-ben.....          | 827        |
| Struktúrák definiálása CIL-ben.....                             | 828        |
| Felsorolt típusok definiálása CIL-ben .....                     | 829        |
| Generikus típusok definiálása a CIL-ben .....                   | 829        |
| A CILTypes.il fájl fordítása.....                               | 830        |
| .NET-alapoztálykönyvtár, C#- és CIL-adattípus-leképezések ..... | 832        |
| Típustagok definiálása CIL-ben.....                             | 833        |
| Adatmezők definiálása CIL-ben .....                             | 833        |
| Típuskonstruktorok definiálása CIL-ben.....                     | 834        |
| Tulajdonságok definiálása CIL-ben.....                          | 835        |
| Tagparaméterek definiálása .....                                | 835        |
| A CIL-vezérlőkódok.....                                         | 836        |
| A .maxstack direktíva.....                                      | 839        |
| Lokális változók deklarálása CIL-ben.....                       | 840        |
| Paraméterek leképzése lokális változókba CIL-ben.....           | 841        |
| A rejtett „this” referencia .....                               | 842        |
| Ciklusok reprezentációja a CIL-ben .....                        | 843        |

|                                                                  |     |
|------------------------------------------------------------------|-----|
| .NET-szerelvény készítése köztes nyelven.....                    | 844 |
| A CILCars.dll létrehozása.....                                   | 844 |
| A CILCarClient.exe létrehozása.....                              | 847 |
| Dinamikus szerelvények .....                                     | 850 |
| A System.Reflection.Emit névtér .....                            | 851 |
| A System.Reflection.Emit.ILGenerator szerepe.....                | 852 |
| Dinamikus szerelvénnyel generálása .....                         | 853 |
| A szerelvénnyel és a modulkészlet generálása.....                | 856 |
| A ModuleBuilder típus szerepe .....                              | 857 |
| A HelloClass típus és a sztringtípusú tagváltozó generálása..... | 859 |
| A konstruktorok generálása .....                                 | 859 |
| A SayHello() metódus generálása .....                            | 861 |
| Dinamikusan létrehozott szerelvénnyel használata.....            | 861 |
| Összefoglalás.....                                               | 863 |
| Tárgymutató.....                                                 | 865 |
| A szakmai lektorról .....                                        | 879 |
| A szerzőről.....                                                 | 881 |





# Köszönetnyilvánítás

Jóllehet egyedül az én nevem jelenik meg a könyv borítóján, de a munka soha nem készülhetett volna el jó néhány tehetséges ember segítségével nélkül. A következőkben azokhoz szövegek, akik lehetővé tették, hogy ezt a könyvet megírjam.

Mindenekelőtt köszönet illeti az Apriss összes munkatársát, akikkel már több éve van szerencsém együtt dolgozni. Rendkívüli emberek, akik képesek voltak az én eredeti Word-dokumentumaimat tökéletesen prózává varázsolni. Hála köszönet érte. Már most nagyon várom, hogy mikor dolgozhatunk ismét együtt egy következő könyvön (de azért előtte szabadságolom magam egy időre).

Külön köszönöm technikai szerkesztőm, Gavin munkáját, aki annyi bölcs tanáccsal segített, hogy úgy érezem, ez a kiadás sokkal színvonalasabb lett, mint az előzőek. Mint mindig, az esetleges gépelési hibákért vagy technikai tévedésekért egyedül én vagyok a felelős.

Végül, de nem utolsósorban, köszönöm családomnak, barátainak és munkatársaimnak, hogy elvisejték magamra viselkedésemet, amely a kézirat elkészítésének utolsó szakaszaiban jó néhányszor jellemezett engem.



# Bevézetés

Ez a könyv nagyjából 2001 nyara óta létezik, ha nem is ebben a formában: a C# és a .NET platform első kiadása ugyanis akkor jelent meg, a .NET 1.0 Beta 2 verziójával karöltve. Nagy örömmel és hálával tölt el, hogy munkám kedvező fogadtatásra lelt a sajtóban, és ami még fontosabb, az olvasók körében is. Az évek során jelölték a Jolt Award döntőjére (vesztettem... kellemtelen) és a 2003-as Referenceware Excellence Awardon a programozási könyvek kategóriájában (nyertem?... de jó!)

Azóta azon dolgoztam, hogy a könyvet frissítssem és aktualizáljam a .NET platform újabb verzióinak megfelelően. Ekközben limítált darabszámmal megjelenttem egy speciális kiadványt, amely bemutatta a .NET 3.0 verzióját (Windows Presentation Foundation, Windows Communication Foundation, és a Windows Workflow Foundation), valamint áttekintést nyújtott olyan külbővítő technológiákról, amelyek még megjelenésre vártak, és amelyeket most LINQ néven ismerünk.

Jelenleg a könyv negyedik kiadását tartja a kezében az Olvasó. Ez a kiadás egyrészt tömören megismétli a korábbi ismereteket, azaz a .NET 3.5-ös verziójában található legfontosabb változások magyarázatát, másrészt számos teljes új fejezetet is tartalmaz, de ezen túlmenően több eddigi rész is jelentősen kibővült.

Ahogy a korábbiakban megszokhattuk, ez a kiadás is egyszerű és érthető nyelven ismerteti a C# nyelvet és a .NET-alapozztálykönyvtárakat. Soha nem értem azokat a szakírókat, akik úgy adják elő mondanójukat, hogy az jobban hasonlít egy GRE-szöszedeletről írt tanulmányra, mint egy szakkönyvre. Munkám továbbra is arra fekteti a hangsúlyt, hogy a benne található információk segítségével jól használható szoftverrendszereket lehessen készíteni, így nem töltök túl sok időt az ezoterikus részletek tanulmányozásával, hiszen ezekkel csak nagyon kevés olvasóm fog valaha is foglalkozni.

## Mi egy csapat vagyunk, az Olvasó és én

A szakírók az emberek igényes csoportjának írnak (nekem már csak tudnom kell – egy vagyok közülük). Tisztaban vagyunk azzal, hogy egy szoftver elkészítése – bármelyik platformmal történjen is (NET, J2EE, COM stb.) – rendkívül aprólékos munka, ugyanakkor nagyon specifikus a részleg, a cég, az ügyfél vagy a tárgykör szerint. Hiszen lehet, hogy épp az elektronikus kiadói iparágban dolgozunk, vagy éppenséggel rendszerrel fejlesztünk az állam-adói iparágban vagy a helyi önkormányzatnak, a NASA-nak vagy egy katonai részlegnek. En a magam részéről gyermeketek oktató szoftvert fejlesztetem, számos n-rétégű rendszerrel, valamint projekteket a gyógygyászati és a pénzügyi ágazatokban. Majdnem nulla az esélye, hogy annak a kódnak, amelyet egyikünk ír a munkájához, köze lesz ahhoz a kódhoz, amelyet a másikunk ír a saját munkájához (kivéve, ha véletlen épp együtt dolgoztunk korábban!).

Ezért szándékosan kerülöm az olyan mintapéldákat, amelyek konkrét iparághoz vagy programozási feladathoz köthetnének: a C#, az objektumorientált programozás és a .NET 3.5 alapoztály könyvtárait ipari alkalmazásoktól független példákra keresztül mutatom be. Ahelyett, hogy minden példám adatok tömkelegét szolgáltatná, berlistát számoyna, vagy ehhez hasonlókkal szolgálna, az én példáimhoz mindenkinek van valami köze: autókrol szólnak (ráadásként néhány geometriai struktúrával és alkalmazottakkal). És ez az a pont, ahol összetalálkozunk.

Az én feladatam, hogy legjobb tudásom szerint elmagyarázzam a C# programozási nyelvet és a .NET platform alapvető jellegzetességeit. Ugyanígy mindent megteszek azért, hogy ellássam a kedves Olvasót különféle eszközökkel és stratégiákkal, amelyek lehetővé teszik, hogy a könyv alapján folytathasson a téma tanulmányozását.

Az olvasó feladata pedig, hogy a megszerzett tudást képes legyen alkalmazni saját programozási feladataiban. Persze egyértelmű, hogy a projekteknem állatnevekkel ellátott autómobilok körül forognak, de hát pont erről szólna az alkalmazott ismeret. Biztos lehet benne az Olvasó, hogy ha egyszerű megértette a könyvben bemutatott elveket, ismereteket birtokában olyan .NET-al alkalmazásokat írhat, amelyek használhatóak saját programozási környezetükben.

## A könyv áttekintése

A *Pro C# 2008 and the .NET 3.5 Platform (A C# és a Microsoft .NET 3.5.)* negyedik kiadása logikailag nyolc különálló részből áll, s ezek mindegyike további fejezeteket tartalmaz. Jó néhány témakör, például az alapvető C#-szerkezetek, az objektumorientált programozás és a platformfüggetlen .NET fejlesztése kiövélt a korábbiakhoz képest, saját fejezetet kapott. Az új kiadás a .NET 3.0-3.5 programozás tulajdonságait (LINQ, WCF, WPF, WF stb.) összefoglaló fejezeteket is tartalmaz.

**Megjegyzés** A magyar változat kiadásának előkészítése során a SZAK Kiadó úgy döntött, hogy az eredeti könyv tartalmát két kötetben jeleníti meg. Ezt a viszonylag nagy terjedelem indokolja. Az alábbiakban mind a két kötet tartalmát ismertetjük, így biztosítva, hogy az is képet kapjon a mű egészéről, akinek csak az egyik kötet kerül a kezébe. (Ezt a Bevezetést változatlan formában mind a két kötetben szerepeltetjük.)

### Első kötet

## 1. rész: Bevezetés a C#-ba és a .NET platformba

Az első rész bemutatja a .NET platform alapvető természetét és számos olyan fejlesztőeszközt (közük több nyílt forráskódú), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány alap C# programozási nyelvi részlettel és a .NET-típusrendszerrel is megismerkedünk.

### 1. fejezet: A .NET filozófiája

Az első fejezet voltaképpen a könyv további részeinek a gerincét képezi. A hagyományos Windows-fejlesztések világának vizsgálatával kezdjük, és feltárjuk a korábbi hátrányokat is. A fejezet elsődleges célja azonban az, hogy megismerjünk jó néhány .NET-központú építőkövet, például az egysegű nyelvi futtatót (Common Language Runtime – CLR), a közös típusrendszert (Common Type System – CTS), az egysegű nyelvi specifikációt (Common Language Specification – CLS) és az alaposztálykönyvtárakat. Egyszerű betekintést nyerhetünk a C# programozási nyelvbe és a .NET-szerelvényformátumba, valamint áttekintést kapunk a .NET platform platformfüggetlen természetéről (a B függelék részletesen tárgyalja ezt a témakört).

## 2. fejezet: C#-alkalmazások készítése

A fejezet célja, hogy különféle eszközök és technikák ismeretével bevezessen minket a C#-forráskód[ra] fordításának folyamatába. Először a parancssoros fordító (csc.exe) és a C#-eredményfájlok használatát ismerhetjük meg. A fejezet további részében pedig számos forráskód-szerkesztőt és integrált fejlesztési környezetet (integrated development environments – IDEs) tanulmányozhatunk, többek között a következőket: TextPad, Notepad++, SharpDevelop, Visual C# 2008 Express és a Visual Studio 2008. Ezenkívül számos olyan további programozási eszközt ismertetünk, amelyet bármely .NET-fejlesztőnek azonnal elő kell tudnia huzni a farszéből.

## 2. rész: A C# alapvető építőelemei

A 2. részben tárgyalunk témakörök kiemelten fontosak, hiszen az itt megjelenő tudnivalókat folyamatosan használnjuk függetlenül attól, hogy milyen jellegű .NET-szoftvert szándékozunk fejleszteni (webes alkalmazás, asztali grafikus alkalmazás, forráskódkönyvtárak, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek a C# nyelv alapvető építőkövei, beleértve az objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti továbbá, hogy hogyan kell kezelni a futásidőű kivételeket, és bevezet minket a .NET személynévű szolgáltatójának részleteibe is.

## 3. fejezet: A C# alapvető építőelemei, I. rész

Ebben a fejezetben kezdjük el részletesen tanulmányozni a C# programozási nyelvet. Megismerkedünk a `Main()` módszer szerepével és számos részlettel a .NET platform belső adattypusaival kapcsolatban, beleértve a szöveges adatkezelést a `System.String` és a `System.Text.StringBuilder` névterek alkalmazásával. Megvizsgáljuk továbbá az iterációs és a döntési szerkezeteket, a szűkítő és a bővítő műveleteket, valamint az unchecked kulcsszó használatát.

## 4. fejezet: A C# alapvető építőelemei, II. rész

A 4. fejezet befejezi az alapvető C#-szemléletek meghatározását, a típusúterhelte metódusok létrehozását az `out`, `ref` és `params` kulcsszavakon keresztül paraméterek definiálásáig. Vizsgáljuk, hogyan kell adathalmazt létrehozni és kezelni, hogyan kell olyan adattípust készíteni, amelynek `null` is lehet az értéke (a `?` és a `??` operátorokkal), és megmutatjuk az *értéktípus* és a *referenciátípus* közötti különbséget is.

## 5. fejezet: Egységbe zárt osztálytípusok definiálása

Ebben a fejezetben az objektumorientált programozással (OOP) ismerkedhetünk meg a C# programozási nyelv használatán keresztül. Először meghatározzuk az OOP pilléreit (beágyazás, származtatás és polimorfizmus), majd a fejezet további részében azt nézzük meg, hogyan kell konstruktorok, tulajdonságok, statikus tagváltozók, konstansok és csak olvasható fájlok használatával robusztusosztály-típusokat létrehozni. Végezetül ismertetjük a részleges típus definícióját és a C# XML-kódjának dokumentációs szintaktikáját.

## 6. fejezet: A származtatás és a polimorfizmus

Ebben a részben az OOP maradék pilléreit (származtatás és polimorfizmus) vizsgáljuk meg; ezek a segítségével összetartozó osztálytípusok családját hozhatjuk létre. Megnézzük a virtuális metódusok és absztrakt metódusok (és az absztrakt alaposztályok) szerepét, valamint a *polymorf interfész* természetét. Végül, de nem utolsósorban, ez a fejezet ismerteti a .NET platform legfontosabb alaposztályának, a *system.object* osztálynak a szerepét is.

## 7. fejezet: Strukturált hibakezelés

Ez a fejezet elsősorban azt mutatja be, mit kell tenni a strukturális kivételkezelés segítségével a futási időben keletkezett anomáliákkal. Nemcsak az ilyen jellegű problémák megoldására szolgáló C#-kulcsszavakat ismerjük itt meg (try, catch, throw és finally), hanem az alkalmazásszintű és a rendszerszintű kivételek közötti különbséget is. Megismerhetünk továbbá többféle, a Visual Studio 2008-ban található eszközt, amelyek segítségével hibakeresést végezhetünk a figyelmen kívül hagyott kivételek között.

## 8. fejezet: Az objektumok életciklusa

Ennek a résznek az utolsó fejezete azt tárgyalja, hogy hogyan kezeli a CLR a memóriát a .NET személynévtáblájának a segítségével. Megérthetjük az alalmazásgyökök, az objektummemorizációk és a *system.gc* típus szerepét. Az alapok megismerése után, a fejezet további része az *eladható objektumokkal* (az *IDisposable* interfészen keresztül) és a véglegesen lefolyamattal (a *system.object.Finalize()* metóduson keresztül) foglalkozik.

### 3. rész: Haladó programozási szerkezetek a C#-ban

A könyv 3. része ismerteti a C# nyelvet haladóbb jellegű (de ugyancsak nagyon fontos) elveit. Az interfészek és metódusreferenciák vizsgálataival befejezzük a .NET típusrendszerével való ismerkedést. A továbbiakban betekintést nyerhetünk a generikus típusok és a számos új C# 2008 nyelvi elem szerepébe, valamint a LINQ-ba is.

#### 9. fejezet: Interfészek használata

Ennek a fejezetnek az anyagát az interfészalapú programozást is magába foglaló objektumalapú fejlesztés alkotja. Megismerhetjük belőle, hogy hogyan kell olyan típusokat definiálni, amelyek többszörös viselkedést támogatnak, hogyan lehet ezeket a viselkedéseket futásidőben feltekerpezni, és hogyan lehet szelektíven elrejteni bizonyos viselkedéseket az *explicit interfészmegvalósítás* segítségével. Továbbá a számos előre definiált .NET-interfésztípus megismeréséhez el kell sajátítanunk azoknak az egyedi interfészeknek a használatát, amelyek ad hoc jellegű eseményarchitektúra készítéséhez szükségesek.

#### 10. fejezet: Gyűjtemények és generikus típusok

Ez a fejezet a `System.Collections` névtérgyűjtemény típusainak tanulmányozásával kezdődik; ezek már a .NET platform kezdeti kiadásának is részei voltak. Noha a .NET 2.0 megjelenése óta a C# programozási nyelv lehetővé teszi *generikus* típusok használatát, ahogy a későbbiekben kiderül, a generikus programozás lényegesen megkönnyíti az alkalmazás teljesítményét és a típusbiztonságot. Nemcsak azt vizsgáljuk meg, hogy milyen különböző generikus típusok léteznek a `System.Collections.Generic` névtérben, de azt is, hogyan készíthetjük el saját generikus metódusainkat és típusainkat (megszorításokkal vagy anélkül).

#### 11. fejezet Metódusreferenciák, események és lambdák

A 11. fejezetnek az a célja, hogy érthetővé tegye a *metódusreferencia*-típusokat. Leegyszerűsítve: a .NET-metódusreferencia olyan objektum, amely az alábbiakhoz hasonlóan működés segítségével a .NET-metódusreferenciák vizsgálata után megismerkedünk a .NET event kulcsszavával, amelyet arra használunk, hogy egyszerűsítsük a nyers metódusreferencia programozásának kezelését. Végezetül megvizsgáljuk a C# 2008 lambda-operátort (`=>`), és felhívjuk a kapcsolatot a metódusreferenciák, a névtelen metódusok és a lambda-kifejezések között.



## 12. fejezet: Indexerek, operátorok és mutatók

Ez a fejezet a C# programozási ismereteinket mélyíti el számos haladó programozási technika segítségével. Megismerhetjük, hogyan kell operátorokat tültetni, és hogyan kell a típusaink számára egyedi konverziós rutinokat írni (akár implicit, akár explicit formában). Megtanulhatjuk azt is, hogy hogyan kell létrehozni és használni a *tipusindexelőket*, és hogyan kell kezelni a C-stílusú mutatókat a „nem biztonságos” *kódkörnyezetben*.

## 13. fejezet: C# 2008 nyelvi újdonosságai

A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközzel bővült, amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API használatát (erről a 14. fejezetben többet is megtudhatunk). Megismerjük a lokális változók implicit típusú ellátásának a szerepét, a részleges metódus, az automatikus tulajdonságok, a bővítő metódusok, a névtelen típusok és az objektuminitializáló szintaxis használatát.

## 14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (LINQ)

Ez a fejezet ismerteti a LINQ-et (Language Integrated Query – nyelvi integrált lekérdezés), amelyet nyugodtan nevezhetünk a .NET 3.5 legérdekesebb részének. Ahogy a fejezet későbbi részéből kiderül, a LINQ segítségével lehetőségek nyílnak erősen típusos *lekérdezéseket* írni, amelyeket számos LINQ-célpontra alkalmazhatunk, hogy a szó legtagabb értelmében „adatot” kezelhessünk. Megtanuljuk a LINQ objektumokra történő alkalmazását, ennek segítségével a LINQ-kifejezéseket adattárolókra (tömbök, kollekciók, egyedi típusok) fogjuk tudni alkalmazni. Ez a tudás nélkülözhetetlen lesz akkor, amikor azt vizsgáljuk meg, hogyan kell LINQ-kifejezéseket relációs adatbázisokon (a LINQ az ADO-n keresztül) és XML-dokumentumokon (a LINQ az XML-en keresztül) használni.

## 4. rész: Programozás .NET-szerelvényekkel

A negyedik rész a .NET-szerelvényformátum részleteit taglalja. Nemcsak azt tanulhatjuk meg, hogyan kell .NET-kódkörnyvtárakat telepíteni és konfigurálni, de azt is, hogy mi a .NET bináris kép belső összeállítására. Ebben a részben lesz szó a .NET-attribútumok szerepéről és a többszálú alkalmazások létrehozásának módjáról is. A későbbi fejezetek néhány haladó témát érintenek, például az objektumkörnyezet, a köztes nyelvi kód és a dinamikus szerelvényeket.

## 15. fejezet: A .NET-szerelvények

Első megközelítésben a *szerelvény* nem más, mint egy felügyelt \*.dll vagy \*.exe bináris fájl leírására szolgáló kifejezés. Am a .NET-szerelvények története sokkal több ennél. Ebben a fejezetben megtudjuk, mi az egyfajlos és a többfajlos szerelvények közötti különbség, és, hogy ezeket hogyan kell készíteni és telepíteni. Megvizsgáljuk, hogyan kell privát és megosztott szerelvényeket konfigurálni XML-alapú \*.config fájlokkal és kibocsátó házirenddel rendelkező szerelvényekkel. Mindeközben megtudhatjuk, hogy milyen a GAC (Global Assembly Cache – globális szerelvénytár) belső struktúrája, és mi a .NET keretrendszer konfigurációs segédprogramjainak a szerepe.

## 16. fejezet: Típusreflexió, késői kötés és attribútumalapú programozás

A 16. fejezetben folytatjuk az ismerkedést a .NET-szerelvényekkel. A system.Reflection névtér használatával kapcsolatban azt vizsgáljuk, a futási időben hogyan valósul meg a típusmeghatározás folyamata. Az ilyen jellegű típusok használatával lehetőségünk van olyan alkalmazások létrehozására, amelyek futás közben képesek a szerelvények metaadatait olvasni. Kiderül, hogyan lehet a *késői kötés* segítségével futási időben dinamikusan betölteni és létrehozni típusokat. A fejezet utolsó témaköre – mind a szabványos, mind az egyedi – .NET-attribútumok szerepének ismertetése. Az egyes témakörök hasznosságának szemléltetésére a fejezet konklúziójaként bemutatunk egy kiterjeszthető Windows Forms-alkalmazást.

## 17. fejezet: Folyamatok, alkalmazástartományok és objektumkörnyezetek

A szerelvényekről szóló tudnivalók után ez a fejezet a betöltött .NET végre-hajtható fájlok felépítését ismerteti. Az elsődleges cél a folyamatok, az alkalmazástartományok és a kontextusbeli határok közötti kapcsolat illusztrálása. Ezek alapozzák meg a következő fejezet témáját: a többszálú alkalmazások készítésének módját.

## 18. fejezet: Többszálú alkalmazások készítése

Ez a fejezet azt mutatja be, hogy hogyan kell többszálú alkalmazásokat készíteni, és számos olyan technikát ismertet, amelyeket a szálbiztos kód készítéséhez alkalmazhatunk. A fejezet elején áttekintjük a .NET metódusreferenciátípusait, hogy megértjük, valójában hogyan támogatják a metódusreferenciák

az aszinkronfüggvény-hívásokat. Majd megvizsgáljuk a `system.thread` ben található típusokat. Ezek között számos olyan van (`thread`, `threadstart`), amelynek segítségével könnyen lehet újabb végrehajlási szálakat készíteni. Végül megismerkedünk a `backgroundworker` típussal, amely nagyban megkönnyíti a felhasználói felületről történő szálkészítést.

## 19. fejezet: A köztes nyelv (CIL) és a dinamikus szerelevények

A 4. rész utolsó fejezetének szerepe kettős. Először a korábbinál részletesebben megvizsgáljuk CIL szintaktikáját és szemantikáját, majd a `system.reflection`. Emi t névter szerepét mutatjuk be. Ezeknek a típusoknak a használatával lehetőségek nyílnak olyan szoftver készítésére, amely képes .NET-szerelevényeket futásidőben generálni a memóriába. Az olyan szerelevényeket, amelyeket a memóriában definiálunk és futtatunk, *dinamikus szerelevényeknek* nevezzük.

## Második kötet

## 5. rész: Bevezetés a .NET alaposztálykönyvtáraiba

Ha elérkeztünk ehhez a részhez, akkor már biztos tudásunk van a C# nyelvről és a .NET-szerelevényformátumok részleteiről. Az ötödik rész továbbfejlesztett ezeket az ismereteket azáltal, hogy számos általánosan használt alaposztálybeli szolgáltatást mutat be, többek között a fájl I/O-t és az adatbáziselszólást az ADO.NET segítségével. Ez a rész tárgyalja továbbá az elosztott alkalmazások készítésének témakörét a Windows Communication Foundation (WCF) segítségével, valamint azokat a munkafolyamat-engedélyezett alkalmazásokat készítését, amelyek a Windows Workflow Foundation (WF) API-t használják.

## 20. fejezet: Fájlműveletek és elkülönített tárolás

A `system.io` névter segítségével lehetőségek vannak a számítógép fájl- és könyvtárszerkezetének elérésére. Ebben a fejezetben megismerjük, hogyan kell programozottan könyvtárrendszert létrehozni (és törölni), valamint, adatokat különböző folyamatokba (fájlalapú, sztringalapú, memórialapú stb.) be-, illetve kihozni. A fejezet utolsó része megvizsgálja az *elkülönített tárolást*, amelynek segítségével lehetőségek vannak biztonságos helyen tárolni az adatokat függetlenül attól, hogy a cél gép biztonsági beállításaitól. Hogy megértsük a `system.io.isolated API` bizonyos elveit, áttekintést kell kapnunk a kódhozzáférés-szabályozásról (Code Access Security – CAS).

## 21. fejezet: Bevezetés az objektumsorosítás világába

Ez a fejezet a .NET-platform objektumsorosító szolgáltatásába nyújt betekintést. Egyszerűen szólva a *sorosítás* segítségével lehetőségünk nyílik arra, hogy egy objektum (vagy a kapcsolódó objektumok egy halmazának) állapotát egy formában tároljuk a későbbi használat céljából. A *deszerializálás* (a sorosítás ellentétes művelete) az a folyamat, amelynek segítségével kivéhejük az objektumot a folyamatból, és a memóriában az alkalmazás számára használható formába hozhatjuk. Az alapok elajátítása után megtanulhatjuk, hogyan kell egyedivé tenni a sorosítási folyamatot az *ISerializable* interfacc és a .NET-attribútumok halmazának segítségével.

## 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat

Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kapunk ismereteket. Ez a rész bevezeti a .NET adatszolgáltatószerpét, valamint azt, hogy hogyan kell a relációs adatbázissal kommunikálni az ADO.NET *elő kapcsolatán* keresztül. Ez a kapcsolati objektumokat, a parancsobjektumokat és az adatolvasó objektumokat jelenti. Ezenkívül ez a fejezet mutatja be az egyedi adatbázis létrehozásának módját, és ismereti azokat az adateleléési könyvtárakat, amelyeket majd a könyv további részében használunk.

## 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat

Ez a fejezet az adatbázis-kezelés különböző módokon történő manipulálási lehetőségeit tárgyalja tovább, jelen esetben az ADO.NET *bontott kapcsolatán* keresztül. Megismerkedhetünk a dataset típus és az adatfeldolgozó objektumok szerpével, valamint számos Visual Studio 2008 eszközzel, amelyek nagyban megkönnyítik az adatvezérelt alkalmazások létrehozását. Mind-eközben rávilágítunk arra, hogyan kell a datarab-objektumokat felhasználói felületi elemekhez kapcsolni, olyanokhoz, mint például a Windows Forms API *GridView* típusa.

## 24. fejezet: A LINQ API programozása

A 24. fejezet a LINQ programozási modelt mutatja be, pontosabban a LINQ „az objektum irányába” részét. Megvizsgáljuk, hogyan kell LINQ-ekérdezéseket alkalmazni relációs adatbázisokon, dataset-objektumokon és XML-dokumentumokon. Mindeközben megtanuljuk az adatkontextus-objektum, valamint az *sqlmetat.exe* segédprogram szerepét, és megvizsgáljuk a Visual Studio 2008 különböző LINQ-támogató funkcióit.

## 25. fejezet: A WCF

A .NET 3.0 bevezetett egy teljesen új API-t, a WCF-et, amely lehetőséget ad arra, hogy szimmetrikus módon, függetlenül az alatta található rétegek felépítésétől elosztott alkalmazást fejlesszünk. Ez a fejezet feltárja a WCF-szol-gáltatások, -gazdagépek és -kliensek készítesének lehetőségeit. A WCF-szol-gáltatások rendkívül flexibilibisek, a gazdagépek és a kliensek XML-alapú kon-figurációs fájlokat használhatnak, hogy deklaratív módon adjanak meg cí-meket, kötéseket és szerződéseket.

## 26. fejezet: A WF

A .NET 3.0 a WCF mellett bevezette a WF API-t is, amely lehetőséget ad arra, hogy *munakafolyamatokat* definiáljunk, futtassunk és monitorozzunk. Ennek segítségével komplex üzleti folyamatok modellezhetők. Megtanuljuk a WF általános célját, ahogy azt is, hogy mi a szerepe az aktivitásoknak, a munka-folyamat tervezőnek, a munka-folyamat futtatási motornak, és a munkafo-lyamat-engedélyezett forráskód könyvtáraknak.

## 6. rész: Felhasználói felületek

Általánosan hibás feltételezés a .NET platformmal kapcsolatban az, hogy ez a keretrendszer kizárólag a webalapú felhasználói felületekkel foglalkozik (ez feltehetően a „NET” kifejezés miatt van, hiszen felidézi az „Internet” szót, azaz a „webprogramokat”). A .NET kiemelkedő módon támogatja a webal-kalmazások készítését, a könyv ezen része azonban a hagyományos felhasználói felületekre koncentrált két grafikus felület keretrendszerén, a Windows Forms és a WPF használatán keresztül.

## 27. fejezet: Windows Forms-programozás

Az eredeti asztali grafikus felhasználói felület, amely a .NET platformmal együtt kapható, a *Windows Forms*. Ez a fejezet ismerteti a felhasználói felület keretrendszerének szerepét, és bemutatja, hogyan kell főablakokat, párbe-szédablakokat és menürendszereket készíteni. Feltárja továbbá az űrlap-szármasztás szerepét, és azt, hogy hogyan kell kétdimenziós adatot rajzolni a System.Drawing névtér segítségével. A témakör illusztrálására a fejezet vé-gén készítünk egy (korlátozott funkcionálitással bíró) rajzolóalkalmazást.

## 28. fejezet: A WPF és az XAML

A .NET 3.0 teljesen új grafikus felhasználói felületet vezetett be, amelyet WPF-nek nevezett el. Röviden, a WPF kimagaslóan interaktív és médiában gazdag front endek készítését teszi lehetővé asztali alkalmazásokhoz (és indirekt módon webalkalmazásokhoz). A Windows Formsszal ellentétben, ez a túlteleített felhasználófelület-keretrendszer számos kulcsfontosságú szolgáltatást (2D-s és 3D-s grafika, animációk, gazdag dokumentumok stb.) integrál egyetlen egységesített objektummodellbe. Ebben a fejezetben megkezdjük az ismerkedést a WPF-fel és a kiterjeszthető alkalmazásfejelő nyelvvel (Extendable Application Markup Language – XAML). Megtanuljuk, hogyan kell XAML-mentes, csak XAML-t használó és a kettő kombinációjából felépülő WPF-programokat létrehozni. A fejezet végén készítsünk egy egyedi XAML-nézegetőt, amelyet a további WPF-fel foglalkozó fejezetekben is használni fogunk.

## 29. fejezet: Programozás WPF-vezérlőelemekkel

Ebben a fejezetben megtudjuk, hogy hogyan kell dolgozni a WPF-vezérlőelem tartalommodelljével, és érintünk számos vezérlőelemekkel kapcsolatos témát is, például a függőségi tulajdonságokat és irányított eseményeket. Jó néhány WPF-vezérlőelem használatát bemutatójuk, szintén ebben a fejezetben magyarázzuk el az elrendezéskezelő, -vezérlő utasítások és a WPF-adatkötés modelljének a használatát.

## 30. fejezet: WPF 2D grafikus renderelés, erőforrások és témák

Az utolsó fejezet a WPF tárgyalását három látszólag független téma vizsgálatával zárja. A WPF grafikus renderelő szolgáltatásának általában szűksége van arra, hogy egyedi erőforrásokat definiáljunk hozzá. Ezeknek az erőforrásoknak a használatával lehetőségünk van egyedi WPF-animációk készítésére, és grafikát, erőforrásokat és animációkat alkalmazva egyedi témák készítését fogjuk tudni megoldani WPF-alkalmazások számára. A különböző témák összeegyeztetéseppen a fejezet utolsó részében azt illusztráljuk, hogyan kell futásidőben egyedi grafikus témákat alkalmazni.

## 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével

A 7. rész az ASP.NET programozási API használatával készített webes alkalmazásokat vizsgálja. Az ASP.NET szándékosan úgy lett kialakítva, hogy az asztali felhasználói felület létrehozását modellezze, úgy, hogy szabványos HTTP kérésekre/válaszokra helyez egy eseményvezérelt, objektumorientált keretrendszert.

### 31. fejezet: ASP.NET weboldalak készítése

Ez a fejezet vezet be bennünket azASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltják az igazi objektumorientált nyelvek (mint a C#, VB.NET és hasonlók). Bemutatójuk egy ASP.NET weboldal készítését, valamint az alatta található programozási modellel és az ASP.NET egyéb kulcsfontosságú elveit, például azt, hogy hogyan kell webkiszolgálót választani, valamint a web.config fájlok használatát.

### 32. fejezet: ASP.NET-vezérlőelemek, témák és mesteroldalak

Ez a fejezet azokkal a vezérlőelemekkel foglalkozik, amelyek a belső vezérlőfát töltik fel adattal. Megismerjük az alapvető ASP.NET webes vezérlőelemeket, többek között a ellenőrző vezérlőelemet, a belső oldali navigációs vezérlőelemeket és a különböző adatköötő műveleteket. Ugyancsak ennek a fejezetnek a feladata, hogy bemutassa a *mesteroldalak* és az ASP.NET témamotorjának a szerepét, amely a hagyományos stíluslapok szerveroldali megfeleltője.

### 33. fejezet: ASP.NET állapotkezelési technikák

Ez a fejezet kiterjeszti az eddigi ASP.NET-ismereteinket, ugyanis megvizsgálja, milyen különböző módokon lehet állapotot kezelni a .NET alatt. Ahogy a klasszikus ASP, az ASP.NET is egyszerűen teszi lehetővé sütik létrehozását, ahogy alkalmazásszintű és munkafolyamat-szintű változók használatát is. Az ASP.NET azonban bevezet egy új állapotkezelő technikát: az alkalmazás-gyorsítótárat. Ha már tudjuk, milyen módokon lehet állapotot kezelni az ASP.NET alatt, megismerkedünk a system.httpapplication alapszály szerepével (a global.asax fájlon belül), valamint azzal, hogy hogyan kell dinamikusan megváltoztatni a webalkalmazásunk futásidejű viselkedését a web.config fájl segítségével.

## 8. rész: Függelék

A könyv utolsó része két fontos témát vizsgál meg, amelyek valójában nem illeszkedtek szervesen a könyv felépítésébe, ezért a függelékbe kerültek. A függelék kitérjésíti a C#-pal és .NET platformmal kapcsolatos ismereteinket, megvizsgáljuk ugyanis, hogyan kell régebbi kódokat integrálni a .NET-alkalmazásainkba, valamint hogyan kell a .NET-fejlesztést a Windows operációs rendszer családján kívül használni.

### A függelék: A COM és a .NET együttműködése

Azok, akik már programoztak Windows operációs rendszer alatt a .NET platform használatát előtt, nagy valószínűséggel ismerik a Component Object Modelt (COM). Noha a COM-nak és a .NET-nek semmi köze egymáshoz (azon túl, hogy mindkettő a Microsoft szerzeménye), a .NET platform teljes névtérrel biztosít ahhoz (system.runtime.interopservicess), hogy .NET-szoftverek COM-komponenseket használjanak, és fordítva. Ez a függelék az együttműködési réteget mutatja be egészen részletesen; ez a téma azért nagyon fontos, hogy a korábban megírt programjainkat át tudjuk menteni az újonnan készített .NET-alkalmazásainkba.

### B függelék: Platformfüggetlen .NET-fejlesztés a Monoval

Végül, de nem utolsósorban a B függelék a .NET nyílt forráskódú implementációjával, a *Monoval* foglalkozik. A Mono segítségével lehetőségünk van gazdaság tulajdonságokkal rendelkező .NET-alkalmazás létrehozására, telepítésére és futtatására különböző operációs rendszerekben, mint például a MacOS X, Solaris, AIX és számos Linux-disztribúció. Mivel a Mono nagyjából kompatibilis a Microsoft .NET platformjával, így már tisztában vagyunk azazal, hogy mit tud nyújtani számunkra. Ezért a függelék a Mono installálását tárgyalja, valamint azt, hogy, milyen fejlesztői eszközök állnak a rendelkezésünkre, és hogyan működik a Mono futtatómotorja.



## Öt szabadon letölthető fejezet – még több információ

Öt szabadon letölthető fejezet – még több információ

Azoknak, akiknek a 33 fejezet és a két függeléket nem lenne elég, szabadon letölthetnek további öt fejezetet. A könyv korábbi verziói három olyan fejezetet tartalmaztak, amelyeket a Windows Forms fejlesztésének szenteltünk (egyedi vezérlők vizsgálataival egyetemben), egy másik fejezetet a .NET remoting rétegének (system.runtime.remoting és társai), és a végül egy fejezetet a hagyományos XML-webszolgáltatások készítésével foglalkozott az ASP.NET Web Service projekt sablonjainak a segítségével.

A könyv jelen kiadása nem tartalmazza ezt az öt fejezetet. Ennek az a legfőbb oka, hogy a .NET 3.0-ban a WCF és a WFP API-k rendre a .NET-remoting/XML-webszolgáltatások és a Windows Forms API-k örökösei. Ha mélyebben szeretnénk elmerülni a Windows Forms világában (azon túl, amit a 27. fejezet nyújtani tud), vagy meg akarjuk nézni, hogyan kell használni a (örökségül hagyott) .NET-remotingot és az XML-webszolgáltatás-API-t, egyszerűen csak meg kell nézni az Apress honlapján a megfelelő részt:

<http://apress.com/book/view/1590598849>.

Az itt található linke kattintva digitális formában letölthetjük a könyv további fejezeit, ha sikerült választani a könyv szövegéből véletlenszerűen fellett kérdésekre. (A letölthető fejezetek csak angol nyelven érhetők el – a SZAK Kiadó megjegyzése).

## A könyv forráskódjának igénylése

A könyvben szereplő összes forráskód-mintapélda (az öt szabadon letölthető további fejezetben levők is) szabadon és azonnal elérhető az Apress honlapjának Source Code/Download részlegéről. Egyszerűen gépeljük be a <http://www.apress.com> címet, válasszuk ki a Source Code/Download linket, és keressük meg a megfelelő címet, és, töltsük le az önmagát kicsomagoló \*.zip fájlt. A forráskódokat fejezetenként osztottuk fel.

Figyelem: a fejezetekben a Forráskód megjelölés, mint amilyen az alábbi is, arra utal, hogy a szövegben forró példákat be lehet tölteni a Visual Studio 2008-ba további vizsgálódások és módosítások céljából:

**Forráskód** Ez a forráskód-megjelölés egy kiválasztott könyvtárra hivatkozik.

Egyszerűen nyissuk meg a \*.s1n fájlt a megfelelő alkönyvtárból. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet további integrált fejlesztői környezetekért), akkor manuálisan töltjük be a felkínált forráskódfájlokat a választott fejlesztői eszközbbe.

Megjegyzés A magyar nyelvű változatban a forráskódok megjegyzéseit lefordítottuk. A letölthető forráskódok fordítása nem állt módunkban, így azok teljesen angol nyelvűek. Ilyen módon inkább megfélemlenek a letölthető kódok használati céljának – a SZAK kiadó megjegyzése.

## A lehetséges javítások

Elképzeltethető, hogy a könyv végigolvasása során nyelvtani vagy forráskódbeli hibákat vél felfedezni a kedves Olvasó, bár remélem, hogy erre nem kerül sor. Ha mégis, elnézést kérek. Az aktuális hibalistát az Apress weboldalról lehet igényelni (amely szintén a könyv honlapján található meg), és ugyan-csak itt található az elérhetőségem, amelyen értesíteni tudnak.

## Elérhetőségem

Ha bármilyen kérdés merül fel a könyvhöz tartozó forráskódokkal kapcsolásban, vagy szükség van valamelyik példa tisztázására, vagy egyszerűen csak véleményt szeretne nyilvánítani a .NET platformról, nyugodtan küldjön levelet a következő e-mail címre (hogy biztosak legyünk benne, hogy a levelem kerül a kéretlen levelek közé, kérem, tegye a tárgymezőbe a „C#FE” szöveget): [atroelsen@intertech.com](mailto:atroelsen@intertech.com).

Annak ellenére, hogy megpróbálok minden levélre lehetőség szerint válaszolni, sajnos, ahogy mindnyájan, időnként nagyon elfoglalt vagyok. Ha nem válaszolok egy-két héten belül, akkor ez nem jelenti azt, hogy nem akarok foglalkozni a problémával, csak épp nagyon nem érek rá (vagy ha szeretném van, valahol épp nyaralok).

Nos, akkor, köszönöm, hogy megvásárolta a könyvem (vagy legalábbis, hogy belenéz a könyvesboltban, miközben azon gondolkodik, hogy megvegye-e). Remélem, élvezni fogja a kedves Olvasó a tanulmányozását, és hasznosítani tudja az újonnan szerzett ismereteket.

Minden jót!  
Andrew Troelsen

# 1. rész

## Bevezetés a C#-ba és a .NET platformba



# A .NET filozófiája

Már a programozók számára elengedhetetlen, hogy a néhány évente naprakésznek kell lenniük a legújabb technológiákból. A szoftverfejlesztés gyöngyszemeinek kikialtított nyelveket (C++, Visual Basic 6.0, Java), keretrendszereket (OWL, MFC, ATL, STL), architektúrákat (COM, COBRA, EJB) és API-kat (mint a .NET Windows-úrlapjai és a GDI+ könyvtárai) végül mindezt háttérbe szorítja valami jobb, de legalábbis valami új. Bár a belső adatbázisunk frissítésekor nyilván csaldodottságot érzünk, ám mindez elkerülhetetlen. Eppen ezért könyvünknek az a legfontosabb célja, hogy betekintsünk a Microsoft szoftverfejlesztéssel kapcsolatos aktuális kínálatának részleteibe: a .NET platformba és a C# programozási nyelvbe.

Ebben a fejezetben elsődleges feladatunk az, hogy tisztázzuk a könyv további részeinek megértéséhez szükséges alapfogalmakat. Köztük számos .NET-hez kapcsolódó téma: például a szerelvények, a köztes nyelvi kód (CIL) és a JIT-fordítás tárgyalását is nyomon követhetjük. A C# programozási nyelv legfontosabb sajátosságainak ismertetése mellett a .NET-keretrendszer különböző megjelenési formái – mint a nyelvek közös futtatórendszer (CLR), a közös típusrendszer (CTS) és a közös nyelvi specifikáció (CLS) – közötti létrejövő kapcsolatra is fény derül.

Ezen kívül az olykor BCL-nek, esetleg FCL-nek (keretrendszer-oztálykönyvtár) rövidített .NET-alapoztálykönyvtárak funkcionálitását is megvizsgáljuk. Végezetül áttekinthetjük a .NET platform nyelvagnosztikus és platformfüggetlen természetét is. (Tudniillik a .NET nem korlátozódik a Windows operációs rendszerre.) Ezeket a témákat a továbbiakban részleteseen is kifejtjük.

# Az előzmények áttekintése

1. fejezet: A .NET filozófiája

A .NET univerzum sajátosságainak számbavétele előtt hasznos lehet néhány olyan ténnyel összefüggő megfigyelés, amely a Microsoft jelenlegi platformjának megalkotását előidézte. A tisztánlátáshoz kezdjük a fejezetet egy rövid, „fájdalommentes” történelemmel, hogy felelevenítsük az előzményeket, és megértsük azok korlátait. (Elvège az első lépés a megoldáshoz vezető úton, ha beismerjük, hogy problémával állunk szemben.) Miután tettünk egy gyors körutazást a múltban, figyelmeztünk a C# és a .NET-platform számtalan előnyének a megismerésére fordíthatjuk.

## Egy C/Win32 API programozó élete

Azt szokták mondani, hogy a Windows operációs rendszerek családjába tartozó szoftverek fejlesztése a C programozási nyelv és a Windows alkalmazásprogramozói felületének (API) együttes használatából áll. Igaz, hogy számos alkalmazást sikeresen létrehoztak ezzel a hagyományos módszerrel, néhányan mégis vitatják, hogy az alkalmazások nyers API-val történő felépítése összetett feladat lenne.

Az első nyilvánvaló probléma az, hogy a C nagyon tömör nyelv. A C# fejlesztői kénytelenek kézi memóriakezeléssel, csúnya pointeraritmetikával és ugyancsak csúnya szintaktikai konstrukciókkal megküzdenni. Ráadásul, mivel a C strukturált nyelv, hiányzik belőle az objektumorientált megoldások előnye (valaki el tudja mondani, mi az a *spagettikód*?). Amikor a Win32 API segítségével vel definiált globális függvények és adattypusok ezreit hozzáadjuk egy eleve túl részletes nyelvhez, nem csoda, hogy olyan sok hibás alkalmazást kapunk.

## Egy C++/MFC-programozó élete

A csupasz C/API fejlesztésének egyik nagy horderejű újítása a C++ programozási nyelv használata volt. A C++ sok szempontból a C-re helyezett objektumorientált *rétegnek* tekinthető. Így annak ellenére, hogy a C++ programozónak használna vannak az objektumorientált programozás közszerit alapjait (egységbezárás, számozás és polimorfizmus), továbbra is ki vannak szolgáltatva a C nyelv nehézségeinek (pl. kézi memóriakezelés, csúnya pointeraritmetika és csúnya szintaktikai konstrukciók).

Összetettégre ellenére, manapság is sok C++-keretrendszer létezik. Például a Microsoft Foundation Classes (MFC) olyan C++-osztály-készletet biztosítanak a fejlesztők számára, amely megkönnyíti a Win32-alkalmazások felépítését. Az MFC legfőbb szerepe, hogy számos osztály, makró és kódgeneráló eszköz (varázsló) mögött a natív Win32 API egy részét beburkolja. Az MFC-keretrendszer hasznos támogatását figyelmen kívül hagyjuk (ugyanígy, mint sok egyéb C++-alapú eszközrendszer), be kell látnunk, hogy a C++-programozás nehézkes, és több hibalehetőséget rejt magában, tekintve, hogy a C-ből ered.

## Egy Visual Basic 6.0 programozó élete

Az egyszerűsége törekvés jegyében sok programozó a C(++)-alapú keretrendszerektől olyan barátságosabb és kifinomultabb nyelvekre tért át, mint például a Visual Basic 6.0 (VB6). A VB6 annak köszönheti népszerűségét, hogy egyszerűen lehet vele összetett felhasználói felületeket, kódkönyvtárakat (pl. COM-kiszolgálók) és adatelérési logikát készíteni. A VB6 sokkal jobban elrejti szem elől a csupasz Win32 API bonyolultságát, mint az MFC, számos beépített forráskód-varázsló, valódi adattípusok, osztályok és VB-specifikus függvények segítségével.

A VB6 legnagyobb hibája (amelyet a .NET-platform megjelenése orvosit), hogy nem teljesen objektumorientált nyelv, hanem inkább „objektumtudas”. A VB6 például nem engedi a programozónak, hogy „ez egy...” („is-a”) kapcsolatot hozzanak létre a típusok között (azaz nem engedi a klasszikus számtattást), és nincs belső támogatása paraméterezett osztályok készítéséhez. Ezenkívül a VB6 nem képes többszálú alkalmazások létrehozására sem, kivéve, ha alacsony szintű Win32 API hívásokat szeretnénk (am ez legjobban esetben össze-tett, legrosszabb esetben pedig veszélyes).

## Egy Java/J2EE-programozó élete

A Java megérkezett. A Java objektumorientált programozási nyelv, amelynek szintaktikai gyökerei a C++-ból erednek. Amint azzal sokan tisztában vannak, a Java előnyei messze túlszárnyalják platformfüggetlenségét támogatását. A Java (mint nyelv) a C++ számos kellemetlen szintaktikai jellemvonását kiküszöbölte. A Java (mint platform) a programozók számára számtalan előre-megadott „csomagot” biztosít, amelyek különféle típusdefiniciókat tartal-

maznak. Ezekkel a típusokkal a Java-programozók képesek 100%-ban tisztázni a Java-alkalmazásokat létrehozni, amelyek teljes adatbázis-összekapcsolhatósággal, üzenetközzvetítési támogatással, webes ügyféloldallal és gazdag asztali felhasználati felülettel rendelkeznek.

Habár a Java nagyon elegáns nyelv, potenciális problémája, hogy használatakor jellemzően előlrol hátrafele kell haladni a fejlesztés folyamán. A Java tulajdonképpen kismértékben újít nyelvi integrációt, minthogy ez ellentmond elsödleleges feladatának – a Java egy egyszerű, bármilyen célra felhasználható programozási nyelv. A valóságban azonban forráskódok milliónyi sora létezik, s ezeket előnyös lenne Java-kódból felhasználni. Sajnos a problémamegoldását a Java nem könnyíti meg. Amíg ugyanis korlátozott képesseget mutat a nem Java-s API-k eléréséhez, addig csak kismértékben támogatja a valódi nyelvközi integrációt.

## Egy COM-fejlesztö élete

A Component Object Model (COM – komponensobjektum-model) volt a Microsoft előző alkalmazásfejlesztési keretrendszer. A COM architektúrája lényegében azt fejezi ki, hogy „[h]a az osztályainkat a COM szabályai szerint építjük fel, *újrafelhasználható bináris kódblokkok* kapunk”.

Egy bináris COM-kiszolgálóban az a nagyszerű, hogy nyelvfüggetlen módon hozzá lehet fenni. Így a C++-programozók olyan COM-osztályokat hozhatnak létre, amelyeket a VB6 felhasználhat. A Delphi-programozók használhatnak olyan COM-osztályokat, amelyeket C segítségével hoztak létre, és így tovább. A COM nyelvfüggetlensége azonban némileg korlátozott. Egy meglevő COM-osztályból például nem lehet újabbat származtatni (mivel a COM nem támogatja a klasszikus származtatást). Ehelyett, a sokkal nehezebb „van neki egy...” („has-a”) kapcsolat segítségével kell újra felhasználni a COM-osztály típusait.

A COM másik előnye az, hogy transzparens tárhellyel rendelkezik. Az olyan szerkezetek használataival, mint az alkalmazásazonosító (AppIDs), a csomk, a proxyk és a COM futási környezet, a programozóknak nem kell sokketeikkel, távoli eljáráshtívásokkal és egyéb alacsony szintű részletekkel dolgozniuk. Nézzük meg például a VB6-COM-ügyfélkódot:



```
' A myCOMClass típus bármely
' COM-alapú nyelven megírt, és a hálózaton bárhol
' elérhető (helyi számítógépen is).
Dim obj as myCOMClass
Set obj = New myCOMClass
obj.DoSomeWork
```

Habár a COM-ot nagyon sikeres objektummodellnek is tarthatjuk, valójában elképesztően bonyolult (legalábbis addig, amíg nem fordítunk több hónapot az átkötések vizsgálatára – elsősorban C++-programozóként). A COM-binárisok fejlesztésének egyszerűsítésére számos COM-alapú keretrendszer létezik. Ilyen például az ATL (Active Template Library), amely olyan C++-osztályok, sablonok és -makrók készletével szolgál, amelyek megkönnyítik a COM-típusok létrehozását.

Sok egyéb nyelv ugyancsak elérhető szem elől a COM-infrastruktúra nagy részét. A nyelvi támogatás azonban önmagában nem elég a COM komplexitásának elrejtéséhez. Még ha egy olyan viszonylag egyszerű COM-alapú nyelvet választunk, mint a VB6, akkor is meg kell küzdenünk a sérülékeny regisztrációs bejegyzésekkel és számos telepítéssel kapcsolatos problémával (gyújtónéven, némileg komikusan, *DLL-pokol*).

## Egy Windows DNA programozó élete

Tovább bonyolítja a helyzetet az internet. Az elmúlt néhány évben a Microsoft jó néhány további internetalapú szolgáltatást épített be az operációs rendszer-családjába és termékeibe. Sajnos a COM-alapú Windows DNA segítségével történő webes alkalmazások létrehozása ugyancsak meglehetősen bonyolult.

Ez a bonyolultság részben annak a ténynek köszönhető, hogy a Windows DNA számos technológia és nyelv használatát igényli (ASP, HTML, XML, JScript, VBScript és COM[+], valamint olyan adatelérési API-t, mint az ADO). Problémát jelent, hogy szintaktikai szempontból a technológiák nagy része nem kapcsolódik egymáshoz. A JScript szintaxisa például sokban hasonlít a C-re, míg a VBScript a VB6 részhalmoz. A COM+-futtatókörnyezetre tervezett COM-kiszolgálók teljesen eltérő megjelölések, mint azok az ASP-oidalak, amelyek hívják őket. Mindez a technológiák teljesen zavaros keverékét eredményezi.

- *Alfogó alapoztállykönyvtár.* Ez a könyvtár teszi lehetővé a natív API-hi-vások komplexitásának elkerülését, és olyan konzisztens objektum-modellt kínál, amelyet az összes .NET-alapú nyelv használ.
- *Közös futtatórendszer az összes .NET-alapú nyelv számára.* Ez a rendszer definiál egy olyan típuskészletet, amelyet minden .NET-alapú nyelv megért.
- *Teljes nyelvi integráció.* A .NET támogatja a nyelvközi származtatást, a nyelvközi kivételkezelést és a nyelvközi hibakeresést.
- *Együttműködési képesség a meglévő forráskóddal.* Ez (természetesen) na-gyon hasznos. A meglévő COM-binárisok keverednek (azaz együtt-működnek) az újabb .NET bináris fájlokkal, és viszont. A PInvoke le-hetővé teszi a C-alapú könyvtárak (beleértve az operációs rendszer API) hívását a .NET-kódból.

Az előző rövid történelemóra tanulsága az volt, hogy a Windows-programo-zók élete bizony nem egyszerű. A .NET-keretrendszer azonban radikálisan egyszerűsíti és megkönnyíti a programozó életét. A .NET a „Változtassunk meg mindent!” megoldást kínálja („elnevezést kérek, de ne az üzenet közvetítőjét okoljuk az üzenetet”). Ahogy a könyv további részeiben látni fogjuk, a .NET-keretrendszer a rendszerfelépítés teljesen új modellje a Windows operációs rendszercsaládban éppúgy, mint számos nem Microsoft-alapú operációs rendszemben (mint például a Mac OS X és a különféle Unix-/Linux-termékek). Elő-zetesen íme a .NET néhány alapvető jellemzőjének összefoglalása:

## A.NET-megoldás

Ezenkívül, és ez talán sokkal fontosabb, minden nyelvnek és/vagy tech-nológiának megvan a saját típusrendszere (és ez lehet, hogy semmiben sem hasonlít egy másikéra). Túl azon, hogy minden egyes API rendelkezik saját előre gyártott kódgyűjteménnyel, az alapvető adattípusokat nem lehet min-dig ugyanúgy kezelni. A COMBSTR az ATL-ben nem teljesen ugyanaz, mint egy string a VB6-ban; egyiknek sincs semmi köze a C char\* adattípusához.

- *Nincs több COM-átkötés.* Az `IClassFactory`-nak, az `IDis-`patchnek, az `IDL`-kód és a nem `til` szimpatikus `variant`-kompatibilis adattípusoknak (`BSTR`, `SAFEARRAY` stb.) nincs helyük egyetlen .NET-bi-nárisban sem.

- *Egy valóban leegyszerűsített telepítési modell.* .NET alatt nem szükséges bejegyezni a bináris egységeket a rendszerleíró adatbázisba. Ráadásul a .NET lehetővé teszi, hogy egyetlen gépen ugyanazon .dll több ver-ziója legyenek jelen.

Ahogy a fenti felsorolásból látszik, a .NET-platformnak semmi köze a COM-hoz (azonfelül, hogy mindkét keretrendszer a Microsofttól származik). A .NET- és a COM-típusok valójában csak az együttműködési réteg segítségével tudnak kommunikálni.

---

**Megjegyzés** A .NET együttműködési réteg leírása a 2 kötet A függelékekben található.

---

## Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS)

A .NET néhány előnyét már megismerve nézzük át azt a három alapvető, egymással összefüggő egységet, amelyek mindezt lehetővé teszik: a CLR-t, a CTS-t és a CLS-t. Egy programozó szemszögéből a .NET felfogható futtató-környezetként és átfogó alapozztálkőnyvtárként. A futtató réteg más néven *közös nyelvi futtatórendszer*, vagyis a CLR. A CLR elsődleges feladata, hogy megkeresse, betöltse és kezelje helyettünk a .NET-típusokat. Ezenkívül olyan alacsony szintű részletekre is figyel, mint a memóriakezelés, alkalmazástár-tományok létrehozása, végrehajtási szálak és objektumkontextus-határok, va-lamint különböző biztonsági ellenőrzések végrehajtása.

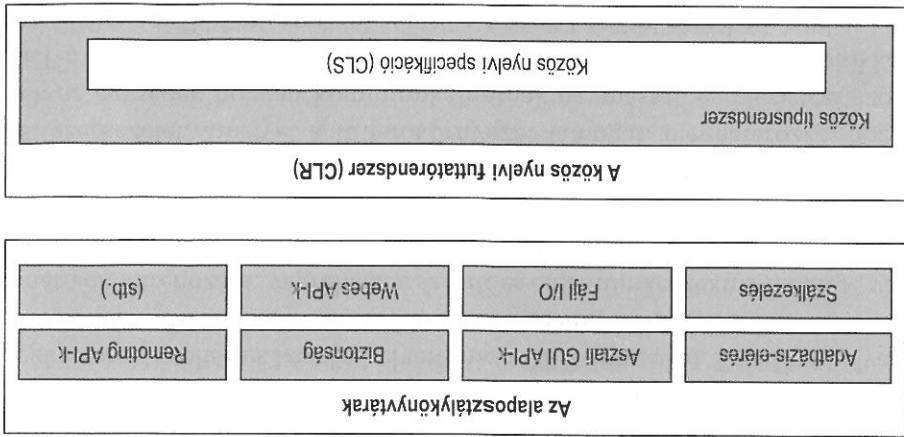
A .NET-platform másik építőeleme a *közös típusrendszer*, azaz a CTS. A CTS-specifikáció teljes körűen leírja az összes lehetséges adattípust és programo-zási szerkezetet, amelyet a futtatókörnyezet támogat, meghatározza, hogy ezek az egységek hogyan kommunikálhatnak egymással, és részletezi azok .NET-metadatumban való megjelenésének módját (a metaadatokkal kapcsolatos további információkat lásd a fejezet későbbi részében; a teljes kö-rű leírást pedig lásd a 16. fejezetben).

Egy adott .NET-alapú nyelv nem szükségeszerűen támogat minden egyes CTS-ben definiált funkciót. A *közös nyelvi specifikáció* (CLS) egy olyan kapcsolódó specifikáció, amely definiálja azoknak a közös típusoknak és programozási szerkezeteknek a részhalmozát, amelyeket minden .NET programozási nyelv elfogad. Ezért, ha olyan .NET-típusokat építünk fel, amelyek csak CLS-kompatibilis funkciókkal rendelkeznek, nyugodtak lehetünk afelel, hogy az összes .NET-alapú nyelv használni tudja azokat. Ellenben, ha olyan adat-típus vagy programozási szerkezetet használunk, amely a CLS határain kívül esik, nem biztos, hogy az összes .NET programozási nyelv képes lesz kommunikálni a .NET-kódkönyvtárunkkal.

## Az alapsztálykönyvtárak szerepe

A CLR- és CTS/CLS-specifikációkon kívül, a .NET-platform az összes .NET programozási nyelv számára biztosít egy alapsztálykönyvtárat. Ez az alapsztálykönyvtár nemcsak beágyaz különböző primitíveket, mint a végrehajtási szálak, fájl input/output (I/O), grafikus renderelés és különféle külső hardvereszközökkel történő kommunikáció, de számos, leginkább az élet-szerű alkalmazások igényeihez mért szolgáltatást is támogat.

Az alapsztálykönyvtárak például olyan típusokat definiálnak, amelyek megkönnyítik az adatbázis-hozzáférést, az XML-dokumentumok manipulációját, a programozott adatvédelmet és a webes (csakúgy, mint a hagyományos asztali és konzolos) front endek létrehozását. A CLR, CTS, CLS és az alapsztálykönyvtár közötti kapcsolatot, az 1.1. ábra szemlélteti.



1.1. ábra: A CLR, CTS, CLS és az alapsztálykönyvtár kapcsolata

## A C# előnye!

Mivel a .NET radikálisan eltér a korábbi technológiáktól, a Microsoft kifejezetten ehhez az új platformhoz hozott létre egy új programozási nyelvet, a C#-t (kifejtve: [si: sa:p]). A C# alapszintaxisa *nagyon* hasonlít a Java szintaxisához. A C# azonban nem a Java egyik módosulata. Mind a C#, mind a Java a C programozási nyelv családjába (C, Objective C, C++ stb.) tartozik, ezért hasonló szintaxisal rendelkeznek. Ahogy a Java sok tekintetben a C++ letisztult verziója, a C#-ot ugyanígy a Java letisztult verziójának tekinthetjük.

Ettől függetlenül az igazság az, hogy a C# szintaktikai felépítése nagyrészt a Visual Basic 6.0 és a C++ alapján készült. Ahogy például a VB6, a C# is támogatja a tulajdonságok létrehozását (ellentétben a tradicionális getter és setter metódusokkal) és a metódusok deklarálásának képességét változó számú argumentummal (argumentumtömbök révén). A C++-hoz hasonlóan, a C# lehetővé teszi az operátorok túlterhelését, valamint strukturák, felsorolt típusok és visszahívható függvények létrehozását (metódusreferenciák révén).

**Megjegyzés** Ahogy a 13. fejezetben látni fogjuk, a C# 2008 számos olyan konstrukciót átvett, amelyek eredetileg különböző funkcionális nyelvekből (pl. LISP vagy Haskell) származnak. Ráadásul, a LINQ megjelenésével (további információkért lásd a 14. és a 11. kötetben található 24. fejezetet) a C# több olyan programozási konstrukciót támogat, amelyek meglehetősen egyedivé teszik a programnyelvek palettáján. Mindamellelt problémára a C#-pal, hogy valóban hatással voltak rá a C-alapú nyelvek.

Mivel a C# több nyelv keresztezéséből született, szintaktikailag legalább annyira, ha nem még inkább lehszultnak tekinthető, mint a Java, olyan egyszerű, mint a VB6, és annyira hatékony és rugalmas, mint a C++ (a vele járó apró csúnyaságok nélkül). Íme, egy lista a C# alapvető tulajdonságairól, amelyek a nyelv valamennyi verzióját jellemzik:

- Nincs szükség mutatókra. A C#-programoknak jellemzően nincs szükségük közvetlen mutatómanipulálásra (de, ha mindenképpen szükség van, le lehet erre a szintre menni).
- Automatikus memóriakezelés a szemétyűjtésen keresztül. Így a C# nem támogatja a delete kulcsszót.
- Formális szintaktikai konstrukcióként megjelenik az osztály, az interfész, a struktúra, a felsorolás és a metódusreferenciák.

- A C++-hoz hasonló operátor-tulterhelés az egyedi típusok esetében, csak egyszerűbben (pl. a meggyőződés arról, hogy a „return \*this to allow chaining” nem a mi problémánk).
  - Az attribútumalapú programozás támogatása. Az ilyen típusú fejlesztés lehetővé teszi, hogy a típusokat és a tagjaikat magyarázatokkal lásdunk el viselkedésük további leírásához.
- A .NET 2.0 verziójával (kb. 2005-től) a C# programozási nyelv számos újdonsággal frissült, amelyek közül az alábbiak a legfontosabbak:

- Generikus típusok és tagok felépítésének a képessége. A generikus típusok használatával rendkívül hatékony és típusbiztos forráskódokat építhetünk fel, amelyek a generikus elemmel való kommunikáció során meghatározott helyőrzőket definiálnak.
- Névtelen metódusok támogatása, amelyek lehetővé teszik, hogy bárhol, ahol metódusreferenciára van szükség, inline függvényeket alkalmazjunk.
- A metódusreferencia/esemény modell számtalan egyszerűsítése, köztük kovariancia, kontravariancia és metóduscsoport-átalakítás.
- Egyetlen típus több kódájlion keresztül definiálásának képessége (vagy ha szükséges, memórián belüli ábrázolása) a partiai kulcsszó segítségével.

- A .NET 3.5 egyértelműen még több funkcionálitással látta el a C# programozási nyelvet (pontosabban a C# 2008 verziót), köztük az alábbiakkal:
  - A különféle adatformátumokkal való kommunikációhoz szükséges erősen típusos leképezések (LINQ) támogatása.
  - Névtelen típusok támogatása, amellyel sokkal inkább a típusok „alakját”, mint viselkedését modellezhetjük.
  - Létező típusok funkcionálitásának kiterjesztése, bővítő metódusok segítségével.
  - A lambda operátor (=>) bevezetése, amely még inkább leegyszerűsíti a .NET-metódusreferencia-típussal végzett munkát.
  - Új objektuminitializáló szintaxis, amely lehetővé teszi a tulajdonságértékek beállítását az objektum létrehozásakor.

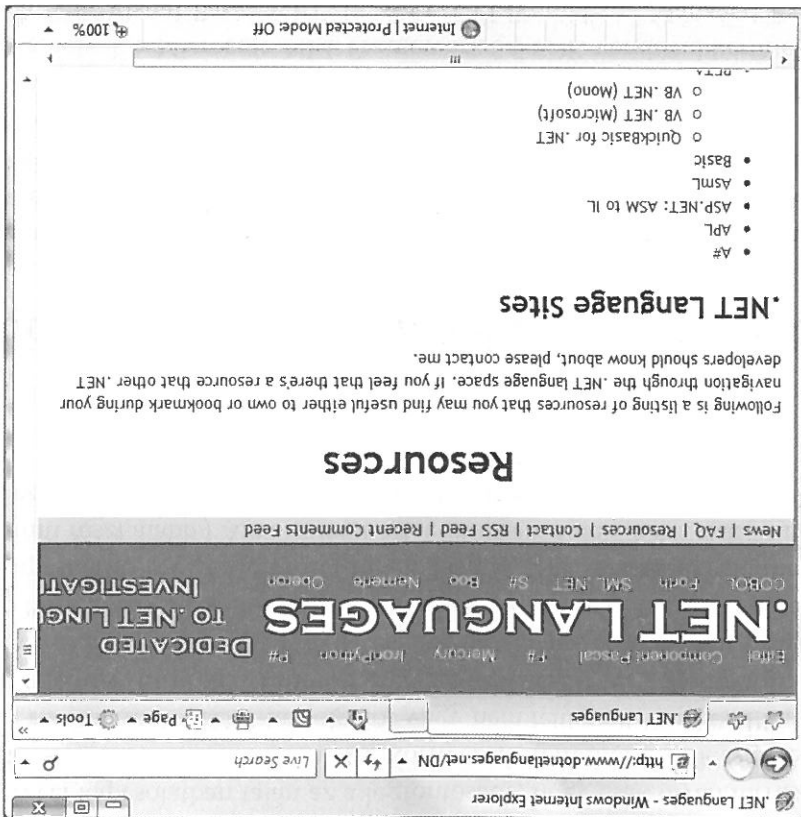
A C# nyelvvel kapcsolatban talán az a legfontosabb, hogy csak olyan forráskódot tud létrehozni, amely .NET-futtatókörnyezetben működik (C# segítségével nem tudunk natív COM-kiszolgálót vagy nem felügyelt Win32 API alkalmazást felépíteni). A .NET-futtatókörnyezethez készült forráskód leírására hivatalosan a *felügyelt programkód* kifejezést használjuk. A felügyelt programkódot tartalmazó bináris egységet *szervetvénynek* nevezzük. (A szerelevényekkel kapcsolatos további információkért lásd a „*.NET-szerelevények áttekintése*” című részt alább.) Azt a forráskódot ellenben, amelyiket a .NET-futtatókörnyezet közvetlenül nem tudja hosztolni, nem *felügyelt programkódnak* nevezzük.

.NET-alkalmazásokat nem csak a C# nyelv segítségével építhetünk fel. Amikor 2000-ben, a Microsoft fejlesztői konferenciáján először bemutatták a nagykülönbségnek a .NET-platformot, több gyártó is bejelentette, hogy felépítik a saját fordítóprogramjuk .NET-alapú verzióját. Azóta nyelvek tucatjai estek át a .NET-felvilágosodáson. Azon az öt nyelven kívül, amelyek a Microsoft .NET Framework 3.5 SDK verzióval kerültek forgalomba (C#, Visual Basic .NET, J#, C++/CLI [korábban „Managed Extensions for C++”] és JavaScript .NET), léteznek .NET-fordítóprogramok Smalltalkra, COBOL-ra és Pascalra is. Jóllehet könyvünk (szinte) kizárólag csak a C#-pal foglalkozik, érdemes ellátogatni a következő webhelyre (a Microsoft fenntartja az URL változtatásának jogát):

<http://www.dotnetlanguges.net>.

Ha a fődolai tetején lévő Resources hivatkozásra kattintunk, számtalan .NET programozási nyelvet és azokhoz kapcsolódó hivatkozásokat tartalmazó listát kapunk, ezekre kattintva pedig különféle fordítókat tölthetünk le (lásd 1.2. ábra).

Ha elsősorban a C# szintaxisa alapján szeretnénk .NET-programokat létrehozni, tanácsos felkeresni ezt a webhelyet, itt biztosan találunk ugyanis a célunknak megfelelő .NET-nyelvet (esetleg a LISP .NET-et?).



1.2. ábra: A *www.DotNetLanguges.net* az egyik olyan webhely, amely ismert .NET programozási nyelveket dokumentál

## Elet egy többnyelvű világban

Amikor a fejlesztők ráébredtek a .NET nyelvi agnosztikai természetére, számos kérdés merült fel. Talán az egyik leggyakoribb kérdés ez lehetett: „Ha az összes .NET-nyelv felügyelt programkódot fordít, miért van szükség több fordítóra?”. Ezt a kérdést többféleképpen meg lehet válaszolni. Először is, mi programozók *nagyon* valogatósak vagyunk, ha bármilyen programozási nyelv kiválasztásáról van szó. Néhanyan a pontosvesszős és kapcsos zárójeles nyelveket szeretik, a lehető legkevesebb nyelvi kulcsszóval. Mások azokat a nyelveket szeretik, amelyek az ember számára olvashatóbb szintaktikai tokeneket tartalmaznak (mint a Visual Basic). Megint mások pedig a nagy-számítógépes tapasztalataikat szeretnék érvényesíteni, amíg áttérnek a .NET-platformra (a COBOL .NET-en keresztül).



Legyünk őszinték. Ha a Microsoft létrehozott volna egy „hivatalos” .NET-nyelvet, amely a BASIC nyelvcsaládból származik, vajon az összes programozó elégedett volna? Vagy, ha az egyetlen „hivatalos” .NET-nyelv a Fortran szintaxisán alapulna, nyilván sokan lennének, akik a .NET-et végképp semmibe vennék. Mivel a .NET-futtatókörnyezetben nem fontos, hogy milyen nyelvi segítségével építették fel a felülgylet kódot, a .NET-programozók hűek maradhatnak ahhoz a szintaxisához, amelyet korábban választottak, és a lefordított szerelevényeket probléma nélkül megoszthatják a csapattagokkal, a részlegekkel és a külső szervezetekkel (tekintet nélkül arra, hogy a másik felmilyen .NET-nyelvet használ).

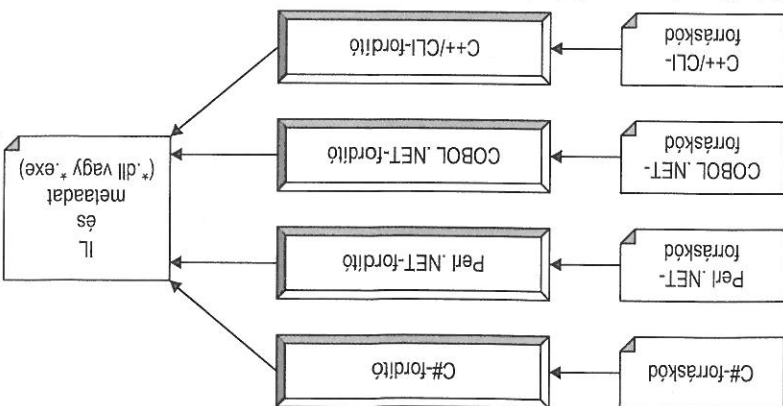
A különféle .NET-nyelveknek egyetlen, egységes szoftvermegoldásba történt integrálásának másik előnye, hogy minden egyes programozási nyelvnek megvan a maga erősségei és gyengeségei. Egyes programozási nyelvek például remekül támogatják a fejlett matematikai feldolgozást. Mások pedig elsősorban a pénzügyi számításokat, a logikai számításokat, a nagy-számtíógépekkel való kommunikációt stb. Ha az egyik programozási nyelv erősségeit vesszük, majd azokat egyesítjük a .NET-plattform előnyeivel, voltaképpen mindenki jól jár.

A valóságban persze elég nagy az esély arra, hogy a szoftverek felépítését a kiválasztott .NET-nyelv segítségével fogjuk végezn. Ha azonban egy .NET-nyelv szintaxisát megismerjük, nagyon könnyű lesz egy másikat is meg tanulni. Ez szintén meglehetősen hasznosnak tűnik, különösen a szoftvertanácsadókna. Ha a választott nyelvünk például a C#, de egy olyan ügyfélnel dolgozunk, ahol csak Visual Basic .NET-et használhatunk, a .NET-keretrendszert funkcionálitását így is kihasználhatjuk, és az alapkod általános szerkezetét nagyobb nehézségek nélkül megérthetjük.

## A .NET-szerelevények áttekintése

Függetlenül attól, melyik .NET-nyelven programozunk, fogadjuk el, hogy noha a .NET bináris fájlok kiterjesztései megegyeznek a COM-kiszolgálók és a nem felülgylet Win32 bináris fájlok kiterjesztéseivel (\* .dll vagy \* .exe), nincs bennük semmiféle belső hasonlóság. A \* .dll .NET bináris fájlok például nem exportálnak metódusokat a COM-futtatókörnyezettel való kommunikációhoz (mivel a .NET *nem* COM). Továbbá a .NET bináris fájlok nem használnak COM-típuskönyvtárakat, és nincsenek bejegyzve a rendszerleíró adatbázisba.

De talán a legfontosabb, hogy a .NET bináris fájlok platformfüggetlen utasítások helyett platformfüggően köztes nyelvet (IL) és típusmetadátákat tartalmaznak. Az 1.3. ábra az eddigieket mutatja be.



1.3. ábra: Az összes .NET-alapú fordító IL-utasításokat és metaadatokat ad ki

**Megjegyzés** Az „IL” rövidítést illetően meg kell említeni, hogy a .NET kifejlesztése során az IL helyett hivatalosan a Microsoft köztes nyelvé (MSIL) kifejezést használták. A .NET végleges kiadásakor azonban már közös köztes nyelvként (CIL) említették. Ebből következően a .NET-irodalomban az IL, az MSIL és a CIL tehát pontosan ugyanarra utal. Az aktuális terminológiát követve a továbbiakban a „CIL” rövidítést fogjuk alkalmazni.

Amikor .NET-alapú fordító segítségével hozunk létre egy \*.dll vagy egy \*.exe kiterjesztésű fájlt, az eredményül kapott modul *szerelvénny*l alkot. (A .NET-szerelvényeket részletesen a 15. fejezetben tárgyaljuk.) A .NET futtató környezet részletes tárgyalása előtt meg kell ismerkednünk az új fájlformátum néhány alapvető tulajdonságával.

A szerelvények tehát köztes nyelvi kódokat tartalmaznak, amelyek elvben a Java-bájtíkhöz hasonlítanak, hiszen addig nem változnak platformfüggetlenségi utasításra, amíg ez nem feltétlenül szükséges. A „feltétlenül szükséges” pont jellemzően az, amikor a .NET-futtatókörnyezet a CIL-utasításblokk (például egy metódusimplementáció) használatára hivatkozik.

A CIL-utasításokon kívül a szerelvények *metaadatokat* is tartalmaznak, amelyek a binárisban található típusok karakterisztikáit részletezik. Ha van például egy sportscar nevű osztályunk, a típusmetadát olyan részleteket ír le, mint a sportscar össztálya, amelynek interfészét (ha létezik) a sportscar implementálja, valamint az összes olyan tagot, amelyeket a sportscar típus támogat.

A .NET-metaadatok hatalmas előrelépést jelentenek a COM-metaadatokhoz képest. Talán már közismert, hogy a COM-binárisok jellemzően hozzárendelt típuskönyvtárakat használnak (ez valamivel több, mint az interfészleíró nyelv [IDL] forráskódjának bináris verziója). A COM-típusinformációkkal az a probléma, hogy nem biztos, hogy megvan, illetve, hogy az IDL-kód nem tudja dokumentálni a kívülről hivatkozott kiszolgálókat, amelyek az aktuális COM-kiszolgáló megfelelő működéséhez szükségesek. Ezzel szemben a .NET-metaadatok mindig jelen vannak, és az adott .NET-alapú fordító automatikusan generálja őket.

Végezetül, a CIL és a típusmetaadatokon kívül maguk a szerelvények is használnak metaadatot, amelyet *manifestum*nak nevezünk. A manifestum a szerelvény aktuális változatáról szóló és kulturális információkat tartalmaz (a sztring- és a képerőforrások lokalizációjához), illetve egy listát, amely a megfelelő végrehajtáshoz szükséges összes külsőleg hivatkozott szerelvényt felsorolja. A következő néhány fejezetben a szerelvények típusainak, metaadatainak és önléíró adatainak vizsgálatához szükséges különféle eszközöket ismerhetjük meg.

## Egyfajlos és többfajlos szerelvények

Az esetek nagy többségében egyszerű egy-egy kapcsolat van a .NET-szerelvény és a bináris fájl (\* .dll vagy \* .exe) között. Ebből következik, ha egy .NET \* .dll létrehozásán dolgozunk, úgy vehetjük, hogy a bináris és a szerelvény egy és ugyanaz. Hasonlóképpen, ha egy végrehajtható asztali alkalmazást készítünk, az \* .exe fájlt egyszerűen tekinthetjük magának a szerelvénynek. (Később azonban, a 15. fejezetben majd kiderül, hogy ez így nem pontos.) Szaknyelven, ha a szerelvény egy \* .dll vagy egy \* .exe modulból áll, azt *egyfajlos szerelvénynek* nevezzük. Az egyfajlos szerelvények az összes szükséges CIL-t, metaadatot és hozzárendelt manifestumot önálló, egyedi és pontosan definiált csomagban tartalmazzák.

A *többfajlos szerelvények* viszont számos .NET bináris fájlból állnak, amelyeket *moduloknak* nevezünk. A többfajlos szerelvény esetében az egyik modul (*elsődleges modul*) tartalmazza a szerelvény menifesztumát (esetleg CIL-tútasításokat és különféle típusok metaadatait). A többi modul egy modul szintű menifesztummal, CIL-tútasításokkal és típusmetaadatokkal rendelkezik. Nyilvánvaló, hogy az elsődleges modul dokumentálja a szükséges másodlagos modulok csoportját a szerelvény menifesztumában.

Miért hozzunk létre többfajlos szereplvényeket? Ha egy szereplvényt különálló modulokra osztunk fel, jóval rugalmasabb telepítési beállításokhoz jutunk. Ha például a felhasználó egy olyan távoli szereplvényre hivatkozik, amelyet szeretne letölteni a számítógépére, a futtatókörnyezet csak a szükséges modulokat fogja letölteni. Ezért felelősségünk úgy is a szereplvényünket, hogy azokat a típusokat, amelyekre ritkábban van szükség (mint például a hardvereket nevé osztály), külön modul tárolja.

Ellenben, ha az összes típust egyfajlos szereplvénybe helyezzük, a végfelhasználó jó néhány felesleges adatot fog letölteni (és ez nyilvánvalóan időpocsékolás). Egy szereplvény tehát valóban egy vagy több kapcsolódó modul *logikai csoportosítását* tartalmazza, amelyeket önálló egységként telepítünk és módosítunk.

## A közös köztes nyelv (CIL) szerepe

A következőkben vizsgáljuk meg kicsit közelebbről a köztes nyelvi kódot, a tpusmetaadatot és a szereplvény-manifesztumot. A CIL olyan nyelv, amely bármilyen platformfüggő gépiutasítás-készlet felett áll. A következő C#-kód például egy hétköznapi számológépet modellez. Ne foglalkozzunk most a pontos szintaxissal, de figyeljünk a `Calc` osztály `Add()` módszerének formátumára:

```
// Calc.cs
using System;

namespace CalculatorExample
{
    // Az osztály tartalmazza az alkalmazás belépési pontját.
    class Program
    {
        static void Main()
        {
            Calc c = new Calc();
            int ans = c.Add(10, 84);
            Console.WriteLine("10 + 84 is {0}.", ans);
        }
    }
    // Várakozás, amíg a felhasználó megnyomja az Enter
    // billentyűt.
    Console.ReadLine();
}
```

```
// C#-számológép.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
```

Ha ezt a kódfájlt a C#-fordítóval (csc.exe) lefordítjuk, olyan egyfájlos \*.exe szerelvényt kapunk, amely tartalmaz manifestumot, CIL-utasításokat és metaadatokat, és ezek minden szempontból leírják a Calc és a Program osztályt.

---

**Megjegyzés** A C#-fordító segítségével történő kódfordítást, illetve a grafikus IDE-k, például a Visual Studio, a Visual C# Express és a SharpDevelop, használatát a 2. fejezetben vizsgáljuk meg részletesen.

---

Ha ezt a szerelvényt például az t1dasm.exe segítségével szeretnénk megnyitni (lásd alább), láthatjuk, hogy az Add() metódus köztes nyelven, a következőképpen jelenik meg:

```
.method public hidebysig instance int32 Add(int32 x,
int32 y) cil managed
{
    // A kód mérete 9 (0x9)
    .maxstack 2
    .locals init (int32 V_0)
    IL_0000: nop
    IL_0001: ldarg.1
    IL_0002: ldarg.2
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007
    IL_0007: ldloc.0
    IL_0008: ret
} // a Calc::Add metódus vége
```

(Az eredményül kapott metódust a 19. fejezetben találjuk a CIL programozási nyelv alapjainak tárgyalásáról szóló részben.) A hangsúly most azon van, hogy a C#-fordító nem platformfüggető gépi utasításokat ad ki, hanem CIL-re fordít. Mindez, ahogy láttuk, az összes .NET-alapú fordítóra igaz. Ennek illusztrálására, tétellezzük fel, hogy ugyanezt az alkalmazást C# helyett Visual Basic .NET segítségével is létrehoztuk:

```

' Calc.vb
Imports System

Namespace CalculatorExample
    ' A VB "Module" is a class that contains only
    ' static members.
    Module Program
        Sub Main()
            Dim c As New Calc
            Dim ans As Integer = c.Add(10, 84)
            Console.WriteLine("10 + 84 is {0}.", ans)
            Console.ReadLine()
        End Sub
    End Module

    Class Calc
        Public Function Add(ByVal x As Integer, ByVal y As Integer)
            As Integer
                Return x + y
            End Function
        End Class
    End Namespace

```

Ha megnezzük az Add() metódus köztes nyelvét, ott hasonló utasításokat találhatunk (a VB .NET-fordító [vbc.exe] által optimalizálva):

```

.method public instance int32 Add(int32 x, int32 y) cil managed
{
    // A kód mérete 8 (0x8)
    .maxstack 2
    .locals init (int32 V_0)
    IL_0000: ldarg.1
    IL_0001: ldarg.2
    IL_0002: add.ovf
    IL_0003: stloc.0
    IL_0004: br.s IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // a calc::Add metódus vége

```

---

**Forráskód A Calc.cs és a Calc.vb kódfájlokat az 1. fejezet alkönyvtára tartalmazza.**

---

## A köztes nyelv előnye!

Adódik a kérdés: miért volt jó, hogy köztes nyelvre fordítottuk a forráskódot, és nem közvetlenül egy specifikus utasításkészletbe. Ennek egyik oka a nyelv-vi integráció. Ahogy már láttuk, minden egyes .NET-alapú fordító közel azonos CIL-utasításokat hoz létre. Így az összes nyelv egy jól meghatározott bináris térben kommunikálhat egymással.

Ráadásul, mivel a CIL platformfüggetlen, és maga a .NET-keretrendszer is az, így a Java-fejlesztők számára már jól ismert előnyöket biztosítja (vagyis egy szimpla kódبázis fut több operációs rendszeren). A C# nyelvre létezik nemzeiközi szabvány , és a .NET-platform, valamint implementációinak a nagy része már több nem Windows operációs rendszeren is rendelkezésre áll (további információkért lásd a fejezet késöbbi részét). Ekkéntben a javával, a .NET lehetővé teszi, hogy a kívánt nyelven hozzunk létre alkalmazásokat.

## A köztes nyelv fordítása platformfügö utasítássá

Mivel a szerelevények inkább CIL-utasításokat, mintsem platformfügö utasításokat tartalmaznak, a köztes nyelv kódot használat előtt, menet közben kell lefordítani. Azt az egységet, amely a CIL-kódot CPU-utasításokká alakítja, *azonnal (JIT) fordítónak* nevezzük, néha tréfásan *Jitternek*. A .NET-futtatókörnyezet minden futó processzor számára fenntart egy JIT-fordítót, s ezek mindegyikét az alapplatformra optimalizálták.

Ha például egy kézieszközbe (pl. Pocket PC-be) szánt alkalmazást készíttünk, a megfelelő Jitter kúttúnden fut a kevés memóriával rendelkező környezetben. Viszont, ha egy kiszolgálóra (ahol memóriában ritkán van hiány) telepítjük a szerelevényt, a Jittert a sok memóriával rendelkező környezethez optimalizáljuk. Elegendő tehát, ha a fejlesztők megírják azt a kódot, amely JIT-fordítással hatékonyan lefordítható, és a gépeken különböző architektúrákkal is végrehajtható.

Ráadásul, mivel a Jitterek a CIL-utasításokat megfelelő gépi kódokká fordítják, az eredményeket a memóriában a céloperációs rendszernek megfelelő formátumban gyorsítótárazzák. Így, ha a `PrintDocument()` metódust hívjuk meg, a CIL-utasításokat a rendszer az első híváskor platformfügö utasításokká fordítja, és a késöbbi felhasználás érdekében a memóriában tárolja. Ebből következően, ha legközelebb meghívjuk a `PrintDocument()` metódust, a CIL-utasítást nem kell újra lefordítani.

---

**Megjegyzés** Azt is megtehetjük, hogy amikor a .NET Framework 3.5 SDK-ban lévő `ngen.exe` parancssori eszköz segítségével telepítjük az alkalmazást, végrehajtunk egy JIT-előfordítást a szerelevényen. Ezzel tökéletesíthetjük a grafikaiag intenzív alkalmazások indulási idejét.

---

## A .NET-típusmetaadat szerepe

A .NET-szerelvények a CIL-utasításokon kívül olyan teljes és szabatos metaadatot tartalmaznak, amely a binárisban definiált összes típust (osztály, struktúra, felsorolt típus stb.), valamint azok tagjait (tulajdonságok, metódusok, események stb.) is leírja. Szerencsére a legfrissebb és legnagyobb típusmetaadat előállításáig a fordító (és nem a programozó) feladata. Mivel a .NET-metaadat rendkívül aprólékos, a szerelvények teljesen önleíró egységek. A .NET-típusmetaadat formátumának szemléltetésére, nézzük meg a korábban megvizsgált, C# `calc` osztály `Add()` metódusa számára generált metaadatot (az `Add()` metódus VB .NET verziója számára generált metaadat is hasonló):

```

TypeDef #2 (02000003)
-----
TypeFullName: CalculatorExample.calc (02000003)
Flags       : [NotPublic] [AutoLayout] [Class]
[AnsiClass] [BeforeFieldInit] (00100001)
Extends     : 01000001 [TypeRef] system.object
Method #1 (06000003)
-----
Methodname: Add (06000003)
Flags      : [Public] [HideBySig] [ReuseSlot] (00000086)
 RVA       : 0x00002090
ImplFlags  : [IL] [Managed] (00000000)
CallConvnt: [DEFAULT]
ReturnType: I4
Arguments  : 2
  Argument #1: I4
  Argument #2: I4
Parameters
(1) ParamToken : (08000001) Name : x flags: [none] (00000000)
(2) ParamToken : (08000002) Name : y flags: [none] (00000000)

```

A metaadatot a .NET-futtatókörnyezet számos célra felhasználja ugyanúgy, ahogy a különféle fejlesztőeszközök. Az IntelliSense szolgáltatás például – ezt az olyan eszközök, mint a Visual Studio 2008 tartalmazza – lehetővé teszi, hogy a szerelvények metaadata a tervezés folyamán olvasható legyen. A metaadatot különféle objektumbővíztető segédprogramok, hibakereső eszközök és maga a C#-fordító is használja. Kétségtelen, hogy a metaadat a gerince számos .NET-technológiának, köztük a WCF-nek, az XML-webszolgáltatásoknak vagy a .NET remotingnak, a reflexiónak, a késői kötésnek és az objektumsorosításnak. (A 16. fejezet tartalmazza a .NET-metaadat szerepének formálizálását.)



## A szerelvény manifestumának szerepe

Végül, de nem utolsósorban, ahogy már szó volt róla, a .NET-szerelvények olyan metaadatot tartalmaznak, amely magát a szerelvényt írja le (szakszóval *manifestum* , *önleírás*). Egyéb részletek mellett, a manifestum rögzíti az aktuális szerelvény megfelelő működéséhez szükséges összes külső szerelvényt, a szerelvény verziószámát, a szerzői jogra vonatkozó adatokat stb. Ugyanúgy, mint a típusmetaadatot, a szerelvény manifestumát is a fordító generálja. Íme a fjezeletben korábban bemutatott calc.csc kód[raj] fordítása-kor generált manifestum néhány releváns részlete (tétellezzük fel, hogy utasítottuk a fordítót, hogy a szerelvényt calc.exe-nek nevezze el):

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
}
{
  .assembly calc
  {
    .hash algorithm 0x00008004
    .ver 0:0:0:0
  }
  .module calc.exe
  .imagebase 0x00400000
  .subsystem 0x00000003
  .file alignment 512
  .corflags 0x00000001
```

Röviden, a manifestum dokumentálja a calc.exe számára szükséges külső szerelvények listáját (az .assembly extern direktíva révén), valamint a szerelvény különböző jellemzőit (verziószám, modulnév stb.). (Az önleíró adatok hasznosságáról a 15. fejezetben lesz szó részletesebben.)

## A közös típusrendszer (CTS)

Egy adott szerelvény bármennyi különböző típust tartalmazhat. A .NET világában a *típus* általános kifejezés, amely a halmaz (osztály, interfész, struktúra, felsorolt típus, metódusreferencia) egy tagjára utal. Amikor egy .NET-alapú nyelv segítségével hozunk létre megoldásokat, valószínűleg sok ilyen típussal találkozunk. Például akkor, amikor a szerelvényünk olyan osztályt definiál, amely néhány interfészt implementál.

Előfordulhat, hogy az egyik interfészmetódus bemeneti paramétere egy felsorolt típus lesz, a hívónak pedig egy struktúrát ad vissza.

Ahogy láttuk, a közös típusrendszer (CTS) olyan formális specifikáció, amely azt dokumentálja, hogyan kell a típusokat úgy definiálni, hogy azokat a közös nyelvi futtatórendszer (CLR) befogadja. Jellemzően kizárólag azokat a fejlesztőket érdekli a közös típusrendszer belső működése, akik a .NET-platformot megcélzó eszközöket és/vagy fordítóprogramokat készítenek. Minden .NET-programozó számára fontos azonban, hogy megtanulja, hogyan kell a közös típusrendszer által definiált öt típusal bármilyen választott nyelven dolgozni. Az alábbiakban következzon egy rövid áttekintés.

## Az egységes típusrendszer osztálytípusai

Minden .NET-alapú nyelv támogatja legalább az *osztálytípus* fogalmát, amely az objektumorientált programozás (OOP) alapjaira. Egy osztálynak bármennyi tagja (pl. tulajdonságok, metódusok és események) és adatpontja (mezője) lehet. A C#-ban a `class` kulcsszóval deklaráljuk az osztályokat:

```
// C# osztálytípus.  
class Calc  
{  
    public int Add(int x, int y)  
    { return x + y; }  
}
```

A C# segítségével készült CTS-osztályok felépítését az 5. fejezetben vizsgáljuk meg; az 1.1. táblázatban felsoroljuk az osztálytípusok néhány jellemzőjét.

| Osztályjellemző | Valós jelentése |
|-----------------|-----------------|
|-----------------|-----------------|

Lezár-e az osztály, vagy sem?

A lezárt osztályok nem lehetnek más osztályok alaposztályai.

Megvalósít az osztály bármilyen interfészt?

Az *interfész* olyan absztrakt tagok gyűjteménye, amelyek kapcsolatot biztosítanak az objektum és az objektumfelhasználó között. Az egységes típusrendszer lehetővé teszi, hogy egy osztály vagy szerkezet bármennyi interfészt megvalósítson.

Az osztály absztrakt vagy valós?

Az *absztrakt osztályokat* nem lehet közvetlenül létrehozni, de egységes viselkedést biztosítanak a származtatott típusoknak. A *valós* osztályokat közvetlenül létre lehet hozni.

| Osztályjellemző | Valós jelentése |
|-----------------|-----------------|
|-----------------|-----------------|

Milyen az osztály „láthatósága”?

Minden osztályt egy láthatósági attribútummal kell konfigurálni. Alapvetően ez a jellemző határozza meg, hogy használhatják-e külső szereplők is az osztályt, vagy csak a definíáló szereplő.

1.1. táblázat: Az egységes típusrendszer osztályjellemzői

## Az egységes típusrendszer interfésztípusai

Az *interfész* nem más, mint absztrakt tagok definícióinak megnevezett gyűjteménye, amelyet egy adott osztály vagy struktúra támogat (megvalósít hat). Ertőően a COM-tól, a .NET-interfészek *nem* egy közös ősiinterfészből származnak (mint az *unknown*). A C#-ban az interfésztípusokat az *interface* kulcsszó segítségével definiálhatjuk:

```
// A C# interfész típust rendszerint publikus
// típusként deklaráljuk, így más szereplőknek típusai
// is megvalósíthatják a viselkedését.
public interface IDraw
{
    void Draw();
}
```

Az interfészeknek önmagukban nincs sok hasznuk. Viszont, amikor egy osztály vagy struktúra megvalósít egy adott interfészt a maga egyedi módon, egy interfész-hivatkozás révén polymorf módon hozzáférhetünk a funkcionálisához. (Az interfészalapú programozást a 9. fejezetben tárgyaljuk.)

## Az egységes típusrendszer struktúrátípusai

A struktúra fogalmát ugyancsak formalizálták az egységes típusrendszerben. Ha a C-t ismerjük, akkor egyértelmű, hogy ezek a felhasználói típusok (UDT-k) a .NET világában is tovább élnek (bár egy kicsit másképp viselkednek). Egy-szerűen fogalmazva, a *struktúrát* egy érték alapú szemantikával rendelkező pehelyisúlyú osztálytípusnak is tekinthetjük. (A struktúrákkal kapcsolatos további részletekért, lásd a 4. fejezetet.) A struktúrák jellemzően geometriai és matematikai adatok modellezésére alkalmasak, és a C#-ban a *struct* kulcsszó segítségével lehet létrehozni őket:

```
// # struktúrátípus.
struct Point
{
    // A struktúrák mezőket tartalmazhatnak.
    public int xPos, yPos;

    // A struktúrák paraméterezett konstruktorokat tartalmazhatnak.
    public Point(int x, int y)
    { xPos = x; yPos = y; }

    // A struktúrák metódusokat definiálhatnak.
    public void Display()
    {
        Console.WriteLine("{0}, {1}", xPos, yPos);
    }
}
```

## Az egységes típusrendszer felsorolt típusai

A *felsorolt típus* olyan praktikus programozási szerkezet, amely lehetővé teszi a név/érték párok csoportosítását. Például lételezzük fel, hogy olyan videójáték-név/érték párok hozunk létre, amely lehetővé teszi a játékosok számára, hogy három karakterkategóriából válasszanak (Wizard – varázsló, Fighter – harcos vagy Thief – tolvaj). A numerikus értékek nyomon követése helyett, amelynek célja az egyes lehetőségek egyesével történő bemutatása lenne, egyedi felsorolt típusokat is létrehozhatunk, az enum kulcsszó segítségével:

```
// # felsorolt típus.
enum CharacterType
{
    Wizard = 100,
    Fighter = 200,
    Thief = 300
}
```

18 CHAPTER 1 n THE PHILOSOPHY OF .NET  
8849CH01.qxd 10/1/07 10:30 AM Page 18

Az alapértelmezés szerint minden egyes elemet 32 bites egész tárol; módosított állapotban a tárolót, ha szükséges (pl. amikor egy Pocket PC-hez hasonló, kevés memóriás eszközre programozunk). Az egységes típusrendszer emellett azt igényli, hogy a felsorolt típusok egy közös, `enum` osztályból származzanak. Ahogy a 4. fejezetről később kiderül: ez az alaposztály egy sor olyan érdekes tagot definiál, amelyek révén a mögöttes név/érték párokat programozással lehet kibontani, manipulálni és átalakítani.

## Az egységes típusrendszer metódusreferencia-típusai

A *metódusreferencia* a típusbiztos C-stílusú függvénymutató .NET-megfelelője. A legtöbb különbség az, hogy a .NET-metódusreferencia a `system.multicast-delegate`-ből származó *osztály*, nem pedig egy nyers memóriacím egyszerű mutatója. A C#-ban a metódusreferenciákat a `delegate` kulcsszóval deklaráljuk:

```
// Ez a C# metódusreferencia típus bármely metódusra mutat,
// amely egész számot ad vissza, és két egész szám a bemenete.
delegate int BinaryOp(int x, int y);
```

A metódusreferenciák egyrészt akkor nagyon hasznosak, ha valamelyik egyed számára lehetőséget szeretnénk teremteni arra, hogy egy másiknak hi-vást továbbítson, másrészt a .NET-eseményarchitektúra alapjainak megte-remtésékor. A metódusreferenciák a többes küldéshez valódi támogatással rendelkeznek (pl. egy kérdés több címzettnek való továbbításához), valamint az aszinkron függvényhíváshoz (részletesen lásd a 11. és 18. fejezetben).

## Az egységes típusrendszer típusai

Az egységes típusrendszerben formalizált összes típus közül a legtöbbnek bár-mennyi *tagja* is lehet. Szaknyelven szólva, a *típus* egy *halmaz* korlátozza (konstruktor, véglegesítő, statikus konstruktor, beágyazott típus, operátor, me-tódus, tulajdonság, indexelő, mező, írásvédelem, konstans, esemény).

Az egységes típusrendszer különféle „diszítőelemeket” definiál, amelye-ket hozzárendelhetünk egy adott taghoz. Minden tag rendelkezik például egy adott láthatósági jellemzővel (pl. nyilvános, privát, védett stb.). Néhány tag absztrakt tagként deklarálható, amely a származtatott típusokra polimorf viselkedést kényszerít, valamint virtuális tagként, amely alapértelmezett (de felüldefiniálható) megvalósítást definiál. Sőt a legtöbb tagot statikusként (osztályszinten kötött) vagy példányként (objektumszinten kötött) is konfi-gurálhatjuk. A típusok létrehozását a következő néhány fejezet során vizs-gáljuk meg.

---

**Megjegyzés** A C# nyelv támogatja a generikus típusok és tagok létrehozását is (lásd a 10. fe-jezetben).

---

Valódi CTS-adattípusok

Az utolsó szempontként említhető az egysegés típusrendszer vizsgálatában, hogy alapvető adattípusok jól meghatározott halmazát hozza létre. Habár minden nyelv rendelkezik egyedi kulcsszóval a valódi CTS-adattípusok deklarálására, az összes kulcsszó az mscorlib.dll nevű szerezvényben definiált típust adja meg. Nézzük meg az 1.2. táblázat alapján, hogy milyen kulcsszóakkal fejezzük ki a CTS-adattípusokat a különböző .NET-nyelvekben.

| CTS-adattípus | VB .NET-kulcsszó | C#-kulcsszó | C++/CLI-kulcsszó |
|---------------|------------------|-------------|------------------|
|---------------|------------------|-------------|------------------|

|                |          |         |                                 |
|----------------|----------|---------|---------------------------------|
| System.Byte    | Byte     | byte    | unsigned char                   |
| System.SByte   | SByte    | sbyte   | signed char                     |
| System.Int16   | short    | short   | short                           |
| System.Int32   | Integer  | int     | int vagy long                   |
| System.Int64   | Long     | long    | __int64                         |
| System.UInt16  | ushort   | ushort  | unsigned short                  |
| System.UInt32  | UInteger | uint    | unsigned int vagy unsigned long |
| System.UInt64  | ULong    | ulong   | unsigned __int64                |
| System.Single  | Single   | float   | Float                           |
| System.Double  | Double   | double  | Double                          |
| System.Object  | Object   | object  | object                          |
| System.Char    | Char     | char    | wchar_t                         |
| System.String  | String   | string  | string                          |
| System.Decimal | Decimal  | decimal | Decimal                         |
| System.Boolean | Boolean  | bool    | Bool                            |

1.2. táblázat: A valódi CTS-adattípusok

Mivel egy felügyelt nyelv egyedi kulcsszavai csak a system névtér valódi típusainak rövidített jelölései lehetnek, többé nem kell a numerikus adatok típusorúlásától, illetve alácsorúlásától tartanunk a különböző nyelveken, ahogy a sztringek és a logikai értékek belső megjelenésétől sem.

Nézzük meg a következő kódreszleteket, amelyek nyelvi kulcsszavak és formális CTS-típusok segítségével definiálnak 32 bites numerikus változókat C# és VB .NET nyelveken.

```
// Egész számok definíciója C# nyelvben.
int i = 0;
System.Int32 j = 0;

' Egész számok definíciója "ints" VB .NET-ben.
Dim i As Integer = 0
Dim j As System.Int32 = 0
```

## A közös nyelvi specifikáció (CLS)

A különböző nyelvek tehát ugyanazokat a programozási szerkezeteket egyedi nyelvspecifikus kifejezésekkel fejezik ki. A C# nyelvben például a sztringek összefűzését a plusz operátorral (+) jelöljük, míg VB .NET-ben az és jelet (&) használjuk. Ha két különböző nyelv ugyanazt a programozási kifejezést használja is (pl. egy olyan függvényt, amelynek nincs visszatérési értéke), nagy az esély, hogy a szintaxis mégis elég eltérő lesz:

```
// C# módszer, amely nem ad vissza semmit.
public void myMethod()
{
    // érdekes kód...
}
' VB módszer, amely nem ad vissza semmit.
public sub myMethod()
    ' érdekes kód...
End sub
```

Ezek az apró szintaktikai eltérések azonban nem fontosak a .NET-futtatókörnyezet szempontjából, mivel a megfelelő fordítók (ebben az esetben a csc.exe vagy a vb.exe) hasonló CIL-utasításokat adnak ki. A nyelvek viszont a működésük általános szintje alapján is eltérőek lehetnek. Egy .NET-nyelv például rendelkezik kulcsszóval az előjel nélküli adatok megjelenítésére, vagy nem, illetve vagy támogatja a mutatótípusokat, vagy nem. Ilyen lehetőségek mellett jó lenne, ha volna egy olyan referencia-adat, amellyel az összes .NET-alapú nyelv összhangban lenne.

A közös nyelvi specifikáció olyan szabálykészlet, amely részletesen leírja, hogy egy adott .NET-alapú fordítónak milyen minimális és teljes szolgáltatásokat kell támogatnia a közös nyelvi futtatórendszerben (CLR) használt forráskódok létrehozásához, illetve ahhoz, hogy minden .NET-platformot célzó nyelvi egységes módon férhessen hozzá. A közös nyelvi specifikáció több szempontból az egységes típusrendszer által definiált, teljes funkcionálitás *részhatármáznak* tekinthető.

A közös nyelvi specifikáció végül is olyan szabálykészlet, amelyhez a fordító megalkotóinak igazodniuk kell, ha azt szeretnék, hogy a termékek zavartalanul működjen a .NET-univerzumban. Minden szabályhoz olyan egyszerű nevet rendeltek hozzá (pl. 6. CLS-szabály), amely leírja, hogy a szabály milyen hatással van azokra, akik a fordítót létrehozzák, és azokra, akik (valamilyen módon) kapcsolatba kerülnek vele. A közös nyelvi specifikáció alapja a 1. szabály:

- 1. szabály: A CLS-szabályok a típusnak csak azokra a részekre vonatkoznak, amelyek a definíció szerelvényen kívülről is hozzáférhetők.

Ebből a szabályból (helyesen) arra következtethetünk, hogy a többi CLS-szabály nem vonatkozik a .NET-típusok belső működésének létrehozásakor alkalmazott logikára. A típusoknak egyetlen szempontból kell összhangban lenniük a közös nyelvi specifikációval: a tagok definíciója (azaz elnevezési rend, paraméterek és visszatérési típusok) alapján. A tagok megvalósítási logikája bármennyi nem CLS-technikát használhat, hiszen a külvilág nem veszi észre a különbséget.

Ennek illusztrálására nézzük az alábbiakat. A következő `Add()` metódus nem CLS-kompatibilis, ugyanis a paraméterek és a visszatérési értékek előjel nélküli adatokat használnak (és ez nem CLS-követelmény):

```
class Calc
{
    // Előjel nélküli adatok hozzáférése nem CLS-kompatibilis!
    public ulong Add(ulong x, ulong y)
    {
        return x + y;
    }
}
```

Ha viszont belsőleg, egyszerűen előjel nélküli adatokat használtunk volna:



```

class Calc
{
    public int Add(int x, int y)
    {
        // Mivel az ulong változót csak belsőleg használjuk,
        // a kód továbbra is CLS-kompatibilis.
        ulong temp = 0;
        ...
        return x + y;
    }
}

```

akkor még mindig megfelelőenk a CLS-szabályoknak, és nyugodtak lehet-  
nénk afelől, hogy bármelyik .NET-nyelv képes meghívni az Add() metódust.  
A közös nyelvi specifikáció persze számos egyéb szabályt is definiál az 1.  
szabályon kívül. Leírja például, hogy egy adott nyelvnek hogyan kell a sztrin-  
geket és a felsorolt típusokat (belül tárolt típus) belsőleg reprezentálni, hogyan  
definiáljunk statikus tagokat, stb. Szerencsére nem kell ezeket a szabályokat  
megjegyeznünk ahhoz, hogy profi .NET-fejlesztők legyünk. Összegezve, a kö-  
zös nyelvi specifikáció és az egyetemes típusrendszer melyebb ismerete csak az  
eszköz-, illetve fordítóalkotók számára szükséges.

## A CLS-megfeleltetés biztosítása

Ahogy azt a későbbiekben látni fogjuk, a C# számos, nem CLS-kompatibilis  
programszervezést definiál. Jó hír azonban, hogy egy .NET-attribútum se-  
gítségével utasíthatjuk a C#-fordítót, hogy ellenőrizze a kódunk CLS-megfe-  
leltetését:

```

// Utasítja a C#-fordítót a kód CLS-megfeleltetésének ellenőrzésére.
[assembly: System.CLSCompliant(true)]

```

A [CLSCompliant] attribútum utasítja a C#-fordítót, hogy ellenőrizze, a kód  
minden egyes sora megfelel-e a CLS-szabályoknak. Ha szabályellenes dolgot  
talál, a fordító hibát jelez, és megmutatja a hibás kódot. (Az attribútumalapú  
programozás részleteit a 16. fejezetben tárgyaljuk.)

## A közös nyelvi futtatórendszer (CLR)

Az egyes típusrendszeren (CTS) és a közös nyelvi specifikáción (CLS) kívül a harmadik hárombetűs rövidítés, amelyről még beszélünk kell, a közös nyelvi futtatórendszer (CLR). Programozási szempontból a *futtatórendszer* kifejezés olyan külső szolgáltatások gyűjteményére utal, amelyek a kód egy adott lefordított egyiségének a végrehajtásához szükségesek. Például, amikor a fejlesztők a Microsoft Foundation Classes (MFC) segítségével hoznak létre új alkalmazást, tisztában vannak azzal, hogy a programjuk igényli az MFC-alapkönyvtárat (azaz az mfc42.d11-t). A többi népszerű nyelvről is megvan a megfelelő futtatórendszer. A VB6-programozóknak szintén egy vagy két futtatókörnyezeti modulhoz kell igazodniuk (pl. msvbvm60.d11). A Java-fejlesztők a Java virtuális géphez (JVM) kötődnek; stb.

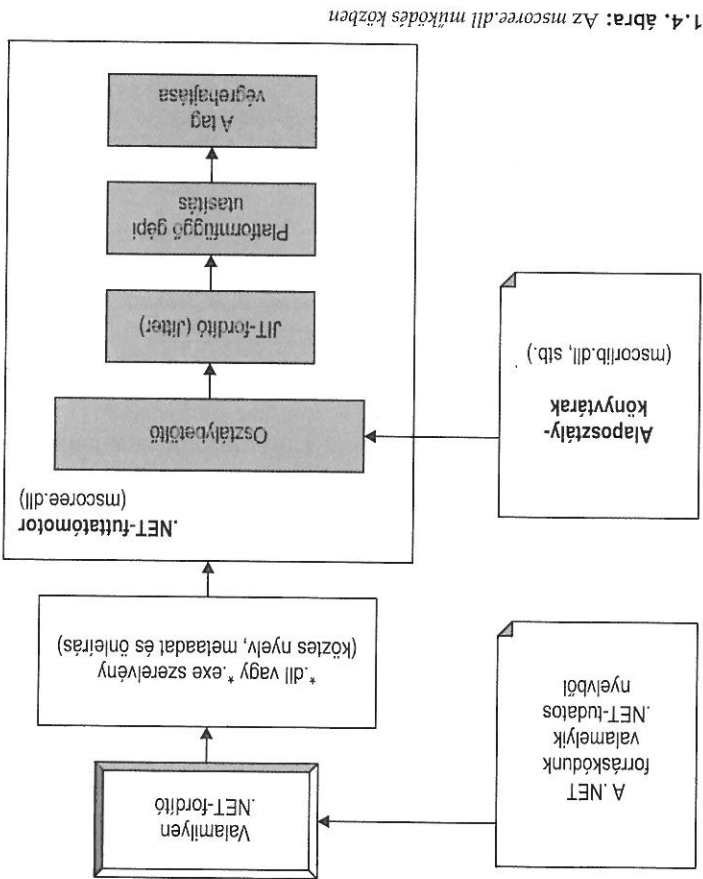
A .NET-platform egy további futtatási rendszert kínál. A legnagyobb különbség a .NET-futtatórendszer és a többi, ímént említett futtatórendszer között az, hogy a .NET-futtatórendszer rendelkezik egy jól meghatározott futtatási réteggel, amelyet az összes .NET-alapu nyelv és -platform használ.

A közös nyelvi futtatórendszer lenyelve fizikailag az mscoree.d11 könyvtárban (más néven Common Object Runtime Engine) jelenik meg. Amikor egy szerelvényt használni szeretnénk, és hivatkozunk rá, az mscoree.d11 automatikusan betöltődik, és ez tölti be a memóriába a szerelvényt. A futtatórendszer számos feladattal felelős. Először megadja a szerelvényt a helyét, és a binárisban lévő metadatot olvasva megkeresi az igényelt típust. A CLR ezt követően a memóriában helyezi el a típust, a hozzárendelt CIL-utasítást platformfüggő gépi utasításokká fordítja, végrehajtja a szükséges biztonsági ellenőrzéseket, majd végrehajtja a szóban forgó kódot.

Az egyedi szerelvények betöltése és az egyedi típusok létrehozása mellett, a CLR, ha szükséges, a .NET-alapoztatalkönyvtárban tárolt típusokkal is kommunikál. Habár a teljes alapoztatalkönyvtár több különálló szerelvényre oszlik, a legfontosabb szerelvény (mscorlib.b.d11) sok alapítípust tartalmaz, amelyek között sok közös programozási feladat található az összes .NET-nyelvtal használó alapvető adattípussal egyetemben. Amikor .NET-megoldásokat hozunk létre, automatikusan hozzáférünk ehhez a szerelvényhez.

Az 1.4. ábra a forráskód (amely az alapoztatalkönyvtár-típusokat használja), egy adott .NET-fordító és a .NET-futtatómotor közötti munkafolyamot mutatja be.

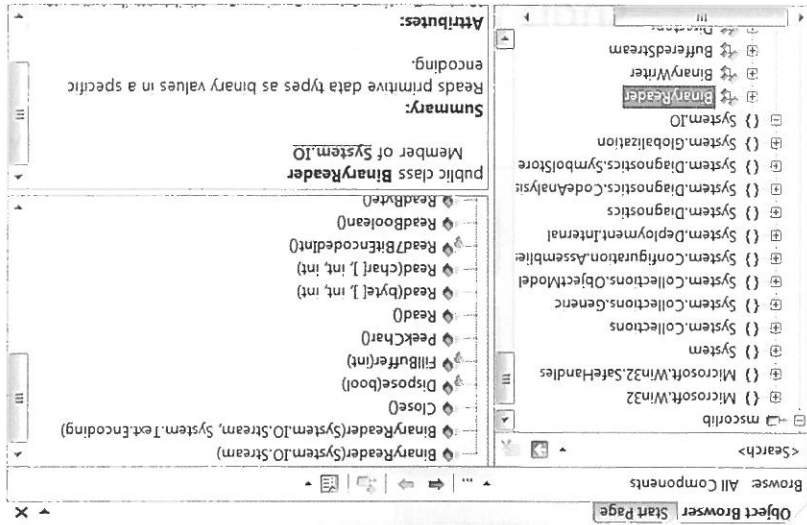
## A szerelvény, a névter és a típus megkülönböztetése



Mindannyian elfogadjuk a forráskódkönyvtárak fontosságát. Az MFC-hez, J2EE-hez és az ATL-hez hasonló könyvtáraknak az a célja, hogy a meglévő forráskódok olyan jól meghatározott készletet biztosítsa a fejlesztők számára, amelyet az alkalmazásaikban felhasználhatnak. A C# nyelv azonban nem rendelkezik nyelvfüggetlen kódkönyvtárral. A C#-fejlesztők ehelyett a nyelv-semleges .NET-könyvtárakat használják. Annak érdekében, hogy a típusok rendezetten helyezkedjenek el az alapsztálykönyvtárakban, a .NET-plat-form messzeemenően alkalmazza a *névter* elvét.

Egyszerűen fogalmazva, a névtér az egy szerelvényben lévő, szemantikai-lag kapcsolódó típusok csoportja. A `system.io` névtér például I/O-típusokat tartalmaz, a `system.data` névtér pedig alapvető adatbázistípusokat definiál; stb. Hangsúlyoznunk kell, hogy egyetlen szerelvény (pl. az `microsoft.d11`) bármennyi névtérrel tartalmazhat, s ezek mindegyike bármennyi típussal rendelkezhet.

Ennek illusztrálására mutatja be az 1.5. ábra a Visual Studio 2008 Object Browser egyik képernyőfotóját. Ezzel az eszközzel megvizsgálhatjuk azokat a szerelvényeket, amelyekre az aktuális projektünk hivatkozik, az egy adott szerelvényben lévő névtérket, az egy névtérben lévő típusokat, valamint egy bizonyos típus tagjait. Az `microsoft.d11` tehát több különböző névtérrel tartalmaz, mindegyiket a saját, szemantikailag kapcsolódó típusaival.



1.5. ábra: Egy szerelvény bármennyi névtérrel tartalmazhat

A legfontosabb különbség e között a felfogás és egy MFC-hez hasonló, névtér-független könyvtár között, hogy bármennyi névtér, amelyik a .NET-futatórendszer veszi célba, *ugyanazokat* a névtérket és *ugyanazokat* a típusokat használja. A következő három program ugyanazt a "Hello World" alkalmazást mutatja be C#, VB .NET és C++/CLI nyelveken:

```

// Hello world C# nyelven
using System;

public class MyApp
{
    static void main()
    {
        Console.WriteLine("Hi from C#");
    }
}

' Hello world VB nyelven
Imports System

Public Module MyApp
    Sub main()
        Console.WriteLine("Hi from VB")
    End Sub
End Module

// Hello world C++/CLI nyelven
#include "stdafx.h"
using namespace System;

int main(array<System::String ^> args)
{
    Console::WriteLine(L"Hi from C++/CLI");
    return 0;
}

```

Minden egyes nyelv használja a `system` névtérben definiált `console` osztályt. Apróbb szintaktikai eltéréseken túl, ez a három alkalmazás nagyon sokban hasonlít egymásra, mind fizikailag, mind logikailag.

.NET-fejlesztőként az elsődleges célunk természetesen a számos .NET-névtérben definiált, temérdek típus megismerése. A legalapvetőbb névtér, amelylyel találkozunk, a `system`. Ez a névtér biztosítja azoknak a típusoknak az alapvető készletét, amelyekre .NET-fejlesztőként időről időre szükségünk van. Valójában egyetlen működőképessé C#-alkalmazást sem hozhatunk létre anélkül, hogy legalább ne hivatkoznánk a `system` névtérre, hiszen az alapvető adattípusokat (`system.int32`, `system.string` stb.) ez a névtér definiálja. Az 1.3. táblázatban a kapcsolódó működések alapján csoportosított, néhány (de nem az összes) .NET-névtér látható.

**.NET-névter**  
**Valós jelentése**

|                                                                     |                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System                                                              | A System névterben számos hasznos típust találunk, amelyek valós adatokkal, matematikai számításokkal, véletlenszám-generálással, környezeti változókkal és személgénykészeléssel foglalkoznak, továbbá közösen használt kivételeket és attribútumokat. |
| System.Collections                                                  | Ezek a névterek számos tárolótípust definiálnak, valamint alaptípusokat és interfészeket, amelyek révén testre szabott gyűjteményeket építhetünk fel.                                                                                                   |
| System.Data                                                         | Ezeket a névtereket a relációs adatbázisokkal folytatott kommunikációra használjuk az ADO.NET alkalmazásakor.                                                                                                                                           |
| System.Data.OleDb<br>System.Data.SqlClient<br>System.Data.SqlClient | Ezek a névterek olyan típusokat definiálnak, amelyeket fájli I/O-ra, adatömörítésre és portkezelésre használunk.                                                                                                                                        |
| System.IO<br>System.IO.Compression<br>System.IO.Ports               | Ezek a névterek olyan típusokat definiálnak, amelyek támogatják a futási időben történő típusmeghatározást, valamint a típusok dinamikus létrehozását.                                                                                                  |
| System.Reflection<br>System.Reflection.Emit                         | Ez a névter biztosítja a .NET-típusok számára azt a képességet, hogy a nem felügyelt kódokkal (pl. C-alapú DLL-ek és Com-kiszolgálók) kommunikáljanak és vizsgált.                                                                                      |
| System.Drawing<br>System.Windows.Forms                              | Ezek a névterek olyan típusokat definiálnak, amelyek révén a .NET eredeti felhasználati felületének eszközkészletét (Windows-UI) használni lehet.                                                                                                       |
| System.Windows<br>System.Windows.Controls<br>System.Windows.Shapes  | A System.Windows névter, számos egyéb névter gyökere, amely a Windows Presentation Foundation (WPF) felhasználati felületi eszközkészletet reprezentálja.                                                                                               |
| System.Linq<br>System.Xml.Linq<br>System.Data.Linq                  | Ezek a névterek a LINQ API-ban végzett programozás során alkalmazott típusokat definiálják.                                                                                                                                                             |

Valós jelentése

.NET-névtér

system.web

Ez az egyik olyan névtér a sok közül, amelyik lehetővé teszi az ASP.NET webalkalmazások létrehozását.

system.ServiceModel

Ez az egyik olyan névtér a sok közül, amelyekkel a WCF API segítségével elosztott alkalmazásokat hozhatunk létre.

system.workflow.activities

Ez a két névtér olyan típusokat definiál, amelyekkel a WF API segítségével munkafolyamat-engedélyezett alkalmazásokat hozhatunk létre.

system.threading

Ez a névtér számos olyan típust definiál, amelyekkel többszárú alkalmazások létrehozására használunk.

system.security

A Security a .NET-univerzum integrált névtér. A biztonságos névtérben számos olyan típust találunk, amelyek engedélyekkel, kriptográfiával stb. foglalkoznak.

System.Xml

Az XML-központú névtér számos olyan típust tartalmaznak, amelyeket az XML-adatokkal való kommunikációra használunk.

1.3. táblázat: A .NET-névtér áttekintése

## A Microsoft-névtér szerepe

Az 1.3. táblázatot végigolvasva bizonyára feltűnt, hogy jó néhány beágyazott névtérnek (system.io, system.data stb.) a System a gyökérnévtér. Ahogy azonban kiderült, a .NET-alapozztálykönyvtár definiálja a system névtérben kívüli legfontosabb gyökérnévtér nagy részét, amelyek közül a leghasznosabb a Microsoft.

Díóhéjban, a Microsoft-névtérbe ágyazott névtér (pl. microsoft.csharp, microsoft.in, microsoft.managementconsole és microsoft.win32) olyan típusokat tartalmaznak, amelyek egyedi, Windows operációs rendszerbeli szolgáltatásokat tartalmaznak, amelyek egyedi, Windows operációs rendszerrel más .NET-sokkal való kommunikációra használatosak. Így ezeket a típusokat más .NET-engedélyezett operációs rendszerrel (pl. a Mac OS X-szel) nem használhatjuk. Könnyűnek a Microsoft-gyökér névtér részleteit azonban nem tárgyalja.

**Megjegyzés** A .NET Framework 3.5 SDK dokumentációjának használatát a 2. fejezet írja le, részletekkel szolgálva a névtérrekről, a típusokról és az alaposztálykönyvtárakban található tagokról.

## Programozott hozzáférés a névterekhez

A névtér tehát nem más, mint a kapcsolódó típusok logikai megértésének és elrendezésének a megfelelő módja. A `system.név` névtérre gondolva, azt feltételezhetjük, hogy a `system.console` egy olyan osztályt jelent, amely a `system.név` névtérben található. A .NET-futtatási rendszer szerint azonban ez nem így van. A futtatási módul csak egy `system.console` névű egységet lát.

A C#-ban a `using` kulcsszó leegyszerűsíti a névtérrekről definiált típusokra történő hivatkozás folyamatát. A működése a következő. Tegyük fel, hogy egy hagyományos asztali alkalmazást szeretnénk létrehozni. A főablak egy kiszolgálóoldali adatbázisból származó adatokra épülő osztálydiagramot jelenít meg, valamint a cégünk logóját. Amíg kevésbé ismerjük még a névtérrekről lévő típusokat, `ime` néhány, amelyekre hivatkozhatunk a programunkban:

```
// ime, az alkalmazás elkészítéséhez szükséges összes névtér.
using system;
using system.drawing;
using system.windows.forms;
using system.data;
using system.data.sqlclient;
using system.server.adathozzaferes;
// Alapvető alaphozzaferes típusok.
// Grafikus rendszertípusok.
// Alapvető alaphozzaferes típusok.
```

Ha egyszer már meghatározunk néhány névtérrel (és hivatkozunk az őket definiáló szerelvényekre), szabadon létrehozhatjuk a bennük található típusok példányait. Ha például a `system.drawing` (amelyet a `system.drawing.név` definiál) egy példányt szeretnénk létrehozni, írjuk a következőket:

```
// A fájl által használt névtérrekről tájékozódás explicit módon.
using system;
using system.drawing;
```

```
class Program
{
    public void DisplayLogo()
    {
        // 20 * 20 pixel-es bittérkép létrehozása.
        Bitmap companyLogo = new Bitmap(20, 20);
        ...
    }
}
```



Mivel a kódjaink a `system.drawing`-ra hivatkozik, a fordító ennek a névternek a tagjaként értelmezi a `Bitmap` osztályt. Ha nem határoztuk meg a `system.drawing` névteret, fordítási hibához jutunk. Teljesen meghatározott névvel azonban szabadon deklarálhatunk változókat is:

```
// A System.Drawing névteret nem tisztázzuk!
using System;

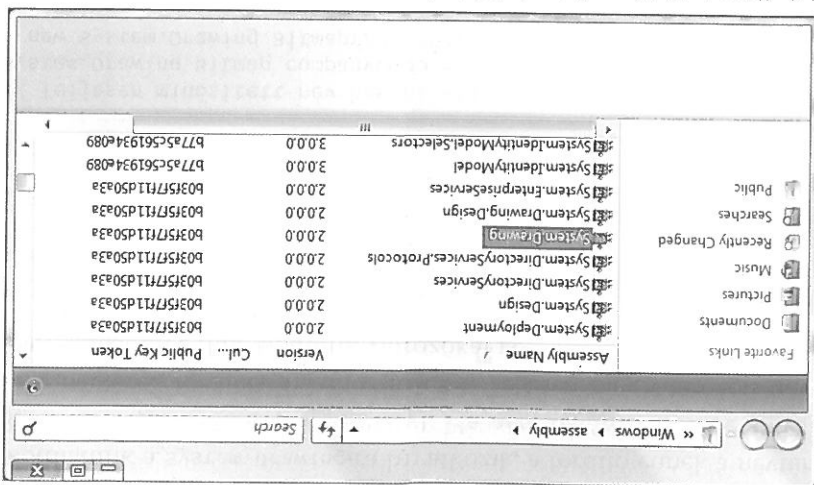
class Program
{
    public void DisplayLogo()
    {
        // Teljesen minősített név használata.
        System.Drawing.CompanyLogo =
            new System.Drawing.Bitmap(20, 20);
        ...
    }
}
```

Ha a típusdefiniálás teljesen minősített névvel történik, akkor tökéletesebb olvashatóságot biztosít, a `C#` `using` kulcsszava ezzel szemben lecsökkenti a bilitentyüleütések számát. A könyvben nem fogjuk a teljesen minősített nevet használni (kivéve, ha kétértelmezhetőséget kell feloldani), hanem a `C#` `using` kulcsszavának egyszerűsített szemléletét követjük.

Ne feledjük azonban, hogy a `using` kulcsszó a típus teljesen minősített nevének a rövidített jelölése, és mindkét szemlélet pontosan ugyanazt a mögöttes CIL-t eredményezi (ugyanis a CIL-kód mindig a teljesen minősített neveket használja), illetőleg nincs hatással a teljesítményre vagy a szerelvény méretére.

## Hivatkozás külső szerelvényekre

Amellett, hogy a `C#` `using` kulcsszavával megadjuk a névteret, a hivatkozott típus aktuális CIL-definícióját tartalmazó szerelvény nevét is meg kell adnunk a `C#`-fordítónak. Amint láttuk, sok alapvető .NET-névter az `microsoft.drawing.dll`-ben található. A `system.drawing.Bitmap` típust azonban a `system.drawing.dll` névű különálló szerelvényben találjuk. A .NET-keretrendszer szerelvényeinek túlnyomó többsége a globális szerelvénytárban (GAC) található. Ez a könyvtár egy Windows-gépen alapértelmezés szerint a `C:\Windows\Assembly` elérési útvonalon érhető el, ahogy az 1.6. ábra mutatja.



1.6. ábra: A globális szerelvénytár (GAC) alaposztálykönyvtára

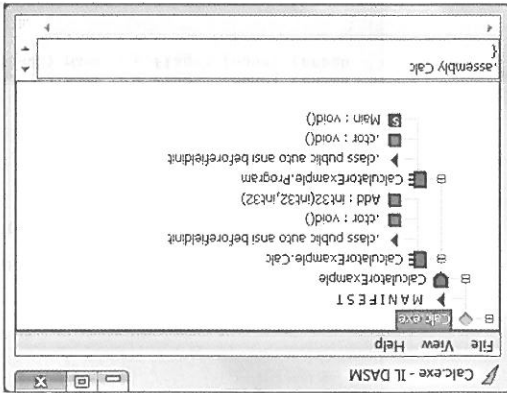
Attól függően, hogy milyen fejlesztőeszköz segítségével hozzuk létre a .NET-al-kalmazásokat, többféleképpen is közzétehetjük a fordítással, hogy mely szerelvényeket szeretnénk beemlíteni a fordítási ciklusba. Ennek a menetével a következő fejezetben ismerkedünk meg.

## Szerelvények feltárasa az ildasm.exe segítségével

Nincs szükség arra, hogy a .NET-platform összes névterét ismerjük és kezeljük, hiszen egy névteret az tesz egyedivé, hogy szemantikailag kapcsolódó típusokat tartalmaz. Ezért, ha nincs szükségünk felhasználói felületre egy egyszerű konzolalkalmazáson kívül, el is felejtethjük a System.Windows.Forms, System.Windows és a System.Web névtereket. Ha egy rajzoló alkalmazást hozunk létre, ahhoz az adatbázis-névtereknek valószínűleg nincs sok közük. Ahogy az előre gyártott kódkezelővel általában lenni szokott, használat közben ismerjük meg őket.

A .NET Framework 3.5 SDK-val megjelent Intermediate Language Disassembler (ildasm.exe) lehetővé teszi bármilyen .NET-szerelvény betöltését és tartalmának vizsgálatát, köztük a hozzárendelt manifestumot, a IL-kódot és a típusmetaadatot. Az ildasm.exe betöltéséhez nyissuk meg a Visual Studio parancssort (Start > All Programs > Microsoft Visual Studio 2008 > Visual Studio Tools), írjuk be, **ildasm**, majd nyomjuk meg az Enter billentyűt.

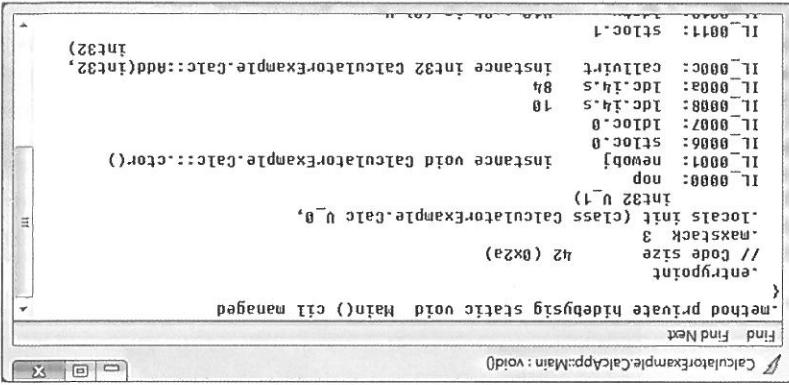
Há már elindítottuk ezt az eszközt, csak nyissuk meg a File > Open menüpontot, és navigáljunk a vizsgálni kívánt szerelvényhez. Ennek szemléltetésére, nézzük meg a Calc.exe szerelvényt, amelyet a fejezet korábbi részében bemutattott calc.csc fájlt alapján generáltunk (lásd 1.7. ábra). Az ildasm.exe hasonló, fánéztű szerelvény szerkezetét mutatja.



1.7. ábra: Az ildasm.exe lehetővé teszi a .NET-szerelvény köztes nyelvi kódjának, manifestumának és metaadatainak megtekintését

## A köztes nyelvi kód megtekintése

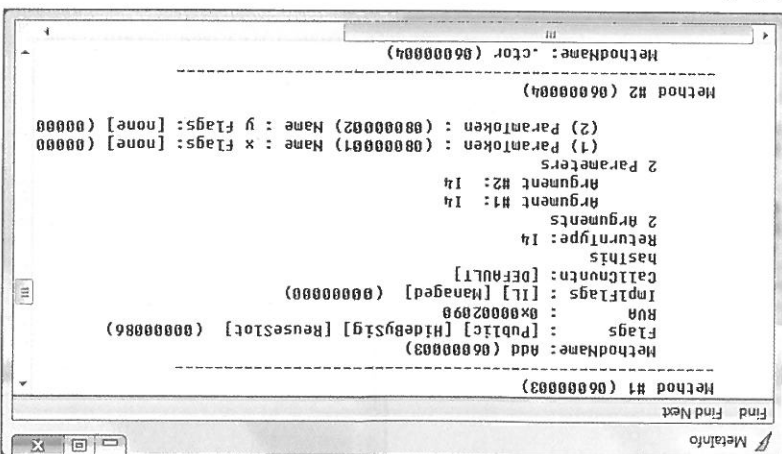
Amellett, hogy megmutatja az adott szerelvényben található névttereket, típusokat és tagokat, az ildasm.exe segítségével egy adott tag CIL-utasításait is megtekinthetjük. Ha például kétszer kattintunk a Program osztály Main() módszerára, egy új ablakban megjelenik a mögöttes köztes nyelvi kód (lásd 1.8. ábra).



1.8. ábra: A mögöttes köztes nyelvi kód megtekintése

## A típusmetaadat megtekintése

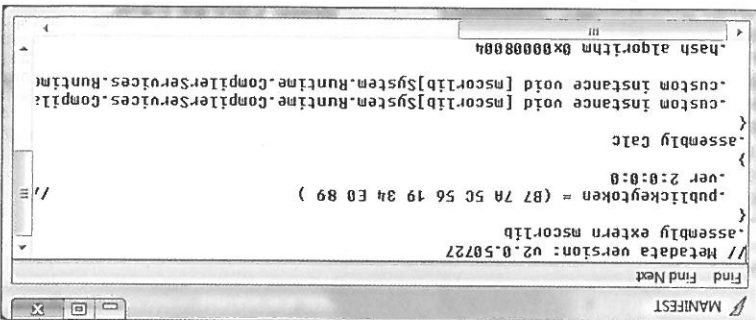
Az aktuálisan betöltött szerelvény típusmetaadatainak megtekintéséhez, nyomjuk meg a Ctrl + M billentyűkombinációt. Az 1.9. ábrán a Calc.Add() módszer metaadatait láthatjuk.



1.9. ábra: Típusmetaadat megtekintése az ildasm.exe segítségével

## Szerelvény metaadatainak (manifesztumának) megtekintése

Végezetül, ha szeretnénk megtekinteni egy szerelvény manifesztumának tartalmát, kattintsunk kétszer a MANIFEST ikonra (lásd 1.10. ábra).



1.10. ábra: Onlívró adatok megtekintése az ildasm.exe segítségével

Természetesen az ildasm.exe-többré is képes, mint amit itt bemutatunk, ezeket a megfelelő részeknél további funkciókkal is szemlélíthetjük.

# A C# 2008 és a .NET 3.5

Első kötet (1-19. fejezet)

Andrew Troelsen



2009

**A C# 2008 és a .NET 3.5 – Első kötet (1-19. fejezet)**

A negyedik kiadás magyarul, két kötetben

Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition

Andrew Troelsen

Copyright 2008 by Apress.

Original English language edition published by Apress. Co Sound View Court 3B, Greenwich CT 06830 USA. Copyright © 2008 by Apress Co.. Hungarian-language edition copyright © 2009 by SZAK Kiadó. All rights reserved.

Magyar fordítás (Hungarian translation) © SZAK Kiadó 2009.

Fordította a SZAK Kiadó fordítócsoportha: Domoszlai László, Erős Melinda, Erős Szilvia, Iváncsy Renáta, Iváncsy Tamás, Lucza Mónika Tiber Melinda, Varga Péter  
Lektor: Dávid Zoltán, MVP és Gincsei Gábor, MVP

Magyar nyelvi gondozás: Laczkó Krisztina

ISBN 978-963-9863-08-8

A könyv fordítása a Kilgray Kft. MemoQ (<http://www.memoqtm.com>) programjával készült, a szöveg helyességét és az elválasztásokat a MorphoLogic He-lyesek nevű programjával ellenőriztük.

**Minden jog fenntartva. Jelen könyvet, illetve annak részeit a kiadó engedélye nélkül tilos reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel elektronikus úton vagy más módon közölni.**



SZAK Kiadó Kft. ■ Az 1795-ben alapított Magyar Könyvkiadók és Könyvterjesztők Egyesülésének a tagja ■ 2060 Bicske, Diófa u. 3. ■ Tel.: 36-22-350-209 ■ Fax: 36-22-565-311 ■ [www.szak.hu](http://www.szak.hu) ■ e-mail: [info@szak.hu](mailto:info@szak.hu) ■ Kiadóvezető: Kis Ádám, e-mail: [adam.kis@szak.hu](mailto:adam.kis@szak.hu) ■ Főszerkesztő: Kis Balázs MCSE, MCT, e-mail: [balazs.kis@szak.hu](mailto:balazs.kis@szak.hu)

Ezt a kiadást Mikkónak, a csodamacskának, a 412-es szám alatt eltöltött éveknek és csodálatos feleségemnek, Amandának ajánlom, aki türelmesen várta, hogy elkészüljek újabb könyvem megírásával.





# Tartalomjegyzék

|                          |      |
|--------------------------|------|
| Köszönetnyilvánítás..... | xxix |
|--------------------------|------|

|                 |      |
|-----------------|------|
| Bevezetés ..... | xxxi |
|-----------------|------|

|                                                                                 |         |
|---------------------------------------------------------------------------------|---------|
| Mi egy csapat vagyunk, az Olvasó és én.....                                     | xxxii   |
| A könyv áttekintése.....                                                        | xxxiii  |
| Első kötet.....                                                                 | xxxiii  |
| 1. rész: Bevezetés a C#-ba és a .NET platformba.....                            | xxxiii  |
| 1. fejezet: A .NET filozófiája.....                                             | xxxiii  |
| 2. fejezet: C#-alkalmazások készítése .....                                     | xxxiv   |
| 2. rész: A C# alapvető építőelemei .....                                        | xxxiv   |
| 3. fejezet: A C# alapvető építőelemei, I. rész .....                            | xxxiv   |
| 4. fejezet: A C# alapvető építőelemei, II. rész.....                            | xxxiv   |
| 5. fejezet: Egységbe zárt osztálytípusok definiálása .....                      | xxxv    |
| 6. fejezet: A származtatás és a polimorfizmus.....                              | xxxv    |
| 7. fejezet: Strukturált hibakezelés.....                                        | xxxv    |
| 8. fejezet: Az objektumok életciklusa.....                                      | xxxv    |
| 3. rész: Haladó programozási szerkezetek a C#-ban .....                         | xxxvi   |
| 9. fejezet: Interfészek használata .....                                        | xxxvi   |
| 10. fejezet: Gyűjtemények és generikus típusok .....                            | xxxvi   |
| 11. fejezet: Metódusreferenciák, események és lambdák.....                      | xxxvi   |
| 12. fejezet: Indexerek, operátorok és mutatók.....                              | xxxvii  |
| 13. fejezet: C# 2008 nyelvi újdonságai.....                                     | xxxvii  |
| 14. fejezet: Bevezetés a nyelvbe ágyazott<br>lekérdezésekbe (LINQ) .....        | xxxvii  |
| 4. rész: Programozás .NET-szerelvényekkel .....                                 | xxxvii  |
| 15. fejezet: A .NET-szerelvények.....                                           | xxxviii |
| 16. fejezet: Típusreflexió, késői kötés és<br>attribútumalapú programozás ..... | xxxviii |
| 17. fejezet: Folyamatok, alkalmazástartományok és<br>objektumkörnyezetek .....  | xxxviii |
| 18. fejezet: Többszálú alkalmazások készítése .....                             | xxxviii |
| 19. fejezet: A köztes nyelv (CIL) és<br>a dinamikus szerelvények.....           | xxxix   |

|                                                                        |       |
|------------------------------------------------------------------------|-------|
| Második kötet .....                                                    | xxxix |
| 5. rész: Bevezetés a .NET alapsztálykönyvtáraiba .....                 | xxxix |
| 20. fejezet: Fájlműveletek és elkülönített tárolás .....               | xxxix |
| 21. fejezet: Bevezetés az objektumsorosítás világába .....             | xl    |
| 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat .....                   | xl    |
| 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat .....                | xl    |
| 24. fejezet: A LINQ API programozása .....                             | xl    |
| 25. fejezet: A WCF .....                                               | xli   |
| 26. fejezet: A WF .....                                                | xli   |
| 6. rész: Felhasználói felületek .....                                  | xli   |
| 27. fejezet: Windows Forms-programozás .....                           | xli   |
| 28. fejezet: A WPF és az XAML .....                                    | xlii  |
| 29. fejezet: Programozás WPF-vezérlőelemekkel .....                    | xlii  |
| 30. fejezet: WPF 2D grafikus renderelés,<br>erőforrások és témák ..... | xlii  |
| 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével .....     | xliii |
| 31. fejezet: ASP.NET weboldalak készítése .....                        | xliii |
| 32. fejezet: ASP.NET-vezérlőelemek,<br>témák és mesteroldalak .....    | xliii |
| 33. fejezet: ASP.NET állapotkezelési technikák .....                   | xliii |
| 8. rész: Függelék .....                                                | xliv  |
| A függelék: A COM és a .NET együttműködése .....                       | xliv  |
| B függelék: Platformfüggetlen .NET-fejlesztés a Monóval .....          | xliv  |
| Öt szabadon letölthető fejezet — még több információ .....             | xlvi  |
| A könyv forráskódjának igénylése .....                                 | xlvi  |
| A lehetséges javítások .....                                           | xlvi  |
| Elérhetőségem .....                                                    | xlvi  |

## 1. rész: Bevezetés a C#-ba és a .NET platformba

### 1. A .NET filozófiája..... 3

|                                             |   |
|---------------------------------------------|---|
| Az előzmények áttekintése .....             | 4 |
| Egy C/Win32 API programozó élete .....      | 4 |
| Egy C++/MFC-programozó élete .....          | 4 |
| Egy Visual Basic 6.0 programozó élete ..... | 5 |
| Egy Java/J2EE-programozó élete .....        | 5 |
| Egy COM-fejlesztő élete .....               | 6 |
| Egy Windows DNA programozó élete .....      | 7 |

|                                                                     |    |
|---------------------------------------------------------------------|----|
| A .NET-megoldás .....                                               | 8  |
| Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS) .....     | 9  |
| Az alaposztálykönyvtárak szerepe .....                              | 10 |
| A C# előnyei .....                                                  | 11 |
| További .NET-alapú programozási nyelvek .....                       | 13 |
| Élet egy többnyelvű világban .....                                  | 14 |
| A .NET-szerelvények áttekintése .....                               | 15 |
| Egyfájlos és többfájlos szerelvények .....                          | 17 |
| A közös köztes nyelv (CIL) szerepe .....                            | 18 |
| A köztes nyelv előnyei .....                                        | 20 |
| A köztes nyelv fordítása platformfüggő utasítássá .....             | 21 |
| A .NET-típusmetaadat szerepe .....                                  | 22 |
| A szerelvény manifesztumának szerepe .....                          | 23 |
| A közös típusrendszer (CTS) .....                                   | 23 |
| Az egységes típusrendszer osztálytípusai .....                      | 24 |
| Az egységes típusrendszer interfésztípusai .....                    | 25 |
| Az egységes típusrendszer struktúrátípusai .....                    | 25 |
| Az egységes típusrendszer felsorolt típusai .....                   | 26 |
| Az egységes típusrendszer metódusreferencia-típusai .....           | 27 |
| Az egységes típusrendszer típustagjai .....                         | 27 |
| Valódi CTS-adattípusok .....                                        | 28 |
| A közös nyelvi specifikáció (CLS) .....                             | 29 |
| A CLS-megfeleltetés biztosítása .....                               | 31 |
| A közös nyelvi futtatórendszer (CLR) .....                          | 32 |
| A szerelvény, a névtér és a típus megkülönböztetése .....           | 33 |
| A Microsoft-névterek szerepe .....                                  | 37 |
| Programozott hozzáférés a névterekhez .....                         | 38 |
| Hivatkozás külső szerelvényekre .....                               | 39 |
| Szerelvények feltárása az ildasm.exe segítségével .....             | 40 |
| A köztes nyelvi kód megtekintése .....                              | 41 |
| A típusmetaadat megtekintése .....                                  | 42 |
| Szerelvény metaadatának (manifesztumának) megtekintése .....        | 42 |
| Szerelvények feltárása a Lutz Roeder's Reflector segítségével ..... | 43 |
| A .NET-futtatókörnyezet telepítése .....                            | 44 |
| A .NET platformfüggetlen természete .....                           | 44 |
| Összefoglalás .....                                                 | 46 |

|                                                                      |           |
|----------------------------------------------------------------------|-----------|
| <b>2. C#-alkalmazások fordítása .....</b>                            | <b>49</b> |
| A .NET 3.5 keretrendszer fejlesztőkészletének szerepe .....          | 49        |
| A Visual Studio 2008 parancssor .....                                | 50        |
| C#-alkalmazások fordítása csc.exe használatával.....                 | 51        |
| Bemeneti és kimeneti célok megadása .....                            | 52        |
| Hivatkozás külső szerelvényekre .....                                | 54        |
| Hivatkozás több külső szerelvényre .....                             | 55        |
| Több forrásfájl fordítása.....                                       | 55        |
| A C#-reakciófájlok használata .....                                  | 56        |
| .NET-alkalmazás készítése TextPad használatával .....                | 59        |
| A C#-kulcsszószínezés bekapcsolása.....                              | 59        |
| A *.cs fájl szűrő konfigurálása.....                                 | 60        |
| A csc.exe integrálása.....                                           | 61        |
| Futtatási parancsok társítása menüelemekhez .....                    | 62        |
| .NET-alkalmazás készítése Notepad++ használatával.....               | 64        |
| Az automatikus kiegészítési lista testreszabása .....                | 65        |
| .NET-alkalmazás készítése SharpDevelop használatával.....            | 66        |
| Egy egyszerű tesztprojekt létrehozása .....                          | 67        |
| .NET-alkalmazás készítése Visual C# 2008 Express használatával ..... | 69        |
| A Visual C# Express néhány jellegzetessége .....                     | 70        |
| .NET-alkalmazás készítése Visual Studio 2008 használatával .....     | 71        |
| A Visual Studio 2008 néhány jellegzetessége.....                     | 72        |
| A .NET-keretrendszer kiválasztása a New Project                      |           |
| párbeszédablak segítségével .....                                    | 72        |
| A Solution Explorer használata .....                                 | 73        |
| Hivatkozás külső szerelvényekre .....                                | 74        |
| Projekt tulajdonságok megtekintése.....                              | 74        |
| A Class View.....                                                    | 75        |
| Az Object Browser .....                                              | 76        |
| Integrált támogatás a kódújratervezéshez.....                        | 76        |
| Kódbővítések és a környezetbe foglalás technológiája.....            | 79        |
| A Visual Class Designer.....                                         | 81        |
| Object Test Bench.....                                               | 84        |
| Az integrált .NET Framework 3.5 dokumentációs rendszer .....         | 85        |
| További .NET-fejlesztőeszközök .....                                 | 87        |
| Összefoglalás.....                                                   | 88        |

## 2. rész: A C# alapvető építőelemei

### 3. A C# alapvető építőelemei, I. rész ..... 91

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| Egy egyszerű C#-program anatómiája.....                                    | 91  |
| Variációk Main() metódusra.....                                            | 93  |
| Alkalmazás-hibakód megadása .....                                          | 94  |
| A parancssori argumentumok feldolgozása .....                              | 96  |
| Parancssori paraméterek megadása Visual Studio 2008-ban.....               | 98  |
| Érdekes kitérő: a System.Environment osztály néhány<br>további tagja ..... | 99  |
| A System.Console osztály .....                                             | 100 |
| A Console osztály alapvető bemenete és kimenete.....                       | 101 |
| A konzolkimenet formázása.....                                             | 102 |
| Numerikus adatok formázása.....                                            | 103 |
| Numerikus adatok formázása a parancssori<br>alkalmazásokon túl.....        | 105 |
| A System-adattípusok és gyorsazonosítók a C#-ban .....                     | 106 |
| Változódeklarálás és inicializálás .....                                   | 108 |
| A beépített adattípusok példányosítása.....                                | 110 |
| Az adattípusok hierarchiája.....                                           | 110 |
| A numerikus adattípusok tagjai .....                                       | 112 |
| A System.Boolean tagjai.....                                               | 113 |
| A System.Char tagjai .....                                                 | 114 |
| Értékek értelmezése sztringadatokból .....                                 | 114 |
| A System.String típus.....                                                 | 115 |
| Egyszerű sztringkezelés.....                                               | 116 |
| Sztringösszefűzés.....                                                     | 117 |
| Vezérlőkarakterek.....                                                     | 118 |
| Szó szerinti sztringek definiálása .....                                   | 119 |
| A sztringek és az egyenlőség.....                                          | 120 |
| A sztring megváltoztathatatlan.....                                        | 121 |
| A System.Text.StringBuilder típus .....                                    | 123 |
| System.DateTime és System.TimeSpan .....                                   | 125 |
| Szűkítő és bővítő adattípus-konverziók.....                                | 126 |
| Hibacsapda állítása a szűkítő adatkonverzióknak .....                      | 128 |
| Projektszintű túlcsoordulás-ellenőrzés beállítása.....                     | 131 |
| Az unchecked kulcsszó .....                                                | 132 |
| A System.Convert szerepe.....                                              | 132 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Ciklus készítése a C#-ban.....                               | 133 |
| A for ciklus.....                                            | 133 |
| A foreach ciklus.....                                        | 134 |
| A while és a do/while ciklusszerkezetek.....                 | 135 |
| Elágazó szerkezetek és a reláció/egyenlőség-operátorok ..... | 136 |
| Az if/else utasítás .....                                    | 136 |
| A switch utasítás .....                                      | 138 |
| Összefoglalás.....                                           | 139 |

#### **4. A C# alapvető építőelemei, II. rész..... 141**

|                                                                  |     |
|------------------------------------------------------------------|-----|
| Metódusok és paramétermódosítók .....                            | 141 |
| Az alapértelmezett paraméterátadási viselkedés .....             | 142 |
| Az out módosító.....                                             | 144 |
| A ref módosító.....                                              | 145 |
| A params módosító .....                                          | 147 |
| A tag túlterhelésének megértése .....                            | 149 |
| Tömbök kezelése a C#-ban .....                                   | 151 |
| A C# tömbinicializáló szintaxisa .....                           | 152 |
| Objektumtömb definiálása .....                                   | 153 |
| Munka a többdimenziós tömbökkel.....                             | 154 |
| Tömbök mint paraméterek (és visszatérési értékek).....           | 156 |
| A System.Array osztály .....                                     | 156 |
| Az Enum típus .....                                              | 159 |
| Az enum alapjául szolgáló tároló vezérlése.....                  | 160 |
| Felsorolások deklarálása és használata.....                      | 160 |
| A System.Enum típus.....                                         | 162 |
| Felsorolás név/érték párainak dinamikus feltárása.....           | 162 |
| A struktúrátípus .....                                           | 165 |
| Struktúraváltozók létrehozása .....                              | 167 |
| Érték- és referenciatípusok.....                                 | 169 |
| Érték- és referenciatípusok, valamint az értékadó operátor ..... | 170 |
| Referenciatípusokat tartalmazó értéktípusok .....                | 172 |
| Referenciatípusok átadása értékként.....                         | 174 |
| Referenciatípusok átadása referenciaként .....                   | 176 |
| Érték- és referenciatípusok: további részletek.....              | 177 |
| Nullázható típusok a C#-ban.....                                 | 179 |
| Munka a nullázható típusokkal .....                              | 180 |
| A ?? operátor.....                                               | 181 |
| Összefoglalás.....                                               | 182 |

## 5. Beágyazott osztálytípusok definiálása ..... 183

|                                                       |     |
|-------------------------------------------------------|-----|
| Bevezetés a C# osztálytípusaiba.....                  | 183 |
| Objektumok memóriaterületének lefoglalása             |     |
| a new kulcsszóval .....                               | 186 |
| A konstruktor.....                                    | 187 |
| Az alapértelmezett konstruktor szerepe.....           | 188 |
| Egyedi konstruktor definiálása .....                  | 189 |
| Az újradefiniált alapértelmezett konstruktor .....    | 190 |
| A this kulcsszó szerepe.....                          | 191 |
| Konstruktorhívások láncolása a this kulcsszóval.....  | 193 |
| A konstruktorok lefutásának sorrendje .....           | 196 |
| A static kulcsszó .....                               | 198 |
| Statikus metódusok (és mezők) definiálása .....       | 199 |
| Statikus adatok definiálása .....                     | 200 |
| Statikus konstruktorok definiálása.....               | 203 |
| Statikus osztályok definiálása .....                  | 205 |
| Az objektumorientált programozás alappillérei .....   | 207 |
| Az egységbe zárás.....                                | 208 |
| A származtatás .....                                  | 209 |
| A polimorfizmus .....                                 | 210 |
| Hozzáférés-módosítók a C#-ban.....                    | 212 |
| Az alapértelmezett hozzáférés-módosítók .....         | 213 |
| Hozzáférés-módosítók és beágyazott típusok .....      | 214 |
| Az első pillér: az egységbe zárás a C#-ban .....      | 215 |
| Egységbe zárás tradicionális lekérdezők és            |     |
| módosítók használatával .....                         | 216 |
| Egységbe zárás típustulajdonságok használatával ..... | 218 |
| A tulajdonságok belső reprezentációja .....           | 220 |
| A get/set utasítástulajdonságok láthatósági           |     |
| szintjeinek vezérlése .....                           | 222 |
| Csak olvasható és csak írható tulajdonságok.....      | 223 |
| Statikus tulajdonságok .....                          | 224 |
| Konstans adatok .....                                 | 226 |
| Írásvédett mezők .....                                | 227 |
| Statikus írásvédett mezők .....                       | 228 |
| Részleges típusok .....                               | 229 |

|                                                                                |            |
|--------------------------------------------------------------------------------|------------|
| A C#-forráskód dokumentálása XML segítségével .....                            | 231        |
| Az XML-fájl generálása.....                                                    | 234        |
| XML-megjegyzések átalakítása az NDoc-on keresztül.....                         | 235        |
| Az elvégzett munka eredményének láthatóvá tétele.....                          | 237        |
| Összefoglalás.....                                                             | 238        |
| <b>6. A származtatás és a polimorfizmus.....</b>                               | <b>239</b> |
| A származtatás alapvető szerkezete .....                                       | 239        |
| Az osztálytípus szülőosztálya.....                                             | 240        |
| Több ősz osztály.....                                                          | 242        |
| A sealed kulcsszó.....                                                         | 243        |
| Osztálydiagramok készítése a Visual Studio 2008-ban .....                      | 245        |
| A második pillér: a származtatás .....                                         | 246        |
| Ősz osztály konstruktorának meghívása a base kulcsszóval .....                 | 249        |
| Családi titok megtartása: a protected kulcsszó.....                            | 251        |
| Lezárt osztály hozzáadása .....                                                | 252        |
| A tartalmazás és a delegálás .....                                             | 254        |
| A beágyazott típus definiálása .....                                           | 255        |
| A harmadik pillér: a polimorfizmus támogatása .....                            | 258        |
| A virtual és override kulcsszavak.....                                         | 259        |
| Virtuális tagok felülbírlása a Visual Studio 2008 segítségével.....            | 261        |
| A virtuális tagok lezárása .....                                               | 262        |
| Az absztrakt osztályok .....                                                   | 263        |
| Polimorfikus interfész építése .....                                           | 265        |
| Tagok eltakarása .....                                                         | 270        |
| Az ősz osztály, a leszármazott osztály és<br>a típusmódosítási szabályok ..... | 272        |
| Az as kulcsszó.....                                                            | 274        |
| Az is kulcsszó .....                                                           | 274        |
| A fő szülőosztály: a System.Object.....                                        | 275        |
| A System.Object.ToString() felüldefiniálása .....                              | 279        |
| A System.Object.Equals() felüldefiniálása.....                                 | 280        |
| A System.Object.GetHashCode() felüldefiniálása.....                            | 281        |
| A módosított Person osztály tesztelése.....                                    | 282        |
| A System.Object statikus tagjai .....                                          | 283        |
| Összefoglalás.....                                                             | 284        |



|                                                                       |                |
|-----------------------------------------------------------------------|----------------|
| <b>7. A strukturált kivételkezelés .....</b>                          | <b>285</b>     |
| Hibák, programhibák és kivételek .....                                | 285            |
| A kivételkezelés szerepe a .NET-ben .....                             | 286            |
| A .NET-kivételkezelés .....                                           | 288            |
| A System.Exception osztály .....                                      | 288            |
| A lehető legegyszerűbb példa .....                                    | 290            |
| Általános kivétel dobása .....                                        | 292            |
| Kivételek elkapása .....                                              | 294            |
| A kivétel tulajdonságainak beállítása .....                           | 295            |
| A TargetSite tulajdonság .....                                        | 295            |
| A StackTrace tulajdonság .....                                        | 297            |
| A HelpLink tulajdonság .....                                          | 297            |
| A Data tulajdonság .....                                              | 298            |
| Rendszerszintű kivételek (System.SystemException) .....               | 301            |
| Alkalmazásszintű kivételek (System.ApplicationException) .....        | 302            |
| Egyedi kivétel készítése, 1. rész .....                               | 302            |
| Egyedi kivétel készítése, 2. rész .....                               | 305            |
| Egyedi kivétel készítése, 3. rész .....                               | 306            |
| Többszörös kivételek feldolgozása .....                               | 308            |
| Általános catch utasítások .....                                      | 310            |
| Kivételek továbbdobása .....                                          | 311            |
| Belső kivételek .....                                                 | 311            |
| A finally blokk .....                                                 | 313            |
| Ki mit dob? .....                                                     | 314            |
| A kezeletlen kivételek eredménye .....                                | 315            |
| Kezeletlen kivételek hibakeresése a Visual Studio használatával ..... | 315            |
| Összefoglalás .....                                                   | 317            |
| <br><b>8. Az objektumok élettartama.....</b>                          | <br><b>319</b> |
| Osztályok, objektumok és referenciák .....                            | 319            |
| Az objektumok élettartama .....                                       | 321            |
| A new kulcsszó köztes nyelve .....                                    | 322            |
| Objektumreferenciák null értékre állítása .....                       | 323            |
| Az alkalmazás gyökérelemének szerepe .....                            | 325            |
| Az objektumgenerációk .....                                           | 327            |
| A System.GC típus .....                                               | 328            |
| A szemétygyűjtés kikényszerítése .....                                | 330            |

|                                                    |     |
|----------------------------------------------------|-----|
| Véglegesíthető objektumok készítése .....          | 333 |
| A System.Object.Finalize() felüldefiniálása .....  | 334 |
| A véglegesítő folyamat részletei .....             | 337 |
| Eldobható objektumok készítése .....               | 337 |
| A C# using kulcsszó egy újabb alkalmazása .....    | 340 |
| Véglegesíthető és eldobható típusok készítése..... | 342 |
| A formalizált felszabadítási minta.....            | 343 |
| Összefoglalás.....                                 | 346 |

### 3. rész: Haladó programozási szerkezetek a C#-ban

#### 9. Interfészek használata ..... 349

|                                                                         |     |
|-------------------------------------------------------------------------|-----|
| Az Interface típus .....                                                | 349 |
| Az interfésztípusok és absztrakt őssosztályok<br>összehasonlítása ..... | 351 |
| Saját interfészek definiálása .....                                     | 354 |
| Az interfész implementálása.....                                        | 356 |
| Interfésztagok hívása az objektumon .....                               | 358 |
| Az interfészreferenciák megszerzése: az as kulcsszó.....                | 360 |
| Az interfészreferenciák megszerzése: az is kulcsszó .....               | 360 |
| Interfészek mint paraméterek.....                                       | 361 |
| Interfészek mint visszatérési értékek.....                              | 364 |
| Interfésztípusok tömbje .....                                           | 364 |
| Interfészimplementálás a Visual Studio 2008 segítségével.....           | 366 |
| Névütközés feloldása explicit interfészimplementációval .....           | 367 |
| Interfészhierarchia tervezése .....                                     | 371 |
| Többszörös öröklés interfésztípusokkal .....                            | 372 |
| Felsorolható típusok készítése (IEnumerable és IEnumerator).....        | 375 |
| Iterátor metódus készítése a yield kulcsszóval.....                     | 379 |
| Nevesített iterátor létrehozása .....                                   | 380 |
| Az iterátormetódusok belső reprezentációja.....                         | 381 |
| Klónozható objektumok készítése (ICloneable) .....                      | 383 |
| Egy összetettebb klónozási példa .....                                  | 385 |
| Összehasonlítható objektumok készítése (Comparable) .....               | 389 |
| Többszörös rendezési sorrend megadása (Comparable) .....                | 392 |
| Egyedi tulajdonságok, egyedi rendezési típusok .....                    | 394 |
| A Callback interfész .....                                              | 395 |
| Összefoglalás.....                                                      | 399 |

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| <b>10. Gyűjtemények és generikus típusok .....</b>                | <b>401</b> |
| A System.Collections névtér interfészei .....                     | 401        |
| Az ICollection interfész .....                                    | 403        |
| Az IDictionary interfész .....                                    | 403        |
| Az IDictionaryEnumerator interfész .....                          | 404        |
| Az IList interfész .....                                          | 404        |
| A System.Collections névtér osztálytípusai .....                  | 405        |
| Az ArrayList típus használata .....                               | 406        |
| A Queue típus használata .....                                    | 408        |
| A Stack típus használata .....                                    | 409        |
| A System.Collections.Specialized névtér .....                     | 410        |
| A dobozolás, kidobozolás és a System.Object kapcsolata .....      | 412        |
| A dobozolással járó probléma .....                                | 415        |
| A típusbiztonság és az erősen típusos gyűjtemények .....          | 416        |
| Egyedi gyűjtemény készítése .....                                 | 417        |
| Az erősen típusos gyűjtemények és a dobozolás .....               | 419        |
| A System.Collections.Generic névtér .....                         | 421        |
| A List<T> típus vizsgálata .....                                  | 423        |
| Saját generikus metódusok létrehozása .....                       | 426        |
| A típusparaméterek kikövetkeztetése .....                         | 427        |
| Generikus struktúrák és osztályok létrehozása .....               | 429        |
| A default kulcsszó a generikus kódban .....                       | 430        |
| Saját generikus gyűjtemény készítése .....                        | 432        |
| A saját generikus gyűjtemény korlátai .....                       | 433        |
| Típusparaméter korlátozása a where kulcsszó használatával .....   | 435        |
| Az operátorkorlátozások hiánya .....                              | 438        |
| Generikus ősosztály létrehozása .....                             | 439        |
| Generikus interfész létrehozása .....                             | 440        |
| Összefoglalás .....                                               | 442        |
| <b>11. Metódusreferenciák, események és lambdák .....</b>         | <b>443</b> |
| A.NET-metódusreferencia .....                                     | 443        |
| A metódusreferencia definiálása C#-ban .....                      | 445        |
| A System.MulticastDelegate és System.Delegate alaposztályok ..... | 448        |
| A legegyszerűbb metódusreferencia .....                           | 449        |
| A metódusreferencia-objektum vizsgálata .....                     | 452        |

|                                                                                  |     |
|----------------------------------------------------------------------------------|-----|
| A Car típus utólagos módosítása metódusreferenciákkal .....                      | 453 |
| Többes küldés engedélyezése .....                                                | 457 |
| Cél eltávolítása a metódusreferencia hívási listájáról .....                     | 459 |
| Egy összetettebb metódusreferencia.....                                          | 460 |
| Metódusreferenciák mint paraméterek .....                                        | 461 |
| Az átirányító kód elemzése .....                                                 | 464 |
| Metódusreferenciák kovarianciája .....                                           | 465 |
| Generikus metódusreferencia készítése .....                                      | 468 |
| A generikus metódusreferencia szimulációja<br>generikus típusok nélkül .....     | 469 |
| C#-események.....                                                                | 471 |
| Az event kulcsszó .....                                                          | 472 |
| Eseménykezelés a felszín alatt .....                                             | 474 |
| A bejövő események figyelése.....                                                | 476 |
| Esemény bejegyzésének egyszerűsítése<br>a Visual Studio 2008 használatával ..... | 477 |
| Egy „korrekt” esemény .....                                                      | 478 |
| A generikus EventHandler<T> metódusreferencia.....                               | 480 |
| Névtelén metódusok a C#-ban.....                                                 | 481 |
| „Külső” változók elérése.....                                                    | 483 |
| A metóduscsoport átalakítása.....                                                | 484 |
| A C# 2008 lambda operátora .....                                                 | 486 |
| A lambda kifejezések részletezése.....                                           | 489 |
| Argumentumok feldolgozása több utasításon keresztül.....                         | 490 |
| A CarDelegate példa áttervezése lambda<br>kifejezések használatával .....        | 492 |
| Lambda kifejezések több paraméterrel<br>(vagy paraméterek nélkül).....           | 494 |
| Összefoglalás.....                                                               | 496 |

## **12. Indexelők, operátorok és mutatók ..... 497**

|                                               |     |
|-----------------------------------------------|-----|
| Az indexelő metódus .....                     | 497 |
| Objektumok indexelése sztringértékekkel ..... | 500 |
| Az indexelő metódusok túlterhelése.....       | 501 |
| Az indexelő metódusok belső ábrázolása.....   | 502 |
| Többdimenziós indexelők.....                  | 503 |
| Indexelő definíciók interfésztípusokon.....   | 503 |

|                                                                  |            |
|------------------------------------------------------------------|------------|
| Az operátor-túlterhelés .....                                    | 504        |
| Kétooperandusú operátorok túlterhelése .....                     | 506        |
| Mi a helyzet a += és -= operátorokkal? .....                     | 508        |
| Egyoperandusú operátorok túlterhelése .....                      | 509        |
| Az egyenlőségvizsgáló operátorok túlterhelése .....              | 510        |
| Összehasonlító operátorok túlterhelése .....                     | 511        |
| A túlterhelt operátorok belső ábrázolása .....                   | 512        |
| Zárszó az operátor-túlterhelésről .....                          | 514        |
| Egyedi típuskonverziók .....                                     | 515        |
| A numerikus konverzió .....                                      | 515        |
| Konverzió az egymásból származó osztálytípusok között .....      | 516        |
| Egyedi konverziós rutinok készítése .....                        | 517        |
| A Square típus további explicit konverziói .....                 | 520        |
| Implicit konverziós rutinok definiálása .....                    | 521        |
| Az egyedi konverziós rutinok belső ábrázolása .....              | 523        |
| Mutatótípusok használata .....                                   | 524        |
| Az unsafe kulcsszó .....                                         | 526        |
| A * és az & operátor .....                                       | 528        |
| A nem felügyelt (és felügyelt) cserefüggvény .....               | 528        |
| Mezőelérés mutatókkal (a -> operátor) .....                      | 530        |
| A stackalloc kulcsszó .....                                      | 530        |
| Típuspéldány rögzítése a fixed kulcsszóval .....                 | 531        |
| A sizeof kulcsszó .....                                          | 532        |
| A C# előfordítói direktívái .....                                | 533        |
| Kódrégiók megadása .....                                         | 534        |
| Feltételes kódfordítás .....                                     | 535        |
| Összefoglalás .....                                              | 537        |
| <b>13. A C# 2008 nyelv szolgáltatásai .....</b>                  | <b>539</b> |
| Implicit típusú lokális változók .....                           | 539        |
| A var használata foreach szerkezetekben .....                    | 541        |
| Az implicit típusú változókra vonatkozó korlátozások .....       | 542        |
| Implicit típusú lokális tömbök .....                             | 543        |
| Az implicit típusú adatok erősen tipizáltak .....                | 544        |
| Az implicit típusú lokális változók előnye .....                 | 545        |
| Automatikus tulajdonságok .....                                  | 546        |
| Interakció automatikus tulajdonságokkal .....                    | 548        |
| Az automatikus tulajdonságok elérésének korlátozása .....        | 549        |
| Az automatikus tulajdonságok és az alapértelmezett értékek ..... | 549        |

|                                                                   |     |
|-------------------------------------------------------------------|-----|
| Bővítő metódusok.....                                             | 551 |
| A bővítő metódusok definiálása .....                              | 552 |
| A bővítő metódusok meghívása példányszinten .....                 | 553 |
| A bővítő metódusok statikus meghívása .....                       | 555 |
| A bővítő metódus hatóköre.....                                    | 555 |
| A bővítő metódusokat definiáló típusok importálása .....          | 557 |
| IntelliSense a bővítő metódusokhoz.....                           | 558 |
| Bővítő könyvtárak létrehozása és használata .....                 | 559 |
| Interfésztípusok kiterjesztése bővítő metódusokkal.....           | 560 |
| Részleges metódusok .....                                         | 562 |
| A részleges metódusok első pillantásra.....                       | 563 |
| A részleges metódusok használata.....                             | 565 |
| Az objektuminicializáló szintaxis.....                            | 566 |
| Egyedi konstruktor meghívása az inicializációs szintaxissal ..... | 568 |
| A belső típusok inicializálása .....                              | 569 |
| Gyűjteményinicializálók .....                                     | 571 |
| Névtelen típusok .....                                            | 572 |
| A névtelen típusok belső ábrázolása .....                         | 573 |
| A ToString() és a GetHashCode() megvalósítása .....               | 575 |
| Az egyenlőség szemantikája névtelen típusoknál.....               | 576 |
| Névtelen típusokat tartalmazó névtelen típusok .....              | 578 |
| Összefoglalás.....                                                | 579 |

## **14. A LINQ. Bevezetés ..... 581**

|                                                    |     |
|----------------------------------------------------|-----|
| A LINQ szerepe .....                               | 581 |
| A LINQ-kifejezések jellemzői                       |     |
| (típzáltság és kiterjeszthetőség) .....            | 583 |
| Alapvető LINQ-szerelvények .....                   | 583 |
| Bevezetés a LINQ lekérdező kifejezésekbe .....     | 585 |
| A LINQ és az implicit típusú lokális változók..... | 588 |
| A LINQ és a bővítő metódusok .....                 | 590 |
| A késleltetett futtatás szerepe.....               | 591 |
| Az azonnali futtatás szerepe .....                 | 592 |
| A LINQ és a generikus gyűjtemények.....            | 593 |
| A LINQ-kifejezés alkalmazása .....                 | 594 |
| A LINQ és a nem generikus gyűjtemények.....        | 595 |
| Adatok szűrése az OfType<T>() használatával.....   | 596 |

|                                                                                                           |     |
|-----------------------------------------------------------------------------------------------------------|-----|
| A LINQ-lekérdezésoperátorok belső ábrázolása .....                                                        | 597 |
| Lekérdező kifejezések létrehozása lekérdezésoperátorok<br>segítségével. Ismétlés .....                    | 598 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a lambdák használatával.....                  | 599 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a névtelen metódusok használatával .....      | 601 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a nyers metódusreferenciák használatával..... | 602 |
| A LINQ-lekérdezésoperátorok .....                                                                         | 603 |
| Elemek számának meghatározása<br>az Enumerable segítségével.....                                          | 604 |
| Egyszerű tesztprojekt létrehozása .....                                                                   | 605 |
| A kiválasztó szintaxis alapjai.....                                                                       | 606 |
| Adatok egy részhalmazának kinyerése.....                                                                  | 608 |
| Új adattípusok leképezése.....                                                                            | 609 |
| Az eredményhalmaz-elemek sorrendjének megfordítása.....                                                   | 610 |
| Kifejezések rendezése .....                                                                               | 610 |
| Különbségek keresése.....                                                                                 | 611 |
| LINQ-lekérdezések: szigetek a szigeteken? .....                                                           | 612 |
| Lekérdezéseredmények transzformálása Array típusúvá .....                                                 | 614 |
| Összefoglalás.....                                                                                        | 615 |

## 4. rész: Programozás .NET-szerelvényekkel

### 15. A .NET-szerelvények ..... 619

|                                                       |     |
|-------------------------------------------------------|-----|
| Egyedi névtér definiálása .....                       | 619 |
| A típus teljesen meghatározott neve .....             | 621 |
| Definiálás álnevek használatával.....                 | 623 |
| Beágyazott névterek létrehozása .....                 | 624 |
| A Visual Studio 2008 „alapértelmezett névtere” .....  | 625 |
| A .NET-szerelvények szerepe.....                      | 626 |
| A szerelvények a kód újrafelhasználását segítik ..... | 626 |
| A szerelvények típushatárokat definiálnak.....        | 627 |
| A szerelvények verzióval rendelkező egységek.....     | 627 |
| A szerelvények önleírók .....                         | 627 |
| A szerelvények konfigurálhatók.....                   | 628 |

|                                                                                    |     |
|------------------------------------------------------------------------------------|-----|
| A .NET-szerelvények felépítése.....                                                | 628 |
| A Win32-fejléc .....                                                               | 629 |
| A CLR-fejléc.....                                                                  | 630 |
| A köztes nyelvi kód, a típusmetaadat és<br>a szerelvénymanifesztum .....           | 631 |
| Opcionális szerelvény-erőforrások.....                                             | 631 |
| Egyfájlos és többfájlos szerelvények .....                                         | 632 |
| Egy állományból álló szerelvény készítése és használata.....                       | 634 |
| A manifesztum .....                                                                | 638 |
| A köztes nyelvi kód .....                                                          | 639 |
| A típusmetaadat.....                                                               | 640 |
| C#-ügyfélalkalmazás készítése .....                                                | 641 |
| Visual Basic ügyfélalkalmazás készítése .....                                      | 643 |
| A nyelvek közti származtatás .....                                                 | 644 |
| Több állományból álló szerelvény készítése és használata.....                      | 645 |
| Az ufo.netmodule fájl.....                                                         | 647 |
| Az airvehicles.dll fájl .....                                                      | 647 |
| A többfájlos szerelvény használata .....                                           | 648 |
| Privát szerelvények .....                                                          | 649 |
| A privát szerelvény azonosítója.....                                               | 650 |
| A keresési sorrend (probing) megadása .....                                        | 651 |
| Privát szerelvények beállítása .....                                               | 651 |
| Konfigurációs fájlok és a Visual Studio 2008.....                                  | 654 |
| A .NET Framework konfiguráló segédprogramja .....                                  | 655 |
| Megosztott szerelvények.....                                                       | 657 |
| Erős nevek megértése.....                                                          | 658 |
| A CarLibrary.dll ellátása erős névvel.....                                         | 660 |
| Erős nevek hozzárendelése a Visual Studio 2008 segítségével.....                   | 662 |
| Megosztott szerelvények telepítése/eltávolítása<br>a globális szerelvénytárba..... | 662 |
| A megosztott szerelvények felhasználása.....                                       | 664 |
| A SharedCarLibClient manifesztuma .....                                            | 665 |
| A megosztott szerelvények konfigurálása .....                                      | 666 |
| Az aktuális megosztott szerelvény befagyasztása.....                               | 667 |
| A megosztott szerelvény 2.0.0.0 verziójának elkészítése.....                       | 668 |
| Dinamikus átirányítás a megosztott szerelvények<br>meghatározott verzióiba .....   | 670 |
| A .NET Framework konfiguráló programja – ismét .....                               | 672 |



|                                                   |     |
|---------------------------------------------------|-----|
| A GAC belső felépítése .....                      | 673 |
| Kibocsátó házirenddel rendelkező szerelvény ..... | 674 |
| A kibocsátó házirend letiltása .....              | 676 |
| A <codeBase> elem .....                           | 677 |
| A System.Configuration névtér .....               | 679 |
| A gépi szintű konfigurációs fájl .....            | 680 |
| Összefoglalás .....                               | 681 |

## **16. Típusreflexió, késői kötés és attribútumalapú programozás..... 683**

|                                                                             |     |
|-----------------------------------------------------------------------------|-----|
| A metaadatok szükségessége .....                                            | 683 |
| Az EngineState felsorolás (részleges) metaadata .....                       | 685 |
| A Car típus (részleges) metaadata .....                                     | 686 |
| A TypeRef .....                                                             | 687 |
| A definiáló szerelvény leírása .....                                        | 687 |
| A hivatkozott szerelvények leírása .....                                    | 688 |
| Sztringliterálok leírása .....                                              | 688 |
| A reflexió .....                                                            | 689 |
| A System.Type osztály .....                                                 | 690 |
| Típusreferencia megszerzése a System.Object.GetType()<br>segítségével ..... | 691 |
| Típusreferencia megszerzése a System.Type.GetType()<br>segítségével .....   | 692 |
| Típusreferencia készítése a typeof() segítségével .....                     | 693 |
| Egyedi metaadat-nézegető létrehozása .....                                  | 693 |
| A metódusok reflexiója .....                                                | 693 |
| Mezők és tulajdonságok reflexiója .....                                     | 694 |
| A típus által megvalósított interfészek reflexiója .....                    | 695 |
| Egyéb érdekességek megjelenítése .....                                      | 695 |
| A Main() metódus megvalósítása .....                                        | 695 |
| Generikus típusok reflexiója .....                                          | 697 |
| Metódusparaméterek és visszatérési értékek reflexiója .....                 | 698 |
| Dinamikusan betöltődő szerelvények .....                                    | 700 |
| Megosztott szerelvények .....                                               | 702 |
| A késői kötés .....                                                         | 704 |
| A System.Activator osztály .....                                            | 705 |
| Paraméter nélküli metódusok hívása .....                                    | 706 |
| Paraméterekkel rendelkező metódusok hívása .....                            | 707 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Programozás attribútumokkal .....                            | 708 |
| Attribútumok feldolgozása .....                              | 709 |
| Attribútumok alkalmazása a C# nyelvben.....                  | 710 |
| Attribútumok konstruktorparamétereinek meghatározása.....    | 712 |
| Az Obsolete attribútum működés közben.....                   | 713 |
| C#-attribútum rövidített formája.....                        | 713 |
| Egyedi attribútum készítése.....                             | 714 |
| Egyedi attribútumok alkalmazása.....                         | 715 |
| Nevesített tulajdonság szintaxisa .....                      | 716 |
| Az attribútumok használatának korlátozása .....              | 716 |
| Alkalmazásszintű (és modulszintű) attribútumok .....         | 718 |
| A Visual Studio 2008 AssemblyInfo.cs fájlja .....            | 718 |
| Attribútumok korai kötés használatával.....                  | 720 |
| Attribútumok késői kötés használatával.....                  | 721 |
| Reflexió, késői kötés és egyedi attribútumok kifejtése ..... | 723 |
| Kiterjeszthető alkalmazás készítése .....                    | 724 |
| A CommonSnappableTypes.dll létrehozása .....                 | 725 |
| C# beépülő modul létrehozása .....                           | 726 |
| Visual Basic beépülő modul készítése .....                   | 726 |
| Kiterjeszthető Windows Forms-alkalmazás készítése .....      | 727 |
| Összefoglalás.....                                           | 731 |

## **17. Folyamatok, alkalmazástartományok és objektumkörnyezetek..... 733**

|                                                        |     |
|--------------------------------------------------------|-----|
| A tradicionális Win32-es folyamat áttekintése .....    | 733 |
| A szálak áttekintése .....                             | 734 |
| Folyamatok kezelése a .NET platformban .....           | 736 |
| A futó folyamatok listázása .....                      | 739 |
| Egy adott folyamat vizsgálata .....                    | 740 |
| Egy folyamat szálkészletének vizsgálata.....           | 741 |
| Egy folyamat modulkészletének vizsgálata .....         | 743 |
| Folyamatok programozott indítása és leállítása .....   | 745 |
| A .NET-alkalmazástartomány .....                       | 746 |
| Folyamat alkalmazástartományainak felsorolása .....    | 749 |
| Új alkalmazástartományok programozott létrehozása..... | 750 |
| Alkalmazástartományok programozott eltávolítása .....  | 752 |
| Az objektumkontextus határai.....                      | 754 |
| Kontextusváltó és kötött típusok .....                 | 755 |

|                                                                     |     |
|---------------------------------------------------------------------|-----|
| Kötött objektum meghatározása .....                                 | 756 |
| Egy objektum kontextusának vizsgálata .....                         | 757 |
| A folyamat, alkalmazástartomány és a kontextus összefoglalása ..... | 759 |
| Összefoglalás .....                                                 | 759 |

## 18. Többszálú alkalmazások létrehozása ..... 761

|                                                                               |     |
|-------------------------------------------------------------------------------|-----|
| A folyamat, az alkalmazástartomány, a kontextus és<br>a szál kapcsolata ..... | 761 |
| A konkurencia problémája .....                                                | 763 |
| A szálszinkronizáció szerepe .....                                            | 764 |
| A .NET-metódusreferenciák rövid áttekintése .....                             | 764 |
| A metódusreferenciák aszinkron viselkedése .....                              | 767 |
| A BeginInvoke() és az EndInvoke() metódusok .....                             | 767 |
| A System.IAsyncResult interfész .....                                         | 768 |
| Metódushívás aszinkron módon .....                                            | 769 |
| A hívó szál szinkronizálása .....                                             | 770 |
| Az AsyncCallback metódusreferencia szerepe .....                              | 771 |
| Az AsyncResult osztály szerepe .....                                          | 773 |
| Egyedi állapotadatok átadása és fogadása .....                                | 774 |
| A System.Threading névtér .....                                               | 775 |
| A System.Threading.Thread osztály .....                                       | 777 |
| Hozzáférés az aktuális szál statisztikáihoz .....                             | 778 |
| A Name tulajdonság .....                                                      | 779 |
| A Priority tulajdonság .....                                                  | 780 |
| Második szál létrehozása kódból .....                                         | 781 |
| A ThreadStart metódusreferencia használata .....                              | 782 |
| A ParameterizedThreadStart metódusreferencia használata .....                 | 784 |
| Előtérben és háttérben futó szálak .....                                      | 786 |
| A konkurencia kérdése .....                                                   | 787 |
| Szinkronizáció a C# lock kulcsszavával .....                                  | 790 |
| Szinkronizáció a System.Threading.Monitor típussal .....                      | 792 |
| Szinkronizáció a System.Threading.Interlocked típussal .....                  | 793 |
| Szinkronizáció a [Synchronization] attribútummal .....                        | 795 |
| Programozás időzített visszahívással .....                                    | 796 |
| A CLR ThreadPool .....                                                        | 798 |
| A BackgroundWorker komponens .....                                            | 800 |
| A BackgroundWorker típus használata .....                                     | 801 |
| Adatfeldolgozás a BackgroundWorker típussal .....                             | 803 |
| Összefoglalás .....                                                           | 805 |

**19. A CIL és a dinamikus szerelvények szerepe ..... 807**

|                                                                  |     |
|------------------------------------------------------------------|-----|
| A CIL-programozás természete.....                                | 807 |
| CIL-direktívák, -attribútumok és -vezérlőkódok.....              | 808 |
| A CIL-direktívák szerepe.....                                    | 809 |
| A CIL-attribútumok szerepe .....                                 | 810 |
| A CIL-vezérlőkódok szerepe.....                                  | 810 |
| A CIL-vezérlőkódok/CIL-mnemonikok megkülönböztetése .....        | 810 |
| Verem írása/olvasása: A CIL veremalapú természete.....           | 811 |
| A round-trip engineering.....                                    | 813 |
| A CIL-kódcímkék szerepe .....                                    | 817 |
| Fejlesztés a CIL-lel: egy *.il fájl módosítása.....              | 818 |
| A CIL-kód fordítása ilasm.exe használatával .....                | 820 |
| CIL-kód fordítása SharpDevelop használatával.....                | 821 |
| A peverify.exe szerepe .....                                     | 823 |
| CIL-direktívák és -attribútumok .....                            | 823 |
| Külsőleg hivatkozott szerelvények meghatározása CIL-ben.....     | 823 |
| Az aktuális szerelvény definiálása CIL-ben.....                  | 824 |
| Névtér definiálása CIL-ben.....                                  | 825 |
| Osztálytípusok definiálása CIL-ben .....                         | 826 |
| Interfészek definiálása és implementálása CIL-ben.....           | 827 |
| Struktúrák definiálása CIL-ben.....                              | 828 |
| Felsorolt típusok definiálása CIL-ben .....                      | 829 |
| Generikus típusok definiálása a CIL-ben .....                    | 829 |
| A CILTypes.il fájl fordítása.....                                | 830 |
| .NET-alaposztálykönyvtár, C#- és CIL-adattípus-leképezések ..... | 832 |
| Típusok definiálása CIL-ben.....                                 | 833 |
| Adatmezők definiálása CIL-ben .....                              | 833 |
| Típuskonstruktorok definiálása CIL-ben.....                      | 834 |
| Tulajdonságok definiálása CIL-ben.....                           | 835 |
| Tagparaméterek definiálása .....                                 | 835 |
| A CIL-vezérlőkódok.....                                          | 836 |
| A .maxstack direktíva.....                                       | 839 |
| Lokális változók deklarálása CIL-ben.....                        | 840 |
| Paraméterek leképzése lokális változókba CIL-ben.....            | 841 |
| A rejtett „this” referencia .....                                | 842 |
| Ciklusok reprezentációja a CIL-ben .....                         | 843 |

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| .NET-szerelvény készítése köztes nyelven.....                     | 844        |
| A CILCars.dll létrehozása.....                                    | 844        |
| A CILCarClient.exe létrehozása.....                               | 847        |
| Dinamikus szerelvények .....                                      | 850        |
| A System.Reflection.Emit névtér .....                             | 851        |
| A System.Reflection.Emit.ILGenerator szerepe.....                 | 852        |
| Dinamikus szerelvény generálása .....                             | 853        |
| A szerelvény- és a modulkészlet generálása .....                  | 856        |
| A ModuleBuilder típus szerepe .....                               | 857        |
| A HelloClass típus és a sztringtípusú tagváltozó generálása ..... | 859        |
| A konstruktorok generálása .....                                  | 859        |
| A SayHello() metódus generálása .....                             | 861        |
| Dinamikusan létrehozott szerelvény használata.....                | 861        |
| Összefoglalás.....                                                | 863        |
| <b>Tárgymutató .....</b>                                          | <b>865</b> |
| <b>A szakmai lektorról .....</b>                                  | <b>879</b> |
| <b>A szerzőről.....</b>                                           | <b>881</b> |



# Köszönetnyilvánítás

Jóllehet egyedül az én nevem jelenik meg a könyv borítóján, de a munka soha nem készülhetett volna el jó néhány tehetséges ember segítségével nélkülül. A következőkben azokhoz szólok, akik lehetővé tették, hogy ezt a könyvet megírjam.

Mindenekelőtt köszönet illeti az Apress összes munkatársát, akikkel már több éve van szerencsém együtt dolgozni. Rendkívüli emberek, akik képesek voltak az én eredeti Word-dokumentumaimat tökéletes prózává varázsolni. Hálás köszönet érte. Már most nagyon várom, hogy mikor dolgozhatunk ismét együtt egy következő könyvön (de azért előtte szabadságolom magam egy időre).

Külön köszönöm technikai szerkesztőm, Gavin munkáját, aki annyi bölcs tanáccsal segített, hogy úgy érzem, ez a kiadás sokkal színvonalasabb lett, mint az előzőek. Mint mindig, az esetleges gépelési hibákért vagy technikai tévedésekért egyedül én vagyok a felelős.

Végül, de nem utolsósorban, köszönöm családomnak, barátaimnak és munkatársaimnak, hogy elviselték mogorva viselkedésemet, amely a kézirat elkészítésének utolsó szakaszaiban jó néhányszor jellemzett engem.





# Bevezetés

Ez a könyv nagyjából 2001 nyara óta létezik, ha nem is ebben a formában: a C# és a .NET platform első kiadása ugyanis akkor jelent meg, a .NET 1.0 Beta 2 verziójával karöltve. Nagy örömmel és hálával tölt el, hogy munkám kedvező fogadtatásra lelt a sajtóban, és ami még fontosabb, az olvasók körében is. Az évek során jelölték a Jolt Award döntőjére (vesztettem... kellemetlen) és a 2003-as Referenceware Excellence Awardon a programozási könyvek kategóriájában (nyertem?... de jól)

Azóta azon dolgoztam, hogy a könyvet frissítsem és aktualizáljam a .NET platform újabb verzióinak megfelelően. Eközben limitált darabszámmal megjelentettem egy speciális kiadványt, amely bemutatta a .NET 3.0 verzióját (Windows Presentation Foundation, Windows Communication Foundation, és a Windows Workflow Foundation), valamint áttekintést nyújtott olyan különböző technológiákról, amelyek még megjelenésre vártak, és amelyeket most LINQ néven ismerünk.

Jelenleg a könyv negyedik kiadását tartja a kezében az Olvasó. Ez a kiadás egyrészt tömören megismétli a korábbi ismereteket, azaz a .NET 3.5-ös verziójában található legfontosabb változások magyarázatát, másrészt számos teljesen új fejezetet is tartalmaz, de ezen túlmenően több eddigi rész is jelentősen bővült.

Ahogy a korábbiakban megszokhattuk, ez a kiadás is egyszerű és érthető nyelven ismerteti a C# nyelvet és a .NET-alaposztálykönyvtárakat. Soha nem értettem azokat a szakírókat, akik úgy adják elő mondandójukat, hogy az jobban hasonlít egy GRE-szószedetről írt tanulmányra, mint egy szakkönyvre. Munkám továbbra is arra fekteti a hangsúlyt, hogy a benne található információk segítségével jól használható szoftverrendszereket lehessen készíteni, így nem töltök túl sok időt az ezoterikus részletek tanulmányozásával, hiszen ezekkel csak nagyon kevés olvasóm fog valaha is foglalkozni.

## Mi egy csapat vagyunk, az Olvasó és én

A szakírók az emberek igényes csoportjának írnak (nekem már csak tudnom kell – egy vagyok közülük). Tisztában vagyunk azzal, hogy egy szoftver elkészítése – bármelyik platformmal történjen is (.NET, J2EE, COM stb.) – rendkívül aprólékos munka, ugyanakkor nagyon specifikus a részleg, a cég, az ügyfél vagy a tárgykör szerint. Hiszen lehet, hogy épp az elektronikus kiadói iparágban dolgozunk, vagy éppenséggel rendszert fejlesztünk az államnak, esetleg a helyi önkormányzatnak, a NASA-nak vagy egy katonai részlegnek. Én a magam részéről gyermekeket oktató szoftvert fejlesztettem, számos n-rétegű rendszert, valamint projekteket a gyógyászati és a pénzügyi ágazatokban. Majdnem nulla az esélye, hogy annak a kódnak, amelyet egyikünk ír a munkájához, köze lesz ahhoz a kódhoz, amelyet a másikunk ír a saját munkájához (kivéve, ha véletlen épp együtt dolgoztunk korábban!).

Ezért szándékosan kerülöm az olyan mintapéldákat, amelyeket konkrét iparághoz vagy programozási feladathoz köthetnénk: a C#, az objektumorientált programozás és a .NET 3.5 alapsztály könyvtárait ipari alkalmazásoktól független példákon keresztül mutatom be. Ahelyett, hogy minden példám adatok tömkelegét szolgáltatná, bérlistát számolna, vagy ehhez hasonlókkal szolgálna, az én példáimhoz mindenkinek van valami köze: autókról szólnak (ráadásként néhány geometriai struktúrával és alkalmazottakkal). És ez az a pont, ahol összetalálkozunk.

Az *én feladatom*, hogy legjobb tudásom szerint elmagyarázzam a C# programozási nyelvet és a .NET platform alapvető jellegzetességeit. Ugyanígy mindent megteszek azért, hogy ellássam a kedves Olvasót különféle eszközökkel és stratégiákkal, amelyek lehetővé teszik, hogy a könyv alapján folytatni lehessen a téma tanulmányozását.

Az *olvasó feladata* pedig, hogy a megszerezett tudást képes legyen alkalmazni saját programozási feladatában. Persze egyértelmű, hogy a projektek nem állatnevekkel ellátott autók körül forognak, de hát pont erről szólna az alkalmazott ismeret. Biztos lehet benne az Olvasó, hogy ha egyszer megértette a könyvben bemutatott elveket, ismereteinek birtokában olyan .NET-alkalmazásokat írhat, amelyek használhatók saját programozási környezetükben.

# A könyv áttekintése

A *Pro C# 2008 and the .NET 3.5 Platform* (A C# és a Microsoft .NET 3.5.) negyedik kiadása logikailag nyolc különálló részből áll, s ezek mindegyike további fejezeteket tartalmaz. Jó néhány témakör, például az alapvető C#-szerkezetek, az objektumorientált programozás és a platformfüggetlen .NET fejlesztése kibővült a korábbiakhoz képest, saját fejezetet kapott. Az új kiadás a .NET 3.0–3.5 programozás tulajdonságait (LINQ, WCF, WPF, WF stb.) összefoglaló fejezeteket is tartalmaz.

---

**Megjegyzés** A magyar változat kiadásának előkészítése során a SZAK Kiadó úgy döntött, hogy az eredeti könyv tartalmát két kötetben jelenteti meg. Ezt a viszonylag nagy terjedelem indokolja. Az alábbiakban mind a két kötet tartalmát ismertetjük, így biztosítva, hogy az is képet kapjon a mű egészéről, akinek csak az egyik kötet kerül a kezébe. (Ezt a Bevezetést változatlan formában mind a két kötetben szerepeltetjük.)

---

## Első kötet

### 1. rész: Bevezetés a C#-ba és a .NET platformba

Az első rész bemutatja a .NET platform alapvető természetét és számos olyan fejlesztőeszközt (köztük több nyílt forráskódút), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány alap C# programozási nyelvi részlettel és a .NET-típusrendszerrel is megismerkedünk.

#### 1. fejezet: A .NET filozófiája

Az első fejezet voltaképpen a könyv további részeinek a gerincét képezi. A hagyományos Windows-fejlesztések világának vizsgálatával kezdjük, és feltárjuk a korábbi hátrányokat is. A fejezet elsődleges célja azonban az, hogy megismerjünk jó néhány .NET-központú építőkövet, például az egységes nyelvi futtatót (Common Language Runtime – CLR), a közös típusrendszert (Common Type System – CTS), az egységes nyelvi specifikációt (Common Language Specification – CLS) és az alaposztálykönyvtárakat. Egyszerű betekintést nyerhetünk a C# programozási nyelvbe és a .NET-szerelvényformátumába, valamint áttekintést kapunk a .NET platform platformfüggetlen természetéről (a B függelék részletesen tárgyalja ezt a témakört).

## 2. fejezet: C#-alkalmazások készítése

A fejezet célja, hogy különféle eszközök és technikák ismertetésével bevezessen minket a C#-forráskódfájl fordításának folyamatába. Először a parancssoros fordító (`csc.exe`) és a C#-eredményfájlok használatát ismerhetjük meg. A fejezet további részében pedig számos forráskódszerkesztőt és integrált fejlesztési környezetet (integrated development environments – IDEs) tanulmányozhatunk, többek között a következőket: TextPad, Notepad++, Sharp-Develop, Visual C# 2008 Express és a Visual Studio 2008. Ezenkívül számos olyan további programozási eszközt ismertetünk, amelyet bármely .NET-fejlesztőnek azonnal elő kell tudnia húzni a farzsebéből.

## 2. rész: A C# alapvető építőelemei

A 2. részben tárgyalt témakörök kiemelten fontosak, hiszen az itt megjelenő tudnivalókat folyamatosan használjuk függetlenül attól, hogy milyen jellegű .NET-szoftvert szándékozunk fejleszteni (webes alkalmazás, asztali grafikus alkalmazás, forráskódkönyvtárak, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek a C# nyelv alapvető építőkövei, beleértve az objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti továbbá, hogy hogyan kell kezelni a futásidejű kivételeket, és bevezet minket a .NET szemétygyűjtő szolgáltatásának részleteibe is.

### 3. fejezet: A C# alapvető építőelemei, I. rész

Ebben a fejezetben kezdjük el részletesen tanulmányozni a C# programozási nyelvet. Megismerkedünk a `main()` metódus szerepével és számos részlettel a .NET platform belső adattípusaival kapcsolatban, beleértve a szövegesadatkezelést a `System.String` és a `System.Text.StringBuilder` névterek alkalmazásával. Megvizsgáljuk továbbá az iterációs és a döntési szerkezeteket, a szűkítő és a bővítő műveleteket, valamint az unchecked kulcsszó használatát.

### 4. fejezet: A C# alapvető építőelemei, II. rész

A 4. fejezet befejezi az alapvető C#-szemléletek meghatározását, a típustúlterhelt metódusok létrehozásától az `out`, `ref` és `params` kulcsszavakon keresztüli paraméterek definiálásáig. Vizsgáljuk, hogyan kell adathalmazt létrehozni és kezelni, hogyan kell olyan adattípust készíteni, amelynek `null` is lehet az értéke (a `?` és a `??` operátorokkal), és megmutatjuk az *értéktípus* és a *referenciatípus* közötti különbséget is.

## 5. fejezet: Egységbe zárt osztálytípusok definiálása

Ebben a fejezetben az objektumorientált programozással (OOP) ismerkedhetünk meg a C# programozási nyelv használatán keresztül. Először meghatározzuk az OOP pilléreit (beágyazás, származtatás és polimorfizmus), majd a fejezet további részében azt nézzük meg, hogyan kell konstruktorok, tulajdonságok, statikus tagváltozók, konstansok és csak olvasható fájlok használatával robusztusosztály-típusokat létrehozni. Végezetül ismertetjük a részleges típus definícióját és a C# XML-kódjának dokumentációs szintaktikáját.

## 6. fejezet: A származtatás és a polimorfizmus

Ebben a részben az OOP maradék pilléreit (származtatás és polimorfizmus) vizsgáljuk meg; ezek a segítségével összetartozó osztálytípusok családját hozhatjuk létre. Megnézzük a virtuális metódusok és absztrakt metódusok (és az absztrakt alaposztályok) szerepét, valamint a *polimorf interfész* természetét. Végül, de nem utolsósorban, ez a fejezet ismerteti a .NET platform legfontosabb alaposztályának, a `System.Object` osztálynak a szerepét is.

## 7. fejezet: Strukturált hibakezelés

Ez a fejezet elsősorban azt mutatja be, mit kell tenni a strukturális kivételkezelés segítségével a futási időben keletkezett anomáliákkal. Nemcsak az ilyen jellegű problémák megoldására szolgáló C#-kulcsszavakat ismerjük itt meg (`try`, `catch`, `throw` és `finally`), hanem az alkalmazásszintű és a rendszerszintű kivételek közötti különbséget is. Megismerhetünk továbbá többféle, a Visual Studio 2008-ban található eszközt, amelyek segítségével hibakeresést végezhetünk a figyelmen kívül hagyott kivételek között.

## 8. fejezet: Az objektumok életciklusa

Ennek a résznek az utolsó fejezete azt tárgyalja, hogy hogyan kezeli a CLR a memóriát a .NET szemétyűjtőjének a segítségével. Megérthetjük az alkalmazásgyökök, az objektumnemzedékek és a `System.GCType` szerepét. Az alapok megismerése után, a fejezet további része az *eldobható objektumokkal* (az `IDisposable` interfészen keresztül) és a véglegesítő folyamattal (a `System.Object.Finalize()` metóduson keresztül) foglalkozik.

### 3. rész: Haladó programozási szerkezetek a C#-ban

A könyv 3. része ismerteti a C# nyelv haladóbb jellegű (de ugyancsak nagyon fontos) elveit. Az interfészek és metódusreferenciák vizsgálatával befejezzük a .NET típusrendszerével való ismerkedést. A továbbiakban betekintést nyerhetünk a generikus típusok és a számos új C# 2008 nyelvi elem szerepébe, valamint a LINQ-ba is.

#### 9. fejezet: Interfészek használata

Ennek a fejezetnek az anyagát az interfészalapú programozást is magába foglaló objektumalapú fejlesztés alkotja. Megismerhetjük belőle, hogy hogyan kell olyan típusokat definiálni, amelyek többszörös viselkedést támogatnak, hogyan lehet ezeket a viselkedéseket futásidőben feltérképezni, és hogyan lehet szelektíven elrejtetni bizonyos viselkedéseket az *explicit interfészmegvalósítás* segítségével. Továbbá a számos előre definiált .NET-interfész típus megismeréséhez el kell sajátítanunk azoknak az egyedi interfészeknek a használatát, amelyek ad hoc jellegű eseményarchitektúra készítéséhez szükségesek.

#### 10. fejezet: Gyűjtemények és generikus típusok

Ez a fejezet a `System.Collections` névtér gyűjtemény típusainak tanulmányozásával kezdődik; ezek már a .NET platform kezdeti kiadásának is részei voltak. Noha a .NET 2.0 megjelenése óta a C# programozási nyelv lehetővé teszi generikus típusok használatát, ahogy a későbbiekben kiderül, a generikus programozás lényegesen megnöveli az alkalmazás teljesítményét és a típusbiztonságot. Nemcsak azt vizsgáljuk meg, hogy milyen különböző generikus típusok léteznek a `System.Collections.Generic` névtérben, de azt is, hogyan készíthetjük el saját generikus metódusainkat és típusainkat (megszorításokkal vagy anélkül).

#### 11. fejezet Metódusreferenciák, események és lambdák

A 11. fejezetnek az a célja, hogy érthetővé tegye a *metódusreferencia*-típusokat. Leegyszerűsítve: a .NET-metódusreferencia olyan objektum, amely az alkalmazás más metódusaira mutat. Ez a minta olyan rendszerek készítését teszi lehetővé, amelyek több objektum összekapcsolódását engedik meg kétirányú együttműködés segítségével. A .NET-metódusreferenciák vizsgálata után megismerkedünk a .NET event kulcsszavával, amelyet arra használunk, hogy egyszerűsítsük a nyers metódusreferencia programozásának kezelését. Végezetül megvizsgáljuk a C# 2008 lambda-operátort ( $\Rightarrow$ ), és feltárjuk a kapcsolatot a metódusreferenciák, a névtelen metódusok és a lambda-kifejezések között.

## 12. fejezet: Indexerek, operátorok és mutatók

Ez a fejezet a C# programozási ismereteinket mélyíti el számos haladó programozási technika segítségével. Megismerhetjük, hogyan kell operátorokat túlterhelni, és hogyan kell a típusaink számára egyedi konverziós rutinokat írni (akár implicit, akár explicit formában). Megtanulhatjuk azt is, hogy hogyan kell létrehozni és használni a *típusindexelőket*, és hogyan kell kezelni a C-stílusú mutatókat a „nem biztonságos” kódkörnyezetben.

## 13. fejezet: C# 2008 nyelvi újdonságai

A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközzel bővült, amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API használatát (erről a 14. fejezetben többet is megtudhatunk). Megismerjük a lokális változók implicit típussal történő ellátásának a szerepét, a részleges metódus, az automatikus tulajdonságok, a bővítő metódusok, a névtelen típusok és az objektuminicializáló szintaxis használatát.

## 14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (LINQ)

Ez a fejezet ismerteti a LINQ-et (Language Integrated Query.- nyelvi integrált lekérdezés), amelyet nyugodtan nevezhetünk a .NET 3.5 legérdekesebb részének. Ahogy a fejezet későbbi részéből kiderül, a LINQ segítségével lehetőségünk nyílik erősen típusos *lekérdezéseket* írni, amelyeket számos LINQ-célpontra alkalmazhatunk, hogy a szó legtágabb értelmében „adatot” kezelhessünk. Megtanuljuk a LINQ objektumokra történő alkalmazását, ennek segítségével a LINQ-kifejezéseket adattárolókra (tömbök, kollekciók, egyedi típusok) fogjuk tudni alkalmazni. Ez a tudás nélkülözhetetlen lesz akkor, amikor azt vizsgáljuk meg, hogyan kell LINQ-kifejezéseket relációs adatbázisokon (a LINQ az ADO-n keresztül) és XML-dokumentumokon (a LINQ az XML-en keresztül) használni.

## 4. rész: Programozás .NET-szerelvényekkel

A negyedik rész a .NET-szerelvényformátum részleteit taglalja. Nemcsak azt tanulhatjuk meg, hogyan kell .NET-kódkönyvtárakat telepíteni és konfigurálni, de azt is, hogy mi a .NET bináris kép belső összeállítása. Ebben a részben lesz szó a .NET-attribútumok szerepéről és a többszálú alkalmazások létrehozásának módjáról is. A későbbi fejezetek néhány haladó témát érintenek, például az objektumkörnyezetet, a köztes nyelvi kódot és a dinamikus szerelvényeket.

## 15. fejezet: A .NET-szerelvények

Első megközelítésben a *szerelvény* nem más, mint egy felügyelt \*.dll vagy \*.exe bináris fájl leírására szolgáló kifejezés. Ám a .NET-szerelvények története sokkal több ennél. Ebben a fejezetben megtudjuk, mi az egyfájlos és a többfájlos szerelvények közötti különbség, és, hogy ezeket hogyan kell készíteni és telepíteni. Megvizsgáljuk, hogyan kell privát és megosztott szerelvényeket konfigurálni XML-alapú \*.config fájlokkal és kibocsátó házirenddel rendelkező szerelvényekkel. Mindeközben megtudhatjuk, hogy milyen a GAC (Global Assembly Cache – globális szerelvénytár) belső struktúrája, és mi a .NET keretrendszer konfigurációs segédprogramjainak a szerepe.

## 16. fejezet: Típusreflexió, késői kötés és attribútumalapú programozás

A 16. fejezetben folytatjuk az ismerkedést a .NET-szerelvényekkel. A `System.Reflection` névtér használatával kapcsolatban azt vizsgáljuk, a futási időben hogyan valósul meg a típusmeghatározás folyamata. Az ilyen jellegű típusok használatával lehetőségünk van olyan alkalmazások létrehozására, amelyek futás közben képesek a szerelvények metaadatait olvasni. Kiderül, hogyan lehet a *késői kötés* segítségével futási időben dinamikusan betölteni és létrehozni típusokat. A fejezet utolsó témaköre – mind a szabványos, mind az egyedi – .NET-attribútumok szerepének ismertetése. Az egyes témakörök hasznosságának szemléltetésére a fejezet konklúziójaként bemutatunk egy kiterjeszthető Windows Forms-alkalmazást.

## 17. fejezet: Folyamatok, alkalmazástartományok és objektumkörnyezetek

A szerelvényekről szóló tudnivalók után ez a fejezet a betöltött .NET végrehajtható fájlok felépítését ismerteti. Az elsődleges cél a folyamatok, az alkalmazástartományok és a kontextusbeli határok közötti kapcsolat illusztrálása. Ezek alapozzák meg a következő fejezet témáját: a többszálú alkalmazások készítésének módját.

## 18. fejezet: Többszálú alkalmazások készítése

Ez a fejezet azt mutatja be, hogy hogyan kell többszálú alkalmazásokat készíteni, és számos olyan technikát ismertet, amelyeket a szálbiztos kód készítéséhez alkalmazhatunk. A fejezet elején átvisszafutunk a .NET metódusreferenciátípusait, hogy megértsük, valójában hogyan támogatják a metódusreferenciák



az aszinkronfüggvény-hívásokat. Majd megvizsgáljuk a `System.Thread` névtérben található típusokat. Ezek között számos olyan van (`Thread`, `ThreadStart`), amelynek segítségével könnyen lehet újabb végrehajtási szálakat készíteni. Végül megismerkedünk a `Backgroundworker` típussal, amely nagyban megkönnyíti a felhasználói felületről történő szálkészítést.

## 19. fejezet: A köztes nyelv (CIL) és a dinamikus szerelvények

A 4. rész utolsó fejezetének szerepe kettős. Először a korábbinál részletesebben megvizsgáljuk CIL szintaktikáját és szemantikáját, majd a `System.Reflection`. Emit névtér szerepét mutatjuk be. Ezeknek a típusoknak a használatával lehetőségünk nyílik olyan szoftver készítésére, amely képes .NET-szerelvényeket futásidőben generálni a memóriába. Az olyan szerelvényeket, amelyeket a memóriában definiálunk és futtatunk, *dinamikus szerelvényeknek* nevezzük.

## Második kötet

## 5. rész: Bevezetés a .NET alaposztálykönyvtáraiba

Ha elérkeztünk ehhez a részhez, akkor már biztos tudásunk van a C# nyelvről és a .NET- szerelvényformátumok részleteiről. Az ötödik rész továbbfejleszti ezeket az ismereteket azáltal, hogy számos általánosan használt alaposztálybeli szolgáltatást mutat be, többek között a fájl I/O-t és az adatbáziselérést az ADO.NET segítségével. Ez a rész tárgyalja továbbá az elosztott alkalmazások készítésének témakörét a Windows Communication Foundation (WCF) segítségével, valamint azokat a munkafolyamat-engedélyezett alkalmazások készítését, amelyek a Windows Workflow Foundation (WF) API-t használják.

## 20. fejezet: Fájlműveletek és elkülönített tárolás

A `System.IO` névtér segítségével lehetőségünk van a számítógép fájl- és könyvtárszerkezetének elérésére. Ebben a fejezetben megismerjük, hogyan kell programozottan könyvtárrendszert létrehozni (és törölni), valamint adatokat különböző folyamatba (fájlalapú, sztringalapú, memóriaalapú stb.) be-, illetve kimotozni. A fejezet utolsó része megvizsgálja az *elkülönített tárolást*, amelynek segítségével lehetőségünk van biztonságos helyen külön tárolni az adatokat függetlenül attól, hogy a célgép biztonsági beállításaitól. Hogy megértsük a `System.IO.Isolated` API bizonyos elveit, áttekintést kell kapnunk a kódhozzáférés-szabályozásról (Code Access Security – CAS).

## 21. fejezet: Bevezetés az objektumsorosítás világába

Ez a fejezet a .NET-platform objektumsorosító szolgáltatásába nyújt betekintést. Egyszerűen szólva a *sorosítás* segítségével lehetőségünk nyílik arra, hogy egy objektum (vagy a kapcsolódó objektumok egy halmazának) állapotát egy folyamban tároljuk a későbbi használat céljából. A *deszerializálás* (a sorosítás ellentétes művelete) az a folyamat, amelynek segítségével kivehetjük az objektumot a folyamból, és a memóriában az alkalmazás számára használható formába hozhatjuk. Az alapok elsajátítása után megtanulhatjuk, hogyan kell egyedivé tenni a sorosítási folyamatot az `ISerializable` interface és a .NET-attribútumok halmazának segítségével.

## 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat

Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kaptunk ismereteket. Ez a rész bevezeti a .NET adatszolgáltató szerepét, valamint azt, hogy hogyan kell a relációs adatbázissal kommunikálni az ADO.NET *élő kapcsolatán* keresztül. Ez a kapcsolati objektumokat, a parancsobjektumokat és az adatolvasó objektumokat jelenti. Ezenkívül ez a fejezet mutatja be az egyedi adatbázis létrehozásának módját, és ismerteti azokat az adatelérési könyvtárakat, amelyeket majd a könyv további részében használunk.

## 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat

Ez a fejezet az adatbázis-kezelés különböző módokon történő manipulálási lehetőségeit tárgyalja tovább, jelen esetben az ADO.NET *bontott kapcsolaton* keresztül. Megismerkedhetünk a `DataSet` típus és az adatfeldolgozó objektumok szerepével, valamint számos Visual Studio 2008 eszközzel, amelyek nagyban megkönnyítik az adatvezérelt alkalmazások létrehozását. Mindeközben rávilágítunk arra, hogyan kell a `DataTable`-objektumokat felhasználni felületi elemekhez kapcsolni, olyanokhoz, mint például a Windows Forms `API GridView` típusa.

## 24. fejezet: A LINQ API programozása

A 24. fejezet a LINQ programozási modellt mutatja be, pontosabban a LINQ „az objektum irányába” részét. Megvizsgáljuk, hogyan kell LINQ-lekérdezéseket alkalmazni relációs adatbázisokon, `DataSet`-objektumokon és XML-dokumentumokon. Mindeközben megtanuljuk az adatkontextus-objektum, valamint az `sqlmetal.exe` segédprogram szerepét, és megvizsgáljuk a Visual Studio 2008 különböző LINQ-támogató funkcióit.

## 25. fejezet: A WCF

A .NET 3.0 bevezetett egy teljesen új API-t, a WCF-et, amely lehetőséget ad arra, hogy szimmetrikus módon, függetlenül az alatta található rétegek felépítésétől elosztott alkalmazást fejlesszünk. Ez a fejezet feltárja a WCF-szolgáltatások, -gazdagépek és -kliensek készítésének lehetőségeit. A WCF-szolgáltatások rendkívül flexibilisek, a gazdagépek és a kliensek XML-alapú konfigurációs fájlokat használhatnak, hogy deklaratív módon adjanak meg címeket, kötéseket és szerződéseket.

## 26. fejezet: A WF

A .NET 3.0 a WCF mellett bevezette a WF API-t is, amely lehetőséget ad arra, hogy *munkafolyamatokat* definiáljunk, futtassunk és monitorozzunk. Ennek segítségével komplex üzleti folyamatok modellezhetők. Megtanuljuk a WF általános célját, ahogy azt is, hogy mi a szerepe az aktivitásoknak, a munkafolyamat tervezőnek, a munkafolyamat futtatási motornak, és a munkafolyamat-engedélyezett forráskód könyvtáraknak.

## 6. rész: Felhasználói felületek

Általánosan hibás feltételezés a .NET platformmal kapcsolatban az, hogy ez a keretrendszer kizárólag a webalapú felhasználói felületekkel foglalkozik (ez feltehetően a „.NET” kifejezés miatt van, hiszen felidézi az „Internet” szót, azaz a „webprogramokat”). A .NET kiemelkedő módon támogatja a webalkalmazások készítését, a könyv ezen része azonban a hagyományos felhasználói felületekre koncentrál két grafikus felület keretrendszerén, a Windows Forms és a WPF használatán keresztül.

## 27. fejezet: Windows Forms-programozás

Az eredeti asztali grafikus felhasználói felület, amely a .NET platformmal együtt kapható, a *Windows Forms*. Ez a fejezet ismerteti a felhasználói felület keretrendszerének szerepét, és bemutatja, hogyan kell főablakokat, párbeszédablakokat és menürendszereket készíteni. Feltárja továbbá az űrlapszármaztatás szerepét, és azt, hogy hogyan kell kétdimenziós adatot rajzolni a *System.Drawing* névtér segítségével. A témakör illusztrálására a fejezet végén készítünk egy (korlátozott funkcionalitással bíró) rajzolóalkalmazást.

## **28. fejezet: A WPF és az XAML**

A .NET 3.0 teljesen új grafikus felhasználói felületet vezetett be, amelyet WPF-nek nevezett el. Röviden, a WPF kimagaslóan interaktív és médiában gazdag front endek készítését teszi lehetővé asztali alkalmazásokhoz (és indirekt módon webalkalmazásokhoz). A Windows Formsszal ellentétben, ez a túltelített felhasználófelület-keretrendszer számos kulcsfontosságú szolgáltatást (2D-s és 3D-s grafika, animációk, gazdag dokumentumok stb.) integrál egyetlen egységesített objektummodellbe. Ebben a fejezetben megkezdjük az ismerkedést a WPF-fel és a kiterjeszthető alkalmazásjelölő nyelvvel (Extendable Application Markup Language – XAML). Megtanuljuk, hogyan kell XAML-mentes, csak XAML-t használó és a kettő kombinációjából felépülő WPF-programokat létrehozni. A fejezet végén készítünk egy egyedi XAML-nézegetőt, amelyet a további WPF-fel foglalkozó fejezetekben is használni fogunk.

## **29. fejezet: Programozás WPF-vezérlőelemekkel**

Ebben a fejezetben megtudjuk, hogy hogyan kell dolgozni a WPF-vezérlőelem tartalommodelljével, és érintünk számos vezérlőelemekkel kapcsolatos témát is, például a függőségi tulajdonságokat és irányított eseményeket. Jó néhány WPF-vezérlőelem használatát bemutatjuk, s szintén ebben a fejezetben magyarázzuk el az elrendezéskezelő, -vezérlő utasítások és a WPF-adatkötés modelljének a használatát.

## **30. fejezet: WPF 2D grafikus renderelés, erőforrások és témák**

Az utolsó fejezet a WPF tárgyalását három látszólag független téma vizsgálatával zárja. A WPF grafikus renderelő szolgáltatásának általában szüksége van arra, hogy egyedi erőforrásokat definiáljunk hozzá. Ezeknek az erőforrásoknak a használatával lehetőségünk van egyedi WPF-animációk készítésére, és grafikát, erőforrásokat és animációkat alkalmazva egyedi témák készítését fogjuk tudni megoldani WPF-alkalmazások számára. A különböző témák összegzéseként a fejezet utolsó részében azt illusztráljuk, hogyan kell futásidőben egyedi grafikus témákat alkalmazni.

## 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével

A 7. rész az ASP.NET programozási API használatával készített webes alkalmazásokat vizsgálja. Az ASP.NET szándékosan úgy lett kialakítva, hogy az asztali felhasználói felület létrehozását modellezze, úgy, hogy szabványos HTTP kérésekre/válaszokra helyez egy eseményvezérelt, objektumorientált keretrendszert.

### 31. fejezet: ASP.NET weboldalak készítése

Ez a fejezet vezet be bennünket az ASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltják az igazi objektumorientált nyelvek (mint a C#, VB.NET és hasonlók). Bemutatjuk egy ASP.NET weboldal készítését, valamint az alatta található programozási modellt és az ASP.NET egyéb kulcsfontosságú elveit, például azt, hogy hogyan kell webkiszolgálót választani, valamint a `web.config` fájlok használatát.

### 32. fejezet: ASP.NET-vezérlőelemek, témák és mesteroldalak

Ez a fejezet azokkal a vezérlőelemekkel foglalkozik, amelyek a belső vezérlőfát töltik fel adattal. Megismerjük az alapvető ASP.NET webes vezérlőelemeket, többek között a ellenőrző vezérlőelemet, a belső oldali navigációs vezérlőelemeket és a különböző adatkötő műveleteket. Ugyancsak ennek a fejezetnek a feladata, hogy bemutassa a *mesteroldalak* és az ASP.NET témamotorjának a szerepét, amely a hagyományos stíluslapok szerveroldali megfelelője.

### 33. fejezet: ASP.NET állapotkezelési technikák

Ez a fejezet kiterjeszti az eddigi ASP.NET-ismereteinket, ugyanis megvizsgálja, milyen különböző módokon lehet állapotot kezelni a .NET alatt. Ahogy a klasszikus ASP, az ASP.NET is egyszerűen teszi lehetővé sütik létrehozását, ahogy alkalmazásszintű és munkafolyamat-szintű változók használatát is. Az ASP.NET azonban bevezet egy új állapotkezelő technikát: az alkalmazás-gyorsítótárat. Ha már tudjuk, milyen módokon lehet állapotot kezelni az ASP.NET alatt, megismerkedünk a `System.HttpApplication` alaposztály szerepével (a `Global.asax` fájlban belül), valamint azzal, hogy hogyan kell dinamikusan megváltoztatni a webalkalmazásunk futásidejű viselkedését a `web.config` fájl segítségével.

## 8. rész: Függelék

A könyv utolsó része két fontos témát vizsgál meg, amelyek valójában nem illeszkedtek szervesen a könyv felépítésébe, ezért a függelékbe kerültek. A függelék kiteljesíti a C#-pal és .NET platformmal kapcsolatos ismereteinket, megvizsgáljuk ugyanis, hogyan kell régebbi kódokat integrálni a .NET-alkalmazásainkba, valamint hogyan kell a .NET-fejlesztést a Windows operációs rendszer családján kívül használni.

### A függelék: A COM és a .NET együttműködése

Azok, akik már programoztak Windows operációs rendszer alatt a .NET platform használata előtt, nagy valószínűséggel ismerik a Component Object Modelt (COM). Noha a COM-nak és a .NET-nek semmi köze egymáshoz (azon túl, hogy mindkettő a Microsoft szerzeménye), a .NET platform teljes névteret biztosít ahhoz (`System.Runtime.InteropServices`), hogy .NET-szoftverek COM-komponenseket használjanak, és fordítva. Ez a függelék az együttműködési réteget mutatja be egészen részletesen; ez a téma azért nagyon fontos, hogy a korábban megírt programjainkat át tudjuk menteni az újonnan készített .NET-alkalmazásainkba.

### B függelék: Platformfüggetlen .NET-fejlesztés a Monóval

Végül, de nem utolsó sorban a B függelék a .NET nyílt forráskódú implementációjával, a *Monóval* foglalkozik. A Mono segítségével lehetőségünk van gazdag tulajdonságokkal rendelkező .NET-alkalmazás létrehozására, telepítésére és futtatására különböző operációs rendszereken, mint például a Max OS X, Solaris, AIX és számos Linux-disztribúció. Mivel a Mono nagyjából kompatibilis a Microsoft .NET platformjával, így már tisztában vagyunk azazal, hogy mit tud nyújtani számunkra. Ezért a függelék a Mono installálását tárgyalja, valamint azt, hogy, milyen fejlesztői eszközök állnak a rendelkezésünkre, és hogyan működik a Mono futtatómotorja.

## Öt szabadon letölthető fejezet – még több információ

Azoknak, akiknek a 33 fejezet és a két függelék nem lenne elég, szabadon letölthetnek további öt fejezetet. A könyv korábbi verziói három olyan fejezetet tartalmaztak, amelyeket a Windows Forms fejlesztésének szenteltünk (egyedi vezérlők vizsgálatával egyetemben), egy másik fejezetet a .NET remoting rétegének (System.Runtime.Remoting és társai), és a végül egy fejezet a hagyományos XML-webszolgáltatások készítésével foglalkozott az ASP.NET Web Service projekt sablonjainak a segítségével.

A könyv jelen kiadása nem tartalmazza ezt az öt fejezetet. Ennek az a legfőbb oka, hogy a .NET 3.0-ban a WCF és a WFP API-k rendre a .NET-remoting/XML-webszolgáltatások és a Windows Forms API-k örökösei. Ha mélyebben szeretnénk elmerülni a Windows Forms világában (azon túl, amit a 27. fejezet nyújtani tud), vagy meg akarjuk nézni, hogyan kell használni a (örökségül hagyott) .NET-remotingot és az XML-webszolgáltatás-API-t, egyszerűen csak meg kell nézni az Apress honlapján a megfelelő részt:

<http://apress.com/book/view/1590598849>.

Az itt található linkre kattintva digitális formában letölthetjük a könyv további fejezeit, ha sikertült válaszolni a könyv szövegéből véletlenszerűen feltett kérdésekre. (A letölthető fejezetek csak angol nyelven érhetők el – a SZAK Kiadó megjegyzése).

## A könyv forráskódjának igénylése

A könyvben szereplő összes forráskód-mintapélda (az öt szabadon letölthető további fejezetben levők is) szabadon és azonnal elérhető az Apress honlapjának Source Code/Download részlegéről. Egyszerűen gépeljük be a <http://www.apress.com> címet, válasszuk ki a Source Code/Download linket, és keressük meg a megfelelő címet, és, töltsük le az önmagát kicsomagoló \*.zip fájlt. A forráskódokat fejezetenként osztottuk fel.

Figyelem: a fejezetekben a Forráskód megjelölés, mint amilyen az alábbi is, arra utal, hogy a szóban forgó példákat be lehet tölteni a Visual Studio 2008-ba további vizsgálódások és módosítások céljából:

---

**Forráskód** Ez a forráskód-megjelölés egy kiválasztott könyvtárra hivatkozik.

---

Egyszerűen nyissuk meg a \*.sln fájlt a megfelelő alkönyvtárból. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet további integrált fejlesztői környezetekért), akkor manuálisan töltjük be a felkínált forráskódfájlokat a választott fejlesztői eszközbe.

---

**Megjegyzés** A magyar nyelvű változatban a forráskódok megjegyzéseit lefordítottuk. A letölthető forráskódok fordítása nem állt módunkban, így azok teljesen angol nyelvűek. Ilyen módon inkább megfelelnek a letölthető kódok használati céljának – a SZAK Kiadó megjegyzése.

---

## A lehetséges javítások

Elképzelhető, hogy a könyv végigolvasása során nyelvtani vagy forráskódbeli hibákat vél felfedezni a kedves Olvasó, bár remélem, hogy erre nem kerül sor. Ha mégis, elnézést kérek. Az aktuális hibalistát az Apress weboldalról lehet igényelni (amely szintén a könyv honlapján található meg), és ugyancsak itt található az elérhetőségem, amelyen értesíteni tudnak.

## Elérhetőségem

Ha bármilyen kérdés merül fel a könyvhöz tartozó forráskódokkal kapcsolatban, vagy szükség van valamelyik példa tisztázására, vagy egyszerűen csak véleményt szeretne nyilvánítani a .NET platformról, nyugodtan küldjön levelet a következő e-mail címre (hogy biztosak legyünk benne, hogy a levele nem kerül a kéretlen levelek közé, kérem, tegye a tárgymezőbe a „C#FE” szöveget): [atroelsen@Intertech.com](mailto:atroelsen@Intertech.com).

Annak ellenére, hogy megpróbálok minden levélre lehetőség szerint válaszolni, sajnos, ahogy mindnyájan, időnként nagyon elfoglalt vagyok. Ha nem válaszolok egy-két héten belül, akkor ez nem jelenti azt, hogy nem akarok foglalkozni a problémával, csak épp nagyon nem érek rá (vagy ha szerencsém van, valahol épp nyaralok).

Nos, akkor, köszönöm, hogy megvásárolta a könyvemet (vagy legalábbis, hogy belenéz a könyvesboltban, miközben azon gondolkodik, hogy megvegye-e). Remélem, élvezni fogja a kedves Olvasó a tanulmányozását, és hasznosítani tudja az újonnan szerzett ismereteket.

Minden jót!  
Andrew Troelsen



# 1. rész

## Bevezetés a C#-ba és a .NET platformba



## ELSŐ FEJEZET

# A .NET filozófiája

Ma már a programozók számára elengedhetetlen, hogy a néhány évente vagy akár még gyakrabban továbbképzéseken vegyenek részt, hiszen mindig naprakésznek kell lenniük a legújabb technológiákból. A szoftverfejlesztés gyöngyszemeinek kikiáltott nyelveket (C++, Visual Basic 6.0, Java), keretrendszereket (OWL, MFC, ATL, STL), architektúrákat (COM, COBRA, EJB) és API-kat (mint a .NET Windows-űrlapjai és a GDI+ könyvtárai) végül mindig háttérbe szorítja valami jobb, de legalábbis valami új. Bár a belső adatbázisunk frissítésekor nyilván csalódottságot érzünk, ám mindez elkerülhetetlen. Éppen ezért könyvünknek az a legfontosabb célja, hogy betekintsünk a Microsoft szoftverfejlesztéssel kapcsolatos aktuális kínálatának részleteibe: a .NET platformba és a C# programozási nyelvbe.

Ebben a fejezetben elsődleges feladatunk az, hogy tisztázzuk a könyv további részeinek megértéséhez szükséges alapfogalmakat. Köztük számos .NET-hez kapcsolódó téma: például a szerelvények, a köztes nyelvi kód (CIL) és a JIT-fordítás tárgyalását is nyomon követhetjük. A C# programozási nyelv legfontosabb sajátosságainak ismertetése mellett a .NET-keretrendszer különböző megjelenési formái – mint a nyelvek közös futtatórendszere (CLR), a közös típusrendszer (CTS) és a közös nyelvi specifikáció (CLS) – között létrejövő kapcsolatra is fény derül.

Ezeket kívül az olykor BCL-nek, esetleg FCL-nek (keretrendszer-osztálykönyvtár) rövidített .NET-alaposztálykönyvtárak funkcionalitását is megvizsgáljuk. Végezetül áttekintjük a .NET platform nyelvagnosztikus és platformfüggetlen természetét is. (Tudniillik a .NET nem korlátozódik a Windows operációs rendszerre.) Ezeket a témákat a továbbiakban részletesen is kifejthetjük.

## Az előzmények áttekintése

A .NET univerzum sajátosságainak számbavétele előtt hasznos lehet néhány olyan tényezőt összegezni, amely a Microsoft jelenlegi platformjának megalkotását előidézte. A tisztánlátáshoz kezdjük a fejezetet egy rövid, „fájdalommentes” történelemórával, hogy felelevenítsük az előzményeket, és megértjük azok korlátait. (Elvégre az első lépés a megoldáshoz vezető úton, ha beismerjük, hogy problémával állunk szemben.) Miután tettünk egy gyors körutazást a múltban, figyelmünket a C# és a .NET-platform számtalan előnyének a megismerésére fordíthatjuk.

### Egy C/Win32 API programozó élete

Azt szokták mondani, hogy a Windows operációs rendszerek családjába tartozó szoftverek fejlesztése a C programozási nyelv és a Windows alkalmazásprogramozói felületének (API) együttes használatából áll. Igaz, hogy számos alkalmazást sikeresen létrehoztak ezzel a hagyományos módszerrel, néhányan mégis vitatják, hogy az alkalmazások nyers API-val történő felépítése összetett feladat lenne.

Az első nyilvánvaló probléma az, hogy a C nagyon tömör nyelv. A C# fejlesztői kénytelenek kézi memóriakezeléssel, csúnya pointeraritmetikával és ugyancsak csúnya szintaktikai konstrukciókkal megküzdeni. Ráadásul, mivel a C strukturált nyelv, hiányzik belőle az objektumorientált megoldások előnye (valaki el tudja mondani, mi az a *spagettikód*?). Amikor a Win32 API segítségével definiált globális függvények és adattípusok ezreit hozzáadjuk egy eleve túl részletes nyelvhez, nem csoda, hogy olyan sok hibás alkalmazást kapunk.

### Egy C++/MFC-programozó élete

A csupasz C/API fejlesztésének egyik nagy horderejű újítása a C++ programozási nyelv használata volt. A C++ sok szempontból a C-re helyezett objektumorientált *rétegnek* tekinthető. Így annak ellenére, hogy a C++ programozóinak hasznára vannak az objektumorientált programozás közismert alappillérei (egységbezárás, származtatás és polimorfizmus), továbbra is ki vannak szolgáltatva a C nyelv nehézségeinek (pl. kézi memóriakezelés, csúnya pointeraritmetika és csúnya szintaktikai konstrukciók).

Összetettsége ellenére, manapság is sok C++-keretrendszer létezik. Például a Microsoft Foundation Classes (MFC) olyan C++-osztály-készletet biztosítanak a fejlesztők számára, amely megkönnyíti a Win32-alkalmazások felépítését. Az MFC legfőbb szerepe, hogy számos osztály, makró és kódgeneráló eszköz (varázsló) mögött a natív Win32 API egy részét beburkolja. Az MFC-keretrendszer hasznos támogatását figyelmen kívül hagyjuk (ugyanúgy, mint sok egyéb C++-alapú eszközrendszert), be kell látnunk, hogy a C++-programozás nehézkes, és több hibalehetőséget rejt magában, tekintve, hogy a C-ből ered.

## Egy Visual Basic 6.0 programozó élete

Az egyszerűsége törekvés jegyében sok programozó a C(++)-alapú keretrendszerekről olyan barátságosabb és kifinomultabb nyelvekre tért át, mint például a Visual Basic 6.0 (VB6). A VB6 annak köszönheti népszerűségét, hogy egyszerűen lehet vele összetett felhasználói felületeket, kódkönyvtárakat (pl. COM-kiszolgálók) és adatelérési logikát készíteni. A VB6 sokkal jobban elrejti szem elől a csupasz Win32 API bonyolultságát, mint az MFC, számos beépített forráskód-varázsló, valódi adattípusok, osztályok és VB-specifikus függvények segítségével.

A VB6 legnagyobb hibája (amelyet a .NET-platform megjelenése orvosolt), hogy nem teljesen objektumorientált nyelv, hanem inkább „objektumtudatos”. A VB6 például nem engedi a programozónak, hogy „ez egy...” („is-a”) kapcsolatokat hozzanak létre a típusok között (azaz nem engedi a klasszikus származtatást), és nincs belső támogatása paraméterezett osztályok készítéséhez. Ezenkívül a VB6 nem képes többszálú alkalmazások létrehozására sem, kivéve, ha alacsony szintű Win32 API hívásokat szeretnénk (ám ez legjobb esetben összetett, legrosszabb esetben pedig veszélyes).

## Egy Java/J2EE-programozó élete

A Java megérkezett. A Java objektumorientált programozási nyelv, amelynek szintaktikai gyökerei a C++-ból erednek. Amint azzal sokan tisztában vannak, a Java előnyei messze túlszárnyalják platformfüggetlenségi támogatását. A Java (mint nyelv) a C++ számos kellemetlen szintaktikai jellemvonását kiküszöbölte. A Java (mint platform) a programozók számára számtalan előre megadott „csomagot” biztosít, amelyek különféle típusdefiníciókat tartal-

maznak. Ezekkel a típusokkal a Java-programozók képesek 100%-ban tiszta Java-alkalmazásokat létrehozni, amelyek teljes adatbázis-összekapcsolhatósággal, üzenetközvetítési támogatással, webes ügyféloldallal és gazdag asztali felhasználói felülettel rendelkeznek.

Habár a Java nagyon elegáns nyelv, potenciális problémája, hogy használatakor jellemzően előről hátrafelé kell haladni a fejlesztés folyamán. A Java tulajdonképpen kismértékben nyújt nyelvi integritást, minthogy ez ellentmond elsődleges feladatának – a Java egy egyszerű, bármilyen célra felhasználható programozási nyelv. A valóságban azonban forráskódok milliónyi sora létezik, s ezeket előnyös lenne Java-kódból felhasználni. Sajnos a probléma megoldását a Java nem könnyíti meg. Amíg ugyanis korlátozott képességet mutat a nem Java-s API-k eléréséhez, addig csak kismértékben támogatja a valódi nyelvközi integrációt.

## Egy COM-fejlesztő élete

A Component Object Model (COM – komponensobjektum-modell) volt a Microsoft előző alkalmazásfejlesztési keretrendszere. A COM architektúrája lényegében azt fejezi ki, hogy „[h]a az osztályainkat a COM szabályai szerint építjük fel, *újrafelhasználható bináris kódblokkot* kapunk”.

Egy bináris COM-kiszolgálóban az a nagyszerű, hogy nyelvfüggetlen módon hozzá lehet férni. Így a C++-programozók olyan COM-osztályokat hozhatnak létre, amelyeket a VB6 felhasználhat. A Delphi-programozók használhatnak olyan COM-osztályokat, amelyeket C segítségével hoztak létre, és így tovább. A COM nyelvfüggetlensége azonban némileg korlátozott. Egy meglévő COM-osztályból például nem lehet újabbat származtatni (mivel a COM nem támogatja a klasszikus származtatást). Ehelyett, a sokkal nehezebb „van neki egy...” („has-a”) kapcsolat segítségével kell újra felhasználni a COM-osztály típusait.

A COM másik előnye az, hogy transzparens tárhellyel rendelkezik. Az olyan szerkezetek használatával, mint az alkalmazásazonosító (AppIDs), a csonk, a proxyk és a COM futási környezet, a programozóknak nem kell socketekkel, távoli eljáráshívásokkal és egyéb alacsony szintű részletekkel dolgozniuk. Nézzük meg például a következő VB6-COM-ügyfélkódot:

```
' A MyCOMClass típus bármely  
' COM-alapú nyelven megírható, és a hálózaton bárhol  
' elhelyezhető (helyi számítógépen is).  
Dim obj as MyCOMClass  
Set obj = New MyCOMClass      ' AppID használatának a helye.  
obj.DoSomeWork
```

Habár a COM-ot nagyon sikeres objektummodellnek is tarthatjuk, valójában elképesztően bonyolult (legalábbis addig, amíg nem fordítunk több hónapot az átkötések vizsgálatára – elsősorban C++-programozóként). A COM-binárisok fejlesztésének egyszerűsítésére számos COM-alapú keretrendszer létezik. Ilyen például az ATL (Active Template Library), amely olyan C++-osztályok, -sablonok és -makrók készletével szolgál, amelyek megkönnyítik a COM-típusok létrehozását.

Sok egyéb nyelv ugyancsak elrejt szem elől a COM-infrastruktúra nagy részét. A nyelvi támogatás azonban önmagában nem elég a COM komplexitásának elrejtéséhez. Még ha egy olyan viszonylag egyszerű COM-alapú nyelvet választunk, mint a VB6, akkor is meg kell küzdenünk a sérülékeny regisztrációs bejegyzésekkel és számos telepítéssel kapcsolatos problémával (gyűjtőnéven, némileg komikusan, *DLL-pokol*).

## Egy Windows DNA programozó élete

Tovább bonyolítja a helyzetet az internet. Az elmúlt néhány évben a Microsoft jó néhány további internetalapú szolgáltatást épített be az operációs-rendszer-családjába és termékeibe. Sajnos a COM-alapú Windows DNA segítségével történő webes alkalmazások létrehozása ugyancsak meglehetősen bonyolult.

Ez a bonyolultság részben annak a ténynek köszönhető, hogy a Windows DNA számos technológia és nyelv használatát igényli (ASP, HTML, XML, JavaScript, VBScript és COM[+], valamint olyan adatelérési API-t, mint az ADO). Problémát jelent, hogy szintaktikai szempontból a technológiák nagy része nem kapcsolódik egymáshoz. A JavaScript szintaxisa például sokban hasonlít a C-re, míg a VBScript a VB6 részhalmaza. A COM+-futtatókörnyezetre tervezett COM-kiszolgálók teljesen eltérő megjelenésűek, mint azok az ASP-oldalak, amelyek hívják őket. Mindez a technológiák teljesen zavaros keverékét eredményezi.

Ezenkívül, és ez talán sokkal fontosabb, minden nyelvnek és/vagy technológiának megvan a saját típusrendszere (és ez lehet, hogy semmiben sem hasonlít egy másikéra). Túl azon, hogy minden egyes API rendelkezik saját előre gyártott kódgyűjteménnyel, az alapvető adattípusokat nem lehet mindig ugyanúgy kezelni. A `COMBSTR` az ATL-ben nem teljesen ugyanaz, mint egy `string` a VB6-ban; egyiknek sincs semmi köze a `C char*` adattípusához.

## A .NET-megoldás

Az előző rövid történelemóra tanulsága az volt, hogy a Windows-programozók élete bizony nem egyszerű. A .NET-keretrendszer azonban radikálisan egyszerűsíti és megkönnyíti a programozó életét. A .NET a „Változtassunk meg mindent!” megoldást kínálja („elnézést kérek, de ne az üzenet közvetítőjét okoljuk az üzenetért”). Ahogy a könyv további részeiben látni fogjuk, a .NET-keretrendszer a rendszerfelépítés teljesen új modellje a Windows operációs rendszercsaládban éppúgy, mint számos nem Microsoft-alapú operációs rendszerben (mint például a Mac OS X és a különféle Unix-/Linux-termékek). Előzetesen íme a .NET néhány alapvető jellemzőjének összefoglalása:

- *Együttműködési képesség a meglévő forráskóddal.* Ez (természetesen) nagyon hasznos. A meglévő COM-binárisok keverednek (azaz együttműködnek) az újabb .NET bináris fájlokkal, és viszont. A `PLnvoke` lehetővé teszi a C-alapú könyvtárak (beleértve az operációs rendszer API) hívását a .NET-kódból.
- *Teljes nyelvi integráció.* A .NET támogatja a nyelvközi származtatást, a nyelvközi kivételkezelést és a nyelvközi hibakeresést.
- *Közös futtatórendszer az összes .NET-alapú nyelv számára.* Ez a rendszer definiál egy olyan típuskészletet, amelyet minden .NET-alapú nyelv megért.
- *Átfogó alaposztálykönyvtár.* Ez a könyvtár teszi lehetővé a natív API-hívások komplexitásának elkerülését, és olyan konzisztens objektummodellt kínál, amelyet az összes .NET-alapú nyelv használ.



- *Nincs több COM-átkötés.* Az `IClassFactory`-nak, az `IUnknown`-nak, az `IDispatch`-nek, az IDL-kód és a nem túl szimpatikus variant-kompatibilis adattípusoknak (`BSTR`, `SAFEARRAY` stb.) nincs helyük egyetlen .NET-binárisban sem.
- *Egy valóban leegyszerűsített telepítési modell.* .NET alatt nem szükséges bejegyezni a bináris egységeket a rendszerleíró adatbázisba. Ráadásul a .NET lehetővé teszi, hogy egyetlen gépen ugyanazon .dll több verziója legyenek jelen.

Ahogy a fenti felsorolásból látszik, a .NET-platformnak semmi köze a COM-hoz (azonfelül, hogy mindkét keretrendszer a Microsofttól származik). A .NET- és a COM-típusok valójában csak az együttműködési réteg segítségével tudnak kommunikálni.

---

**Megjegyzés** A .NET együttműködési réteg leírása a 2 kötet A függelékében található.

---

## Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS)

A .NET néhány előnyét már megismerve nézzük át azt a három alapvető, egymással összefüggő egységet, amelyek mindezt lehetővé teszik: a CLR-t, a CTS-t és a CLS-t. Egy programozó szemszögéből a .NET felfogható futtató-környezetként és átfogó alaposztálykönyvtárként. A futtató réteg más néven *közös nyelvi futtatórendszer*, vagyis a CLR. A CLR elsődleges feladata, hogy megkeresse, betöltse és kezelje helyettünk a .NET-típusokat. Ezenkívül olyan alacsony szintű részletekre is figyel, mint a memóriakezelés, alkalmazástartományok létrehozása, végrehajtási szálak és objektumkontextus-határok, valamint különböző biztonsági ellenőrzések végrehajtása.

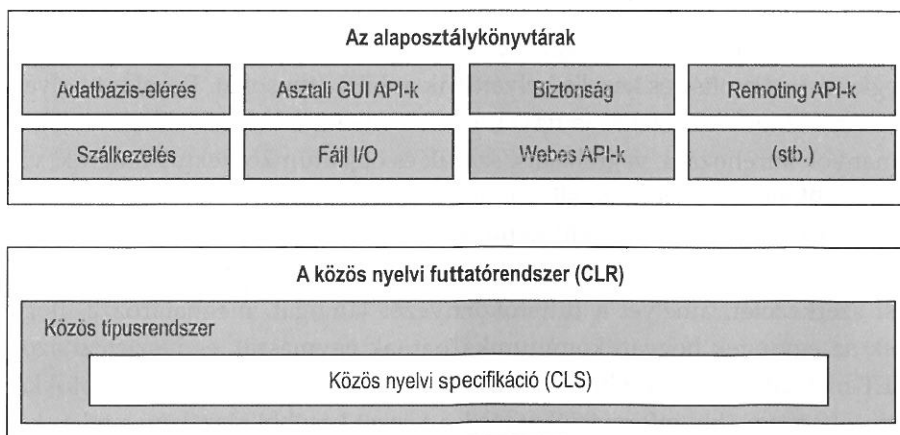
A .NET-platform másik építőeleme a *közös típusrendszer*, azaz a CTS. A CTS-specifikáció teljes körűen leírja az összes lehetséges adattípust és programozási szerkezetet, amelyet a futtatókörnyezet támogat, meghatározza, hogy ezek az egységek hogyan kommunikálhatnak egymással, és részletezi azok .NET-metaadat-formátumban való megjelenésének módját (a metaadatokkal kapcsolatos további információkat lásd a fejezet későbbi részében; a teljes körű leírást pedig lásd a 16. fejezetben).

Egy adott .NET-alapú nyelv nem szükségszerűen támogat minden egyes CTS-ben definiált funkciót. A *közös nyelvi specifikáció* (CLS) egy olyan kapcsolódó specifikáció, amely definiálja azoknak a közös típusoknak és programozási szerkezeteknek a részhalmazát, amelyeket minden .NET programozási nyelv elfogad. Ezért, ha olyan .NET-típusokat építünk fel, amelyek csak CLS-kompatibilis funkciókkal rendelkeznek, nyugodtak lehetünk afelől, hogy az összes .NET-alapú nyelv használni tudja azokat. Ellenben, ha olyan adattípus vagy programozási szerkezetet használunk, amely a CLS határain kívül esik, nem biztos, hogy az összes .NET programozási nyelv képes lesz kommunikálni a .NET-kódkönyvtárunkkal.

## Az alaposztálykönyvtárak szerepe

A CLR- és CTS/CLS-specifikációkon kívül, a .NET-platform az összes .NET programozási nyelv számára biztosít egy alaposztálykönyvtárat. Ez az alaposztálykönyvtár nemcsak beágyaz különböző primitíveket, mint a végrehajtási szálak, fájl input/output (I/O), grafikus renderelés és különféle külső hardvereszközökkel történő kommunikáció, de számos, leginkább az élet-szerű alkalmazások igényeihez mért szolgáltatást is támogat.

Az alaposztálykönyvtárak például olyan típusokat definiálnak, amelyek megkönnyítik az adatbázis-hozzáférést, az XML-dokumentumok manipulációját, a programozott adatvédelmet és a webes (csakúgy, mint a hagyományos asztali és konzolos) front endek létrehozását. A CLR, CTS, CLS és az alaposztálykönyvtár közötti kapcsolatot, az 1.1. ábra szemlélteti.



1.1. ábra: A CLR, CTS, CLS és az alaposztálykönyvtár kapcsolata

## A C# előnyei

Mivel a .NET radikálisan eltér a korábbi technológiáktól, a Microsoft kifejezetten ehhez az új platformhoz hozott létre egy új programozási nyelvet, a C#-t (kiejtve: [si: sa:p]). A C# alapszintaxisa *nagyon* hasonlít a Java szintaxisára. A C# azonban nem a Java egyik módosulata. Mind a C#, mind a Java a C programozási nyelv családjába (C, Objective C, C++ stb.) tartozik, ezért hasonló szintaxissal rendelkeznek. Ahogy a Java sok tekintetben a C++ letisztult verziója, a C#-ot ugyanígy a Java letisztult verziójának tekinthetjük.

Ettől függetlenül az igazság az, hogy a C# szintaktikai felépítése nagyrészt a Visual Basic 6.0 és a C++ alapján készült. Ahogy például a VB6, a C# is támogatja a tulajdonságok létrehozását (ellentétben a tradicionális getter és setter metódusokkal) és a metódusok deklarálásának képességét változó számú argumentummal (argumentumtömbök révén). A C++-hoz hasonlóan, a C# lehetővé teszi az operátorok túlterhelését, valamint struktúrák, felsorolt típusok és visszahívható függvények létrehozását (metódusreferenciák révén).

---

**Megjegyzés** Ahogy a 13. fejezetben látni fogjuk, a C# 2008 számos olyan konstrukciót átvett, amelyek eredetileg különböző funkcionális nyelvekből (pl. LISP vagy Haskell) származnak. Ráadásul, a LINQ megjelenésével (további információkért lásd a 14. és a II. kötetben található 24. fejezetet) a C# több olyan programozási konstrukciót támogat, amelyek meglehetősen egyedivé teszik a programnyelvek palettáját. Mindamellett probléma a C#-pal, hogy valóban hatással voltak rá a C-alapú nyelvek.

---

Mivel a C# több nyelv keresztezéséből született, szintaktikailag legalább annyira, ha nem még inkább letisztultnak tekinthető, mint a Java, olyan egyszerű, mint a VB6, és annyira hatékony és rugalmas, mint a C++ (a vele járó apró csúnyaságok nélkül). Íme, egy lista a C# alapvető tulajdonságairól, amelyek a nyelv valamennyi verzióját jellemzik:

- Nincs szükség mutatókra. A C#-programoknak jellemzően nincs szükségük közvetlen mutatómanipulálásra (de, ha mindenképpen szükséges, le lehet erre a szintre menni).
- Automatikus memóriakezelés a szemétygyűjtésen keresztül. Így a C# nem támogatja a `delete` kulcsszót.
- Formális szintaktikai konstrukcióként megjelenik az osztály, az interfész, a struktúra, a felsorolás és a metódusreferenciák.

- A C++-hoz hasonló operátor-túlterhelés az egyedi típusok esetében, csak egyszerűbben (pl. a meggyőződés arról, hogy a „return \*this to allow chaining” nem a mi problémánk).
- Az attribútumalapú programozás támogatása. Az ilyen típusú fejlesztés lehetővé teszi, hogy a típusokat és a tagjaikat magyarázatokkal lássuk el viselkedésük további leírásához.

A .NET 2.0 verziójával (kb. 2005-től) a C# programozási nyelv számos újdonsággal frissült, amelyek közül az alábbiak a legfontosabbak:

- Generikus típusok és tagok felépítésének a képessége. A generikus típusok használatával rendkívül hatékony és típusbiztos forráskódokat építhetünk fel, amelyek a generikus elemmel való kommunikáció során meghatározott helyőrzőket definiálnak.
- Névtelen metódusok támogatása, amelyek lehetővé teszik, hogy bárhol, ahol metódusreferenciára van szükség, inline függvényeket alkalmazzunk.
- A metódusreferencia/esemény modell számtalan egyszerűsítése, köztük kovariancia, kontravariancia és metóduscsoport-átalakítás.
- Egyetlen típus több kódfájlon keresztüli definiálásának képessége (vagy ha szükséges, memórián belüli ábrázolása) a partial kulcsszó segítségével.

A .NET 3.5 egyértelműen még több funkcionalitással látta el a C# programozási nyelvet (pontosabban a C# 2008 verziót), köztük az alábbiakkal:

- A különféle adatformátumokkal való kommunikációhoz szükséges erősen típusos lekérdezések (LINQ) támogatása.
- Névtelen típusok támogatása, amellyel sokkal inkább a típusok „alakját”, mint viselkedését modellezhetjük.
- Létező típusok funkcionalitásának kiterjesztése, bővítő metódusok segítségével.
- A lambda operátor (=>) bevezetése, amely még inkább leegyszerűsíti a .NET-metódusreferencia-típussal végzett munkát.
- Új objektuminicializáló szintaxis, amely lehetővé teszi a tulajdonságértékek beállítását az objektum létrehozásakor.

A C# nyelvvel kapcsolatban talán az a legfontosabb, hogy csak olyan forráskódot tud létrehozni, amely .NET-futtatókörnyezetben működik (C# segítségével nem tudunk natív COM-kiszolgálót vagy nem felügyelt Win32 API alkalmazást felépíteni). A .NET-futtatókörnyezethez készült forráskód leírására hivatalosan a *felügyelt programkód* kifejezést használjuk. A felügyelt programkódot tartalmazó bináris egységet *szerelvénynek* nevezzük. (A szerelvényekkel kapcsolatos további információkért lásd a „.NET-szerelvények áttekintése” című részt alább.) Azt a forráskódot ellenben, amelyiket a .NET-futtatókörnyezet közvetlenül nem tudja hosztolni, *nem felügyelt programkódnak* nevezzük.

## További .NET-alapú programozási nyelvek

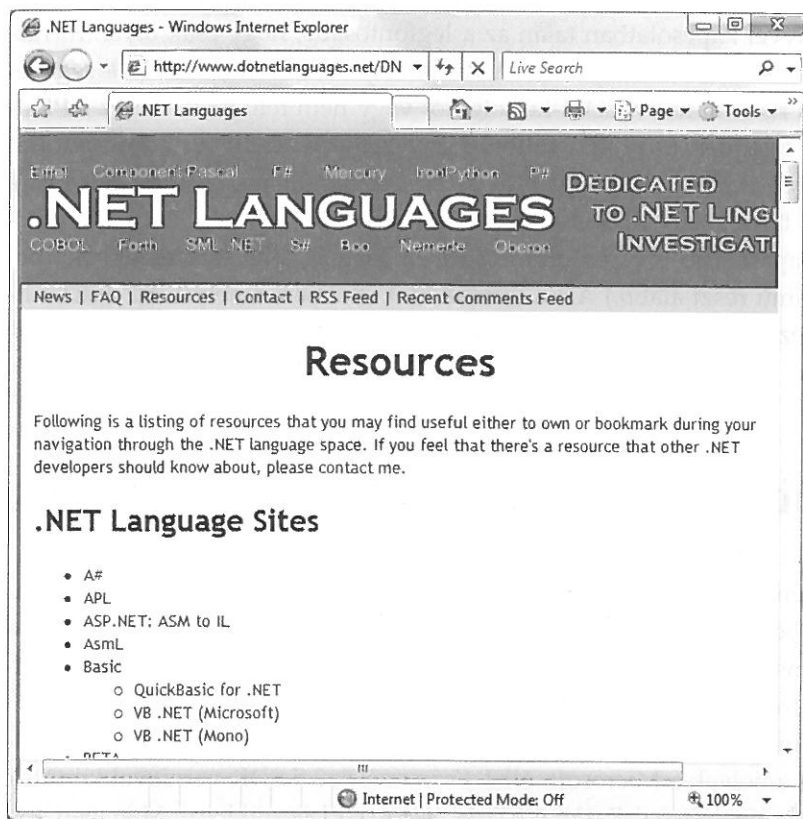
.NET-alkalmazásokat nem csak a C# nyelv segítségével építhetünk fel. Amikor 2000-ben, a Microsoft fejlesztői konferenciáján először bemutatták a nagyközönségnek a .NET-platformot, több gyártó is bejelentette, hogy felépítik a saját fordítóprogramjuk .NET-alapú verzióját.

Azóta nyelvek tucatjai estek át a .NET-felvilágosodáson. Azon az öt nyelven kívül, amelyek a Microsoft .NET Framework 3.5 SDK verzióval kerültek forgalomba (C#, Visual Basic .NET, J#, C++/CLI [korábban „Managed Extensions for C++”] és JScript .NET), léteznek .NET-fordítóprogramok Smalltalkra, COBOL-ra és Pascalra is. Jóllehet könyvünk (szinte) kizárólag csak a C#-pal foglalkozik, érdemes ellátogatni a következő webhelyre (a Microsoft fenntartja az URL változtatásának jogát):

<http://www.dotnetlanguages.net>.

Ha a főoldal tetején lévő Resources hivatkozásra kattintunk, számtalan .NET programozási nyelvet és azokhoz kapcsolódó hivatkozásokat tartalmazó listát kapunk, ezekre kattintva pedig különféle fordítókat tölthetünk le (lásd 1.2. ábra).

Ha elsősorban a C# szintaxisa alapján szeretnénk .NET-programokat létrehozni, tanácsos felkeresni ezt a webhelyet, itt biztosan találunk ugyanis a célnak megfelelő .NET-nyelvet (esetleg a LISP .NET-et?).



1.2. ábra: A [www.DotNetLanguages.net](http://www.DotNetLanguages.net) az egyik olyan webhely, amely ismert .NET programozási nyelveket dokumentál

## Élet egy többnyelvű világban

Amikor a fejlesztők ráébredtek a .NET nyelvagnosztikai természetére, számos kérdés merült fel. Talán az egyik leggyakoribb kérdés ez lehetett: „Ha az összes .NET-nyelv felügyelt programkódot fordít, miért van szükség több fordítóra?”. Ezt a kérdést többféleképpen meg lehet válaszolni. Először is, mi programozók *nagyon* válogatósak vagyunk, ha bármilyen programozási nyelv kiválasztásáról van szó. Néhányan a pontosvesszős és kapcsos zárójeles nyelveket szeretik, a lehető legkevesebb nyelvi kulcsszóval. Mások azokat a nyelveket szeretik, amelyek az ember számára olvashatóbb szintaktikai tokeneket tartalmaznak (mint a Visual Basic). Megint mások pedig a nagyszámítógépes tapasztalataikat szeretnék érvényesíteni, amíg áttérnek a .NET-platformra (a COBOL .NET-en keresztül).

Legyünk őszinték. Ha a Microsoft létrehozott volna egy „hivatalos” .NET-nyelvet, amely a BASIC nyelvcsaládból származik, vajon az összes programozó elégedett volna? Vagy, ha az egyetlen „hivatalos” .NET-nyelv a Fortran szintaxisán alapulna, nyilván sokan lennének, akik a .NET-et végképp semmibe vennék. Mivel a .NET-futtatókörnyezetben nem fontos, hogy milyen nyelv segítségével építették fel a felügyelt kódot, a .NET-programozók hűek maradhatnak ahhoz a szintaxishoz, amelyet korábban választottak, és a lefordított szerelvényeket probléma nélkül megoszthatják a csapattagokkal, a részlegekkel és a külső szervezetekkel (tekintet nélkül arra, hogy a másik fél milyen .NET-nyelvet használ).

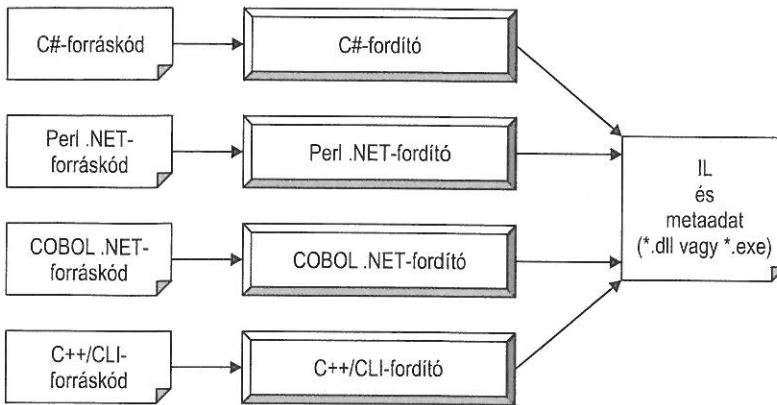
A különféle .NET-nyelveknek egyetlen, egységes szoftvermegoldásba történt integrálásának másik előnye, hogy minden egyes programozási nyelvnek megvannak a maga erősségei és gyengeségei. Egyes programozási nyelvek például remekül támogatják a fejlett matematikai feldolgozást. Mások pedig elsősorban a pénzügyi számításokat, a logikai számításokat, a nagyszámítógépekkel való kommunikációt stb. Ha az egyik programozási nyelv erősségeit vesszük, majd azokat egyesítjük a .NET-platform előnyeivel, voltaképpen mindenki jól jár.

A valóságban persze elég nagy az esély arra, hogy a szoftverek felépítését a kiválasztott .NET-nyelv segítségével fogjuk végezni. Ha azonban egy .NET-nyelv szintaxisát megismerjük, nagyon könnyű lesz egy másikét is megtanulni. Ez szintén meglehetősen hasznosnak tűnik, különösen a szoftvertanácsadóknak. Ha a választott nyelvünk például a C#, de egy olyan ügyfélnél dolgozunk, ahol csak Visual Basic .NET-et használhatunk, a .NET-keretrendszer funkcionalitását így is kihasználhatjuk, és az alapkód általános szerkezetét nagyobb nehézségek nélkül megérthetjük.

## A .NET-szerelvények áttekintése

Függetlenül attól, melyik .NET-nyelven programozunk, fogadjuk el, hogy noha a .NET bináris fájlok kiterjesztései megegyeznek a COM-kiszolgálók és a nem felügyelt Win32 bináris fájlok kiterjesztéseivel (\*.dll vagy \*.exe), nincs bennük semmiféle belső hasonlóság. A \*.dll .NET bináris fájlok például nem exportálnak metódusokat a COM-futtatókörnyezettel való kommunikációhoz (mivel a .NET *nem* COM). Továbbá a .NET bináris fájlok nem használnak COM-típuskönyvtárakat, és nincsenek bejegyezve a rendszerleíró adatbázisba.

De talán a legfontosabb, hogy a .NET bináris fájlok platformfüggő utasítások helyett platformfüggetlen *köztes nyelvet (IL)* és típusmetaadatokat tartalmaznak. Az 1.3. ábra az eddigieket mutatja be.



1.3. ábra: Az összes .NET-alapú fordító IL-utasításokat és metaadatokat ad ki

**Megjegyzés** Az „IL” rövidítést illetően meg kell említeni, hogy a .NET kifejlesztése során az IL helyett hivatalosan a Microsoft köztes nyelv (MSIL) kifejezést használták. A .NET végleges kiadásakor azonban már közös köztes nyelvként (CIL) említették. Ebből következően a .NET-irodalomban az IL, az MSIL és a CIL tehát pontosan ugyanarra utal. Az aktuális terminológiát követve a továbbiakban a „CIL” rövidítést fogjuk alkalmazni.

Amikor .NET-alapú fordító segítségével hozunk létre egy \*.dll vagy egy \*.exe kiterjesztésű fájlt, az eredményül kapott modul *szerelvényt* alkot. (A .NET-szerelvényeket részletesen a 15. fejezetben tárgyaljuk.) A .NET futtató környezet részletes tárgyalása előtt meg kell ismerkednünk az új fájlformátum néhány alapvető tulajdonságával.

A szerelvények tehát köztes nyelvi kódokat tartalmaznak, amelyek elvben a Java-bájtkódhoz hasonlítanak, hiszen addig nem változnak platformfüggő gépi utasítássá, amíg ez nem feltétlenül szükséges. A „feltétlenül szükséges” pont jellemzően az, amikor a .NET-futtatókörnyezet a CIL-utasításblokk (például egy metódusimplementáció) használatára hivatkozik.

A CIL-utasításokon kívül a szerelvények *metaadatokat* is tartalmaznak, amelyek a binárisban található típusok karakterisztikáját részletezik. Ha van például egy sportscar nevű osztályunk, a típusmetaadat olyan részleteket ír le, mint a sportscar ősosztálya, amelynek interfészeit (ha léteznek) a sportscar implementálja, valamint az összes olyan tagot, amelyeket a sportscar típusa támogat.



A .NET-metaadatok hatalmas előrelépést jelentenek a COM-metaadatokhoz képest. Talán már közismert, hogy a COM-binárisok jellemzően hozzárendelt típuskönyvtárakat használnak (ez valamivel több, mint az interfészleíró nyelv [IDL] forráskódjának bináris verziója). A COM-típusinformációkkal az a probléma, hogy nem biztos, hogy megvannak, illetve, hogy az IDL-kód nem tudja dokumentálni a kívülről hivatkozott kiszolgálókat, amelyek az aktuális COM-kiszolgáló megfelelő működéséhez szükségesek. Ezzel szemben a .NET-metaadatok mindig jelen vannak, és az adott .NET-alapú fordító automatikusan generálja őket.

Végezetül, a CIL és a típusmetaadatokon kívül maguk a szerelvények is használnak metaadatot, amelyet *manifesztumnak* nevezünk. A manifesztum a szerelvény aktuális változatáról szóló és kulturális információkat tartalmaz (a sztring- és a képerőforrások lokalizációjához), illetve egy listát, amely a megfelelő végrehajtáshoz szükséges összes külsőleg hivatkozott szerelvényt felsorolja. A következő néhány fejezetben a szerelvények típusainak, metaadatainak és önleíró adatainak vizsgálatához szükséges különféle eszközöket ismerhetjük meg.

## Egyfájlos és többfájlos szerelvények

Az esetek nagy többségében egyszerű egy-egy kapcsolat van a .NET-szerelvény és a bináris fájl (\*.dll vagy \*.exe) között. Ebből következően, ha egy .NET \*.dll létrehozásán dolgozunk, úgy vehetjük, hogy a bináris és a szerelvény egy és ugyanaz. Hasonlóképpen, ha egy végrehajtható asztali alkalmazást készítünk, az \*.exe fájl egyszerűen tekinthetjük magának a szerelvénynek. (Később azonban, a 15. fejezetben majd kiderül, hogy ez így nem pontos.) Szaknyelven, ha a szerelvény egy \*.dll vagy egy \*.exe modulból áll, azt *egyfájlos szerelvénynek* nevezzük. Az egyfájlos szerelvények az összes szükséges CIL-t, metaadatot és hozzárendelt manifesztumot önálló, egyedi és pontosan definiált csomagban tartalmazzák.

A *többfájlos szerelvények* viszont számos .NET bináris fájlból állnak, amelyeket *moduloknak* nevezünk. A többfájlos szerelvény esetében az egyik modul (*elsődleges modul*) tartalmazza a szerelvény manifesztumát (esetleg CIL-utasításokat és különféle típusok metaadatait). A többi modul egy modul-szintű manifesztummalsal, CIL-utasításokkal és típusmetaadatokkal rendelkezik. Nyilvánvaló, hogy az elsődleges modul dokumentálja a szükséges másodlagos modulok csoportját a szerelvény manifesztumában.

Miért hozzunk létre többfájlos szerelvényeket? Ha egy szerelvényt különálló modulokra osztunk fel, jóval rugalmasabb telepítési beállításokhoz jutunk. Ha például a felhasználó egy olyan távoli szerelvényre hivatkozik, amelyet szeretne letölteni a számítógépére, a futtatókörnyezet csak a szükséges modulokat fogja letölteni. Ezért felépíthetjük úgy is a szerelvényünket, hogy azokat a típusokat, amelyekre ritkábban van szükség (mint például a `HardDriverRef` nevű osztály), külön modul tárolja.

Ellenben, ha az összes típust egyfájlos szerelvénybe helyezzük, a végfelhasználó jó néhány felesleges adatot fog letölteni (és ez nyilvánvalóan időpocsékolás). Egy szerelvény tehát valóban egy vagy több kapcsolódó modul *logikai csoportosítását* tartalmazza, amelyeket önálló egységként telepítünk és módosítunk.

## A közös köztes nyelv (CIL) szerepe

A következőkben vizsgáljuk meg kicsit közelebbről a köztes nyelvi kódot, a típusmetaadatot és a szerelvény-manifesztumot. A CIL olyan nyelv, amely bármilyen platformfüggő gépiutasítás-készlet felett áll. A következő C#-kód például egy hétköznapi számológépet modellez. Ne foglalkozunk most a pontos szintaxissal, de figyeljünk a `Calc` osztály `Add()` metódusának formátumára:

```
// Calc.cs
using System;

namespace CalculatorExample
{
    // Az osztály tartalmazza az alkalmazás belépési pontját.
    class Program
    {
        static void Main()
        {
            Calc c = new Calc();
            int ans = c.Add(10, 84);
            Console.WriteLine("10 + 84 is {0}.", ans);

            // Várakozás, amíg a felhasználó megnyomja az Enter
            // billentyűt.
            Console.ReadLine();
        }
    }
}
```

```
// C#-számológép.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
}
```

Ha ezt a kódfájlt a C#-fordítóval (csc.exe) lefordítjuk, olyan egyfájlos \*.exe szerelvényt kapunk, amely tartalmaz manifesztumot, CIL-utasításokat és metaadatokat, és ezek minden szempontból leírják a Calc és a Program osztályt.

---

**Megjegyzés** A C#-fordító segítségével történő kódfordítást, illetve a grafikus IDE-k, például a Visual Studio, a Visual C# Express és a SharpDevelop, használatát a 2. fejezetben vizsgáljuk meg részletesen.

---

Ha ezt a szerelvényt például az ildasm.exe segítségével szeretnénk megnyitni (lásd alább), láthatjuk, hogy az Add() metódus köztes nyelven, a következőképpen jelenik meg:

```
.method public hidebysig instance int32 Add(int32 x,
int32 y) cil managed
{
    // A kód mérete 9 (0x9)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: nop
    IL_0001: ldarg.1
    IL_0002: ldarg.2
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007
    IL_0007: ldloc.0
    IL_0008: ret
} // a Calc::Add metódus vége
```

(Az eredményül kapott metódust a 19. fejezetben találjuk a CIL programozási nyelv alapjainak tárgyalásáról szóló részben.) A hangsúly most azon van, hogy a C#-fordító nem platformfüggetlen gépi utasításokat ad ki, hanem CIL-re fordít.

Mindez, ahogy láttuk, az összes .NET-alapú fordítóra igaz. Ennek illusztrálására, tételizzük fel, hogy ugyanezt az alkalmazást C# helyett Visual Basic .NET segítségével is létrehoztuk:

```
' Calc.vb
Imports System

Namespace CalculatorExample
' A VB "Module" is a class that contains only
' static members.
Module Program
    Sub Main()
        Dim c As New Calc
        Dim ans As Integer = c.Add(10, 84)
        Console.WriteLine("10 + 84 is {0}.", ans)
        Console.ReadLine()
    End Sub
End Module

Class Calc
    Public Function Add(ByVal x As Integer, ByVal y As Integer)
        As Integer
        Return x + y
    End Function
End Class
End Namespace
```

Ha megnézzük az Add() metódus köztes nyelvét, ott hasonló utasításokat találhatunk (a VB .NET-fordító [vbc.exe] által optimalizálva):

```
.method public instance int32 Add(int32 x, int32 y) cil managed
{
    // A kód mérete 8 (0x8)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: ldarg.1
    IL_0001: ldarg.2
    IL_0002: add.ovf
    IL_0003: stloc.0
    IL_0004: br.s IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // a Calc::Add metódus vége
```

---

**Forráskód** A Calc.cs és a Calc.vb kódfájlokat az 1. fejezet alkönyvtára tartalmazza.

---

## A köztes nyelv előnyei

Adódik a kérdés: miért volt jó, hogy köztes nyelvre fordítottuk a forráskódot, és nem közvetlenül egy specifikus utasításkészletbe. Ennek egyik oka a nyelvi integráció. Ahogy már láttuk, minden egyes .NET-alapú fordító közel azonos CIL-utasításokat hoz létre. Így az összes nyelv egy jól meghatározott bináris térben kommunikálhat egymással.

Ráadásul, mivel a CIL platformfüggetlen, és maga a .NET-keretrendszer is az, így a Java-fejlesztők számára már jól ismert előnyöket biztosítja (vagyis egy szimpla kódbázis fut több operációs rendszeren). A C# nyelvre létezik nemzetközi szabvány, és a .NET-platform, valamint implementációinak a nagy része már több nem Windows operációs rendszeren is rendelkezésre áll (további információkért lásd a fejezet későbbi részét). Ellentétben a Javával, a .NET lehetővé teszi, hogy a kívánt nyelven hozzunk létre alkalmazásokat.

### A köztes nyelv fordítása platformfüggetti utasítássá

Mivel a szerelvények inkább CIL-utasításokat, mintsem platformfüggetti utasításokat tartalmaznak, a köztes nyelvi kódot használat előtt, menet közben kell lefordítani. Azt az egységet, amely a CIL-kódot CPU-utasításokká alakítja, *azonnal (JIT) fordítónak* nevezzük, néha tréfásan *Jitternek*. A .NET-futtatókörnyezet minden futó processzor számára fenntart egy JIT-fordítót, s ezek mindegyikét az alapplatformra optimalizálták.

Ha például egy kézeszközbe (pl. Pocket PC-be) szánt alkalmazást készítünk, a megfelelő Jitter kitűnően fut a kevés memóriával rendelkező környezetben. Viszont, ha egy kiszolgálóra (ahol memóriában ritkán van hiány) telepítjük a szerelvényt, a Jittert a sok memóriával rendelkező környezethez optimalizáljuk. Elegendő tehát, ha a fejlesztők megírják azt a kódot, amely JIT-fordítással hatékonyan lefordítható, és a gépeken különböző architektúrákkal is végrehajtható.

Ráadásul, mivel a Jitterek a CIL-utasításokat megfelelő gépi kódokká fordítják, az eredményeket a memóriában a céloperációs rendszernek megfelelő formátumban gyorsítótárazzák. Így, ha a `PrintDocument()` metódust hívjuk meg, a CIL-utasításokat a rendszer az első híváskor platformfüggetti utasításokká fordítja, és a későbbi felhasználás érdekében a memóriában tárolja. Ebből következően, ha legközelebb meghívjuk a `PrintDocument()` metódust, a CIL-utasítást nem kell újra lefordítani.

---

**Megjegyzés** Azt is megtehetjük, hogy amikor a .NET Framework 3.5 SDK-ban lévő `ngen.exe` parancssori eszköz segítségével telepítjük az alkalmazást, végrehajtunk egy JIT-előfordítást a szerelvényen. Ezzel tökéletesíthetjük a grafikai intenzív alkalmazások indulási idejét.

---

## A .NET-típusmetaadat szerepe

A .NET-szerelvények a CIL-utasításokon kívül olyan teljes és szabatos metaadatot tartalmaznak, amely a binárisban definiált összes típust (osztály, struktúra, felsorolt típus stb.), valamint azok tagjait (tulajdonságok, metódusok, események stb.) is leírja. Szerencsére a legfrissebb és legnagyobb típusmetaadat előállítására mindig a fordító (és nem a programozó) feladata. Mivel a .NET-metaadat rendkívül aprólékos, a szerelvények teljesen önleíró egységek.

A .NET-típusmetaadat formátumának szemléltetésére, nézzük meg a korábban megvizsgált, C# `Calc` osztály `Add()` metódusa számára generált metaadatot (az `Add()` metódus VB.NET verziója számára generált metaadat is hasonló):

TypeDef #2 (02000003)

```
-----
TypeDefName: CalculatorExample.Calc (02000003)
Flags       : [NotPublic] [AutoLayout] [Class]
[AnsiClass] [BeforeFieldInit] (00100001)
Extends     : 01000001 [TypeRef] System.Object
Method #1 (06000003)
-----
```

```
MethodName: Add (06000003)
Flags       : [Public] [HideBySig] [ReuseSlot] (00000086)
RVA         : 0x00002090
ImplFlags   : [IL] [Managed] (00000000)
CallConvntn: [DEFAULT]
hasThis
ReturnType: I4
  2 Arguments
  Argument #1: I4
  Argument #2: I4
  2 Parameters
    (1) ParamToken : (08000001) Name : x flags: [none] (00000000)
    (2) ParamToken : (08000002) Name : y flags: [none] (00000000)
```

A metaadatot a .NET-futtatókörnyezet számos célra felhasználja ugyanúgy, ahogy a különféle fejlesztőeszközök. Az IntelliSense szolgáltatás például – ezt az olyan eszközök, mint a Visual Studio 2008 tartalmazzák – lehetővé teszi, hogy a szerelvények metaadata a tervezés folyamán olvasható legyen. A metaadatot különféle objektumböngésző segédprogramok, hibakereső eszközök és maga a C#-fordító is használja. Kétségtelen, hogy a metaadat a gerince számos .NET-technológiának, köztük a WCF-nek, az XML-webszolgáltatásoknak vagy a .NET remotingnak, a reflexiónak, a késői kötésnek és az objektumsorosításnak. (A 16. fejezet tartalmazza a .NET-metaadat szerepének formalizálását.)

## A szerelvény manifestumának szerepe

Végül, de nem utolsósorban, ahogy már szó volt róla, a .NET-szerelvények olyan metaadatot tartalmaznak, amely magát a szerelvényt írja le (szakszóval *manifestum*, *önleírás*). Egyéb részletek mellett, a manifestum rögzíti az aktuális szerelvény megfelelő működéséhez szükséges összes külső szerelvényt, a szerelvény verziószámát, a szerzői jogra vonatkozó adatokat stb. Ugyanúgy, mint a típusmetaadatot, a szerelvény manifestumát is a fordító generálja. Íme a fejezetben korábban bemutatott `calc.csc` kódfájl fordításakor generált manifestum néhány releváns részlete (tételezzük fel, hogy utasítottuk a fordítót, hogy a szerelvényt `calc.exe`-nek nevezze el):

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
.assembly calc
{
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module calc.exe
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
```

Röviden, a manifestum dokumentálja a `calc.exe` számára szükséges külső szerelvények listáját (az `.assembly extern` direktíva révén), valamint a szerelvény különböző jellemzőit (verziószám, modulnév stb.). (Az önleíró adatok hasznosságáról a 15. fejezetben lesz szó részletesebben.)

## A közös típusrendszer (CTS)

Egy adott szerelvény bármennyi különböző típust tartalmazhat. A .NET világában a *típus* általános kifejezés, amely a halmaz (osztály, interfész, struktúra, felsorolt típus, metódusreferencia) egy tagjára utal. Amikor egy .NET-alapú nyelv segítségével hozunk létre megoldásokat, valószínűleg sok ilyen típussal találkozunk. Például akkor, amikor a szerelvényünk olyan osztályt definiál, amely néhány interfészt implementál.

Előfordulhat, hogy az egyik interfészmetódus bemeneti paramétere egy felsorolt típus lesz, a hívónak pedig egy struktúrát ad vissza.

Ahogy láttuk, a közös típusrendszer (CTS) olyan formális specifikáció, amely azt dokumentálja, hogyan kell a típusokat úgy definiálni, hogy azokat a közös nyelvi futtatórendszer (CLR) befogadja. Jellemzően kizárólag azokat a fejlesztőket érdekli a közös típusrendszer belső működése, akik a .NET-platformot megcélzó eszközöket és/vagy fordítóprogramokat készítenek. Minden .NET-programozó számára fontos azonban, hogy megtanulja, hogyan kell a közös típusrendszer által definiált öt típussal bármilyen választott nyelven dolgozni. Az alábbiakban következzen egy rövid áttekintés.

## Az egységes típusrendszer osztálytípusai

Minden .NET-alapú nyelv támogatja legalább az *osztálytípus* fogalmát, amely az objektumorientált programozás (OOP) alappillére. Egy osztálynak bármennyi tagja (pl. tulajdonságok, metódusok és események) és adatpontja (mezője) lehet. A C#-ban a `class` kulcsszóval deklaráljuk az osztályokat:

```
// C# osztálytípus.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
```

A C# segítségével készült CTS-osztályok felépítését az 5. fejezetben vizsgáljuk meg; az 1.1. táblázatban felsoroljuk az osztálytípusok néhány jellemzőjét.

| Osztályjellemző                             | Valós jelentése                                                                                                                                                                                                                                     |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lezárt-e az osztály, vagy sem?              | A lezárt osztályok nem lehetnek más osztályok alaposztályai.                                                                                                                                                                                        |
| Megvalósít az osztály bármilyen interfészt? | Az <i>interfész</i> olyan absztrakt tagok gyűjteménye, amelyek kapcsolatot biztosítanak az objektum és az objektumfelhasználó között. Az egységes típusrendszer lehetővé teszi, hogy egy osztály vagy szerkezet bármennyi interfészt megvalósítson. |
| Az osztály absztrakt vagy valós?            | Az <i>absztrakt osztályokat</i> nem lehet közvetlenül létrehozni, de egységes viselkedést biztosítanak a származtatott típusoknak. A <i>valós osztályokat</i> közvetlenül létre lehet hozni.                                                        |



| Osztályjellemező                 | Valós jelentése                                                                                                                                                                                     |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Milyen az osztály „láthatósága”? | Minden osztályt egy láthatósági attribútummal kell konfigurálni. Alapvetően ez a jellemező határozza meg, hogy használhatják-e külső szerelvények is az osztályt, vagy csak a definiáló szerelvény. |

1.1. táblázat: Az egységes típusrendszer osztályjellemezői

## Az egységes típusrendszer interfésztípusai

Az *interfész* nem más, mint absztrakt tagok definícióinak megnevezett gyűjteménye, amelyet egy adott osztály vagy struktúra támogathat (megvalósíthat). Eltérően a COM-tól, a .NET-interfészek *nem* egy közös ősfinterfészből származnak (mint az IUnknown). A C#-ban az interfésztípusokat az `interface` kulcsszó segítségével definiálhatjuk:

```
// A C# interfész típust rendszerint publikus
// típusként deklaráljuk, így más szerelvények típusai
// is megvalósíthatják a viselkedését.
public interface IDraw
{
    void Draw();
}
```

Az interfészeknek önmagukban nincs sok hasznuk. Viszont, amikor egy osztály vagy struktúra megvalósít egy adott interfészt a maga egyedi módján, egy interfész-hivatkozás révén polimorf módon hozzáférhetünk a funkcionálitáshoz. (Az interfészalapú programozást a 9. fejezetben tárgyaljuk.)

## Az egységes típusrendszer struktúratípusai

A struktúra fogalmát ugyancsak formalizálták az egységes típusrendszerben. Ha a C-t ismerjük, akkor egyértelmű, hogy ezek a felhasználói típusok (UDT-k) a .NET világában is tovább élnek (bár egy kicsit másképp viselkednek). Egyszerűen fogalmazva, a *struktúrát* egy értékalapú szemantikával rendelkező pehelysúlyú osztálytípusnak is tekinthetjük. (A struktúrákkal kapcsolatos további részletekért, lásd a 4. fejezetet.) A struktúrák jellemzően geometriai és matematikai adatok modellezésére alkalmasak, és a C#-ban a `struct` kulcsszó segítségével lehet létrehozni őket:

```
// C# struktúratípus.
struct Point
{
    // A struktúrák mezőket tartalmazhatnak.
    public int xPos, yPos;

    // A struktúrák paraméterezett konstruktorokat tartalmazhatnak.
    public Point(int x, int y)
    { xPos = x; yPos = y; }

    // A struktúrák metódusokat definiálhatnak.
    public void Display()
    {
        Console.WriteLine("{0}, {1}", xPos, yPos);
    }
}
```

## Az egységes típusrendszer felsorolt típusai

A *felsorolt típus* olyan praktikus programozási szerkezet, amely lehetővé teszi a név/érték párok csoportosítását. Például tételezzük fel, hogy olyan videojáték-alkalmazást hozunk létre, amely lehetővé teszi a játékosok számára, hogy három karakterkategóriából válasszanak (Wizard – varázsló, Fighter – harcos vagy Thief – tolvaj). A numerikus értékek nyomon követése helyett, amelynek célja az egyes lehetőségek egyesével történő bemutatása lenne, egyedi felsorolt típusokat is létrehozhatunk, az enum kulcsszó segítségével:

```
// C# felsorolt típus.
enum CharacterType
{
    18 CHAPTER 1 n THE PHILOSOPHY OF .NET
    8849CH01.qxd 10/1/07 10:30 AM Page 18
    Wizard = 100,
    Fighter = 200,
    Thief = 300
}
```

Az alapértelmezés szerint minden egyes elemet 32 bites egész tárol; módosíthatjuk azonban a tárolót, ha szükséges (pl. amikor egy Pocket PC-hez hasonló, kevés memóriás eszközre programozunk). Az egységes típusrendszer emellett azt igényli, hogy a felsorolt típusok egy közös, `System.Enum` őssztályból származzanak. Ahogy a 4. fejezetből később kiderül: ez az alaposztály egy sor olyan érdekes tagot definiál, amelyek révén a mögöttes név/érték párokat programozással lehet kibontani, manipulálni és átalakítani.

## Az egységes típusrendszer metódusreferencia-típusai

A *metódusreferencia* a típusbiztos C-stílusú függvénymutató .NET-megfelelője. A legfőbb különbség az, hogy a .NET-metódusreferencia a `System.MulticastDelegate`-ből származó *osztály*, nem pedig egy nyers memóriacím egyszerű mutatója. A C#-ban a metódusreferenciákat a `delegate` kulcsszóval deklaráljuk:

```
// Ez a C# metódusreferencia típus bármely metódusra mutathat,  
// amely egész számot ad vissza, és két egész szám a bemenete.  
delegate int BinaryOp(int x, int y);
```

A metódusreferenciák egyrészt akkor nagyon hasznosak, ha valamelyik egyed számára lehetőséget szeretnénk teremteni arra, hogy egy másiknak hívást továbbítson, másrészt a .NET-eseményarchitektúra alapjainak megteremtésekor. A metódusreferenciák a többes küldéshez valódi támogatással rendelkeznek (pl. egy kérés több címzettnek való továbbításához), valamint az aszinkron függvényhíváshoz (részletesen lásd a 11. és 18. fejezetben).

## Az egységes típusrendszer típustagjai

Az egységes típusrendszerben formalizált összes típus közül a legtöbbször bármennyi *tagja* is lehet. Szaknyelven szólva, a *típustagot* a halmaz korlátozza (konstruktor, véglegesítő, statikus konstruktor, beágyazott típus, operátor, metódus, tulajdonság, indexelő, mező, írásvédett mező, konstans, esemény).

Az egységes típusrendszer különféle „díszítőelemeket” definiál, amelyeket hozzárendelhetünk egy adott taghoz. Minden tag rendelkezik például egy adott láthatósági jellemzővel (pl. nyilvános, privát, védett stb.). Néhány tag absztrakt tagként deklarálható, amely a származtatott típusokra polimorf viselkedést kényszerít, valamint virtuális tagként, amely alapértelmezett (de felüldefiniálható) megvalósítást definiál. Sőt a legtöbb tagot statikusként (osztályszinten kötött) vagy példányként (objektumszinten kötött) is konfigurálhatjuk. A típustagok létrehozását a következő néhány fejezet során vizsgáljuk meg.

---

**Megjegyzés** A C# nyelv támogatja a generikus típusok és tagok létrehozását is (lásd a 10. fejezetben).

---

## Valódi CTS-adattípusok

Az utolsó szempontként említhető az egységes típusrendszer vizsgálatában, hogy alapvető adattípusok jól meghatározott halmazát hozza létre. Habár minden nyelv rendelkezik egyedi kulcsszóval a valódi CTS-adattípusok deklarálására, az összes kulcsszó az `mscorlib.dll` nevű szerelvényben definiált típust adja meg. Nézzük meg az 1.2. táblázat alapján, hogy milyen kulcsszavakkal fejezzük ki a CTS-adattípusokat a különböző .NET-nyelvekben.

| CTS-adattípus               | VB .NET-kulcsszó      | C#-kulcsszó          | C++/CLI-kulcsszó                                          |
|-----------------------------|-----------------------|----------------------|-----------------------------------------------------------|
| <code>System.Byte</code>    | <code>Byte</code>     | <code>byte</code>    | <code>unsigned char</code>                                |
| <code>System.SByte</code>   | <code>SByte</code>    | <code>sbyte</code>   | <code>signed char</code>                                  |
| <code>System.Int16</code>   | <code>Short</code>    | <code>short</code>   | <code>short</code>                                        |
| <code>System.Int32</code>   | <code>Integer</code>  | <code>int</code>     | <code>int</code> vagy <code>long</code>                   |
| <code>System.Int64</code>   | <code>Long</code>     | <code>long</code>    | <code>__int64</code>                                      |
| <code>System.UInt16</code>  | <code>UShort</code>   | <code>ushort</code>  | <code>unsigned short</code>                               |
| <code>System.UInt32</code>  | <code>UInteger</code> | <code>uint</code>    | <code>unsigned int</code> vagy <code>unsigned long</code> |
| <code>System.UInt64</code>  | <code>ULong</code>    | <code>ulong</code>   | <code>unsigned __int64</code>                             |
| <code>System.Single</code>  | <code>Single</code>   | <code>float</code>   | <code>Float</code>                                        |
| <code>System.Double</code>  | <code>Double</code>   | <code>double</code>  | <code>Double</code>                                       |
| <code>System.Object</code>  | <code>Object</code>   | <code>object</code>  | <code>Object^</code>                                      |
| <code>System.Char</code>    | <code>Char</code>     | <code>char</code>    | <code>wchar_t</code>                                      |
| <code>System.String</code>  | <code>String</code>   | <code>String</code>  | <code>String^</code>                                      |
| <code>System.Decimal</code> | <code>Decimal</code>  | <code>decimal</code> | <code>Decimal</code>                                      |
| <code>System.Boolean</code> | <code>Boolean</code>  | <code>bool</code>    | <code>Bool</code>                                         |

1.2. táblázat: A valódi CTS-adattípusok

Mivel egy felügyelt nyelv egyedi kulcsszavai csak a `system` névtér valódi típusainak rövidített jelölései lehetnek, többé nem kell a numerikus adatok túlcsoordulásától, illetve alácsordulásától tartanunk a különböző nyelveken, ahogy a sztringek és a logikai értékek belső megjelenésétől sem.

Nézzük meg a következő kódrészleteket, amelyek nyelvi kulcsszavak és formális CTS-típusok segítségével definiálnak 32 bites numerikus változókat C# és VB .NET nyelveken.

```
// Egész számok definíciója C# nyelvben.
int i = 0;
System.Int32 j = 0;

' Egész számok definíciója "ints" VB .NET-ben.
Dim i As Integer = 0
Dim j As System.Int32 = 0
```

## A közös nyelvi specifikáció (CLS)

A különböző nyelvek tehát ugyanazokat a programozási szerkezeteket egyedi, nyelvspecifikus kifejezésekkel fejezik ki. A C# nyelvben például a sztringek összefűzését a plusz operátorral (+) jelöljük, míg VB .NET-ben az és jelet (&) használjuk. Ha két különböző nyelv ugyanazt a programozási kifejezést használja is (pl. egy olyan függvényt, amelynek nincs visszatérési értéke), nagy az esély, hogy a szintaxis mégis elég eltérő lesz:

```
// C# metódus, amely nem ad vissza semmit.
public void MyMethod()
{
    // Érdekes kód...
}

' VB metódus, amely nem ad vissza semmit.
Public Sub MyMethod()
    ' Érdekes kód...
End Sub
```

Ezek az apró szintaktikai eltérések azonban nem fontosak a .NET-futtatókörnyezet szempontjából, mivel a megfelelő fordítók (ebben az esetben a csc.exe vagy a vb.exe) hasonló CIL-utasításokat adnak ki. A nyelvek viszont a működésük általános szintje alapján is eltérőek lehetnek.

Egy .NET-nyelv például vagy rendelkezik kulcsszóval az előjel nélküli adatok megjelenítésére, vagy nem, illetve vagy támogatja a mutatótípusokat, vagy nem. Ilyen lehetőségek mellett jó lenne, ha volna egy olyan referencia-adat, amellyel az összes .NET-alapú nyelv összhangban lenne.

A közös nyelvi specifikáció olyan szabálykészlet, amely részletesen leírja, hogy egy adott .NET-alapú fordítónak milyen minimális és teljes szolgáltatásokat kell támogatnia a közös nyelvi futtatórendszerben (CLR) hosztolt forráskódok létrehozásához, illetve ahhoz, hogy minden .NET-platformot célzó nyelv egységes módon férhessen hozzá. A közös nyelvi specifikáció több szempontból az egységes típusrendszer által definiált, teljes funkcionalitás *részalmazának* tekinthető.

A közös nyelvi specifikáció végül is olyan szabálykészlet, amelyhez a fordító megalkotóinak igazodniuk kell, ha azt szeretnék, hogy a termékük zavartalanul működjön a .NET-univerzumban. Minden szabályhoz olyan egyszerű nevet rendeltek hozzá (pl. 6. CLS-szabály), amely leírja, hogy a szabály milyen hatással van azokra, akik a fordítót létrehozzák, és azokra, akik (valamilyen módon) kapcsolatba kerülnek vele. A közös nyelvi specifikáció alapja a 1. szabály:

- **1. szabály:** A CLS-szabályok a típusnak csak azokra a részeire vonatkoznak, amelyek a definiáló szerelvényen kívülről is hozzáférhetők.

Ebből a szabályból (helyesen) arra következtethetünk, hogy a többi CLS-szabály nem vonatkozik a .NET-típusok belső működésének létrehozásakor alkalmazott logikára. A típusoknak egyetlen szempontból kell összhangban lenniük a közös nyelvi specifikációval: a tagok definíciója (azaz elnevezési rend, paraméterek és visszatérési típusok) alapján. A tagok megvalósítási logikája bármennyi nem CLS-technikát használhat, hiszen a kívüllág nem veszi észre a különbséget.

Ennek illusztrálására nézzük az alábbiakat. A következő `Add()` metódus nem CLS-kompatibilis, ugyanis a paraméterek és a visszatérési értékek előjel nélküli adatokat használnak (és ez nem CLS-követelmény):

```
class Calc
{
    // Előjel nélküli adatok hozzáférése nem CLS-kompatibilis!
    public ulong Add(ulong x, ulong y)
    { return x + y; }
}
```

Ha viszont belsőleg, egyszerűen előjel nélküli adatokat használtunk volna:

```

class Calc
{
    public int Add(int x, int y)
    {
        // Mivel az ulong változót csak belsőleg használjuk,
        // a kód továbbra is CLS-kompatibilis.
        ulong temp = 0;
        ...
        return x + y;
    }
},

```

akkor még mindig megfelelnek a CLS-szabályoknak, és nyugodtak lehetnénk afelől, hogy bármelyik .NET-nyelv képes meghívni az Add() metódust.

A közös nyelvi specifikáció persze számos egyéb szabályt is definiál az 1. szabályon kívül. Leírja például, hogy egy adott nyelvnek hogyan kell a sztringeket és a felsorolt típusokat (belül tárolt típus) belsőleg reprezentálni, hogyan definiáljunk statikus tagokat, stb. Szerencsére nem kell ezeket a szabályokat megjegyeznünk ahhoz, hogy profi .NET-fejlesztők legyünk. Összegezve, a közös nyelvi specifikáció és az egységes típusrendszer mélyebb ismerete csak az eszköz-, illetve fordítóalkotók számára szükséges.

## A CLS-megfeleltetés biztosítása

Ahogy azt a későbbiekben látni fogjuk, a C# számos, nem CLS-kompatibilis programszerkezetet definiál. Jó hír azonban, hogy egy .NET-attribútum segítségével utasíthatjuk a C#-fordítót, hogy ellenőrizze a kódunk CLS-megfeleltetését:

```

// Utasítja a C#-fordítót a kód CLS-megfeleltetésének ellenőrzésére.
[assembly: System.CLSCompliant(true)]

```

A [CLSCompliant] attribútum utasítja a C#-fordítót, hogy ellenőrizze, a kód minden egyes sora megfelel-e a CLS-szabályoknak. Ha szabályellenes dolgot talál, a fordító hibát jelez, és megmutatja a hibás kódot. (Az attribútumalapú programozás részleteit a 16. fejezetben tárgyaljuk.)

## A közös nyelvi futtatórendszer (CLR)

Az egységes típusrendszeren (CTS) és a közös nyelvi specifikáción (CLS) kívül a harmadik hárombetűs rövidítés, amelyről még beszélnünk kell, a közös nyelvi futtatórendszer (CLR). Programozási szempontból a *futtatórendszer* kifejezés olyan külső szolgáltatások gyűjteményére utal, amelyek a kód egy adott lefordított egységének a végrehajtásához szükségesek. Például, amikor a fejlesztők a Microsoft Foundation Classes (MFC) segítségével hoznak létre új alkalmazást, tisztában vannak azzal, hogy a programjuk igényli az MFC-alapkönyvtárat (azaz az `mfc42.dll`-t). A többi népszerű nyelvnek is megvan a megfelelő futtatórendszere. A VB6-programozóknak szintén egy vagy két futtatókörnyezeti modulhoz kell igazodniuk (pl. `msvbvm60.dll`). A Java-fejlesztők a Java virtuális géphez (JVM) kötődnek; stb.

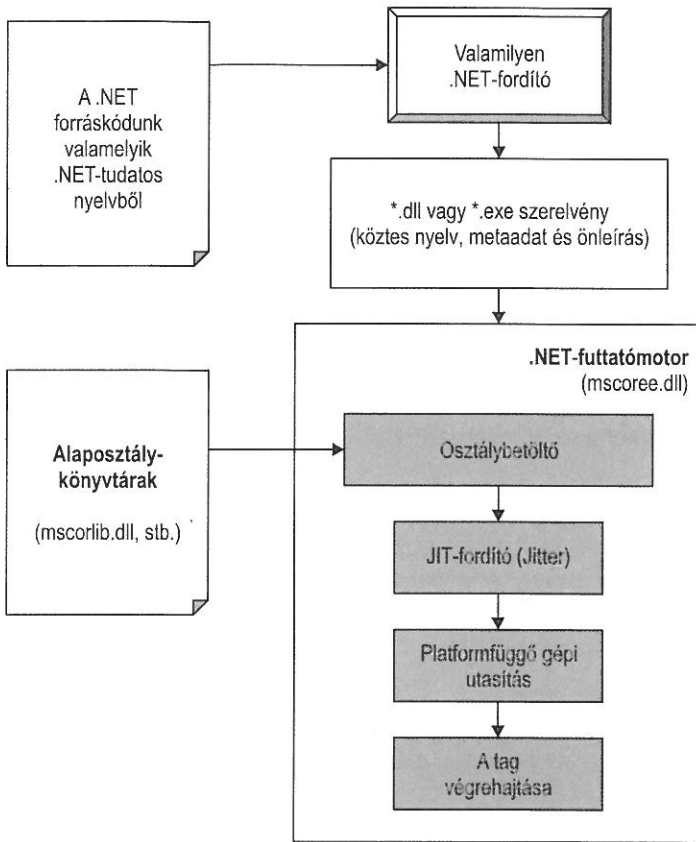
A .NET-platform egy további futtatási rendszert kínál. A legnagyobb különbség a .NET-futtatórendszer és a többi, imént említett futtatórendszer között az, hogy a .NET-futtatórendszer rendelkezik egy jól meghatározott futtatási réteggel, amelyet az *összes* .NET-alapú nyelv és -platform használ.

A közös nyelvi futtatórendszer lényege fizikailag az `mscorlib.dll` könyvtárban (más néven Common Object Runtime Execution Engine) jelenik meg. Amikor egy szerelvényt használni szeretnénk, és hivatkozunk rá, az `mscorlib.dll` automatikusan betöltődik, és ez tölti be a memóriába a szerelvényt. A futtatórendszer számos feladatért felelős. Elsősorban megadja egy szerelvény helyét, és a binárisban lévő metaadatot olvasva megkeresi az igényelt típust. A CLR ezt követően a memóriában helyezi el a típust, a hozzárendelt CIL-utasítást platformfüggő gépi utasításokká fordítja, végrehajtja a szükséges biztonsági ellenőrzéseket, majd végrehajtja a szóban forgó kódot.

Az egyedi szerelvények betöltése és az egyedi típusok létrehozása mellett, a CLR, ha szükséges, a .NET-alaposztálykönyvtárban tárolt típusokkal is kommunikál. Habár a teljes alaposztálykönyvtár több különálló szerelvényre oszlik, a legfontosabb szerelvény (`mscorlib.dll`) sok alaptípust tartalmaz, amelyek között sok közös programozási feladat található az összes .NET-nyelv által használt alapvető adattípussal egyetemben. Amikor .NET-megoldásokat hozunk létre, automatikusan hozzáférünk ehhez a szerelvényhez.

Az 1.4. ábra a forráskód (amely az alaposztálykönyvtár-típusokat használja), egy adott .NET-fordító és a .NET-futtatómotor közötti munkafolyamot mutatja be.





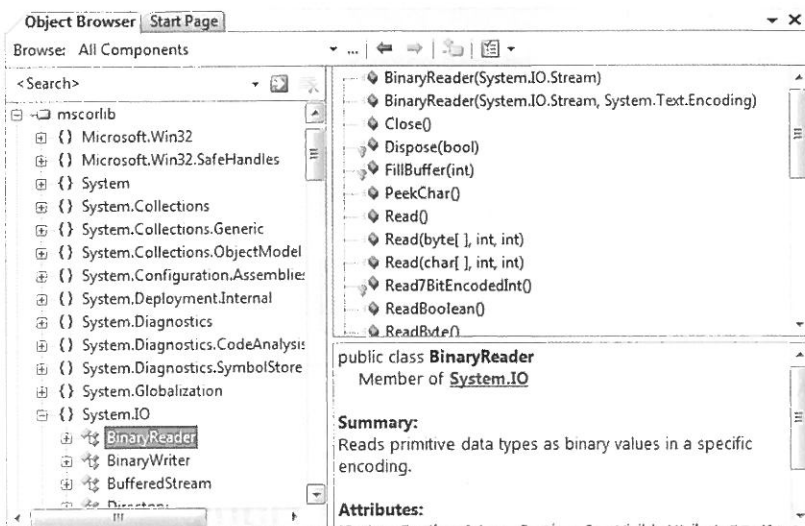
1.4. ábra: Az *mscorlib.dll* működés közben

## A szerelvény, a névtér és a típus megkülönböztetése

Mindannyian elfogadjuk a forráskódkönyvtárak fontosságát. Az MFC-hez, J2EE-hez és az ATL-hez hasonló könyvtáraknak az a célja, hogy a meglévő forráskódok olyan jól meghatározott készletét biztosítsa a fejlesztők számára, amelyet az alkalmazásaikban felhasználhatnak. A C# nyelv azonban nem rendelkezik nyelvfüggő kódkönyvtárral. A C#-fejlesztők ehelyett a nyelvsemleges .NET-könyvtárakat használják. Annak érdekében, hogy a típusok rendezetten helyezkedjenek el az alapostálykönyvtárakban, a .NET-platform messzemenően alkalmazza a *névtér* elvét.

Egyszerűen fogalmazva, a névtér az egy szerelvényben lévő, szemantikailag kapcsolódó típusok csoportja. A `System.IO` névtér például I/O-típusokat tartalmaz, a `System.Data` névtér pedig alapvető adatbázistípusokat definiál; stb. Hangsúlyoznunk kell, hogy egyetlen szerelvény (pl. az `microsoft.dll`) bármennyi névtérrel tartalmazhat, s ezek mindegyike bármennyi típussal rendelkezhet.

Ennek illusztrálására mutatja be az 1.5. ábra a Visual Studio 2008 Object Browserének egyik képernyőfotóját. Ezzel az eszközzel megvizsgálhatjuk azokat a szerelvényeket, amelyekre az aktuális projektünk hivatkozik, az egy adott szerelvényben lévő névtereket, az egy névtérben lévő típusokat, valamint egy bizonyos típus tagjait. Az `microsoft.dll` tehát több különböző névtérrel tartalmaz, mindegyiket a saját, szemantikailag kapcsolódó típusaival.



1.5. ábra: Egy szerelvény bármennyi névtérrel tartalmazhat

A legfontosabb különbség e között a felfogás és egy MFC-hez hasonló, nyelv-függő könyvtár között, hogy bármelyik nyelv, amelyik a .NET-futtatórendszert veszi célba, *ugyanazokat* a névtereket és *ugyanazokat* a típusokat használja. A következő három program ugyanazt a „Hello World” alkalmazást mutatja be C#, VB .NET és C++/CLI nyelveken:

```
// Hello world C# nyelven
using System;

public class MyApp
{
    static void Main()
    {
        Console.WriteLine("Hi from C#");
    }
}

' Hello world VB nyelven
Imports System

Public Module MyApp
    Sub Main()
        Console.WriteLine("Hi from VB")
    End Sub
End Module

// Hello world C++/CLI nyelven
#include "stdafx.h"
using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hi from C++/CLI");
    return 0;
}
```

Minden egyes nyelv használja a `System` névtérben definiált `Console` osztályt. Apróbb szintaktikai eltéréseken túl, ez a három alkalmazás nagyon sokban hasonlít egymásra, mind fizikailag, mind logikailag.

.NET-fejlesztőként az elsődleges célunk természetesen a számos .NET-névtérben definiált, temérdek típus megismerése. A legalapvetőbb névtér, amelyel találkozunk, a `System`. Ez a névtér biztosítja azoknak a típusoknak az alapvető készletét, amelyekre .NET-fejlesztőként időről időre szükségünk van. Valójában egyetlen működőképes C#-alkalmazást sem hozhatunk létre anélkül, hogy legalább ne hivatkoznánk a `System` névtérre, hiszen az alapvető adattípusokat (`System.Int32`, `System.String` stb.) ez a névtér definiálja. Az 1.3. táblázatban a kapcsolódó működések alapján csoportosított, néhány (de nem az összes) .NET-névtér látható.

| .NET-névtér                                                                                               | Valós jelentése                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System                                                                                                    | A System névtérben számos hasznos típust találunk, amelyek valós adatokkal, matematikai számításokkal, véletlenszám-generálással, környezeti változókkal és szemétygyűjtéssel foglalkoznak, továbbá közösen használt kivételeket és attribútumokat. |
| System.Collections<br>System.Collections.Generic                                                          | Ezek a névterek számos tárolótípust definiálnak, valamint alaptípusokat és interfészeket, amelyek révén testre szabott gyűjteményeket építhetünk fel.                                                                                               |
| System.Data<br>System.Data.Odbc<br>System.Data.OracleClient<br>System.Data.OleDb<br>System.Data.SqlClient | Ezeket a névtereket a relációs adatbázisokkal folytatott kommunikációra használjuk az ADO .NET alkalmazásakor.                                                                                                                                      |
| System.IO<br>System.IO.Compression<br>System.IO.Ports                                                     | Ezek a névterek olyan típusokat definiálnak, amelyeket fájl I/O-ra, adattömörítésre és portkezelésre használunk.                                                                                                                                    |
| System.Reflection<br>System.Reflection.Emit                                                               | Ezek a névterek olyan típusokat definiálnak, amelyek támogatják a futási időben történő típusmeghatározást, valamint a típusok dinamikus létrehozását.                                                                                              |
| System.Runtime.InteropServices                                                                            | Ez a névtér biztosítja a .NET-típusok számára azt a képességet, hogy a nem felügyelt kódokkal (pl. C-alapú Dll-ek és Com-kiszolgálók) kommunikáljanak és vizsont.                                                                                   |
| System.Drawing<br>System.Windows.Forms                                                                    | Ezek a névterek olyan típusokat definiálnak, amelyek révén a .NET eredeti felhasználói felületének eszközkészletét (Windows-űrlapok) használó asztali alkalmazásokat hozhatunk létre.                                                               |
| System.Windows<br>System.Windows.Controls<br>System.Windows.Shapes                                        | A System.Windows névtér, számos egyéb névtér gyökere, amely a Windows Presentation Foundation (WPF) felhasználói felületi eszközkészletet reprezentálja.                                                                                            |
| System.Linq<br>System.Xml.Linq<br>System.Data.Linq                                                        | Ezek a névterek a LINQ API-ban végzett programozás során alkalmazott típusokat definiálják.                                                                                                                                                         |

| .NET-névtér                                           | Valós jelentése                                                                                                                                                     |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System.Web                                            | Ez az egyik olyan névtér a sok közül, amelyik lehetővé teszi az ASP .NET webalkalmazások létrehozását.                                                              |
| System.ServiceModel                                   | Ez az egyik olyan névtér a sok közül, amelyekkel a WCF API segítségével elosztott alkalmazásokat hozhatunk létre.                                                   |
| System.Workflow.Runtime<br>System.Workflow.Activities | Ez a két névtér olyan típusokat definiál, amelyekkel a WF API segítségével munkafolyamat-engedélyezett alkalmazásokat hozhatunk létre.                              |
| System.Threading                                      | Ez a névtér számos olyan típust definiál, amelyeket többszálú alkalmazások létrehozására használunk.                                                                |
| System.Security                                       | A Security a .NET-univerzum integrált nézőpontja. A biztonságos névterekben számos olyan típust találunk, amelyek engedélyekkel, kriptográfiával stb. foglalkoznak. |
| System.Xml                                            | Az XML-központú névterek számos olyan típust tartalmaznak, amelyeket az XML-ada-tokkal való kommunikációra használunk.                                              |

1.3. táblázat: A .NET-névterek áttekintése

## A Microsoft-névterek szerepe

Az 1.3. táblázatot végigolvasva bizonyára feltűnt, hogy jó néhány beágyazott névtérnek (System.IO, System.Data stb.) a System a gyökérnévtére. Ahogy azonban kiderült, a .NET-alapostálykönyvtár definiálja a system névtéren kívüli legfontosabb gyökérnévterek nagy részét, amelyek közül a leghasznosabb a Microsoft.

Dióhéjban, a Microsoft-névtérbe ágyazott névterek (pl. Microsoft.CSharp, Microsoft.Ink, Microsoft.ManagementConsole és Microsoft.Win32) olyan típusokat tartalmaznak, amelyek egyedi, Windows operációs rendszerbeli szolgáltatásokkal való kommunikációra használatosak. Így ezeket a típusokat más .NET-engedélyezett operációs rendszerrel (pl. a Mac OS X-szel) nem használhatjuk. Könyvünk a Microsoft-gyökerű névterek részleteit azonban nem tárgyalja.

---

**Megjegyzés** A .NET Framework 3.5 SDK dokumentációjának használatát a 2. fejezet írja le, részletekkel szolgálva a névterekről, a típusokról és az alapsztálykönyvtárakban található tagokról.

---

## Programozott hozzáférés a névterekhez

A névtér tehát nem más, mint a kapcsolódó típusok logikai megértésének és elrendezésének a megfelelő módja. A `System` névtérre gondolva, azt feltételezhetjük, hogy a `System.Console` egy olyan `Console` nevű osztályt jelent, amely a `System` nevű névtérben található. A .NET-futtatórendszer szerint azonban ez nem így van. A futtatómodul csak egy `System.Console` nevű egységet lát.

A C#-ban a `using` kulcsszó leegyszerűsíti a névterekben definiált típusokra történő hivatkozás folyamatát. A működése a következő. Tegyük fel, hogy egy hagyományos asztali alkalmazást szeretnénk létrehozni. A főablak egy kiszolgálóoldali adatbázisból származó adatokra épülő oszlopdiagramot jelenít meg, valamint a cégünk logóját. Amíg kevésbé ismerjük még a névterekben lévő típusokat, íme néhány, amelyekre hivatkozhatunk a programunkban:

```
// Íme, az alkalmazás elkészítéséhez szükséges összes névtér.
using System;           // Általános alapsztálykönyvtár típusok.
using System.Drawing;    // Grafikus renderelési típusok.
using System.Windows.Forms; // Windows Forms GUI céleszköz típusok.
using System.Data;       // Általános adatközpontú típusok.
using System.Data.SqlClient; // MS SQL Server adathozzáférés
                           // típusok.
```

Ha egyszer már meghatároztunk néhány névtérrel (és hivatkoztunk az őket definiáló szerelvényekre), szabadon létrehozhatjuk a bennük található típusok példányait. Ha például a `Bitmap` osztály (amelyet a `System.Drawing` névtér definiál) egy példányát szeretnénk létrehozni, írjuk a következőket:

```
// A fájl által használt névterek listázása explicit módon.
using System;
using System.Drawing;

class Program
{
    public void DisplayLogo()
    {
        // 20 * 20 pixeles bittérkép létrehozása.
        Bitmap companyLogo = new Bitmap(20, 20);
        ...
    }
}
```

# A C# 2008 és a .NET 3.5

Első kötet (1-19. fejezet)

Andrew Troelsen



2009

**A C# 2008 és a .NET 3.5 – Első kötet (1-19. fejezet)**

A negyedik kiadás magyarul, két kötetben

Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition

Andrew Troelsen

Copyright 2008 by Apress.

Original English language edition published by Apress. Co Sound View Court 3B, Greenwich CT 06830 USA. Copyright © 2008 by Apress Co.. Hungarian-language edition copyright © 2009 by SZAK Kiadó. All rights reserved.

Magyar fordítás (Hungarian translation) © SZAK Kiadó 2009.

Fordította a SZAK Kiadó fordítócsoportha: Domoszlai László, Erős Melinda, Erős Szilvia, Iváncsy Renáta, Iváncsy Tamás, Lucza Mónika Tiber Melinda, Varga Péter

Lektor: Dávid Zoltán, MVP és Gincsei Gábor, MVP

Magyar nyelvi gondozás: Laczkó Krisztina

ISBN 978-963-9863-08-8

A könyv fordítása a Kilgray Kft. MemoQ (<http://www.memoqtm.com>) programjával készült, a szöveg helyességét és az elválasztásokat a MorphoLogic He-lyesek nevű programjával ellenőriztük.

**Minden jog fenntartva. Jelen könyvet, illetve annak részeit a kiadó engedélye nélkül tilos reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel elektronikus úton vagy más módon közölni.**



SZAK Kiadó Kft. ■ Az 1795-ben alapított Magyar Könyvkiadók és Könyvterjesztők Egyesülésének a tagja ■ 2060 Bicske, Diófa u. 3. ■ Tel.: 36-22-350-209 ■ Fax: 36-22-565-311 ■ [www.szak.hu](http://www.szak.hu) ■ e-mail: [info@szak.hu](mailto:info@szak.hu) ■ Kiadóvezető: Kis Ádám, e-mail: [adam.kis@szak.hu](mailto:adam.kis@szak.hu) ■ Főszerkesztő: Kis Balázs MCSE, MCT, e-mail: [balazs.kis@szak.hu](mailto:balazs.kis@szak.hu)



Ezt a kiadást Mikkónak, a csodamacskának, a 412-es szám alatt eltöltött éveknek és csodálatos feleségemnek, Amandának ajánlom, aki türelmesen várta, hogy elkészüljek újabb könyvem megírásával.



# Tartalomjegyzék

|                          |      |
|--------------------------|------|
| Köszönetnyilvánítás..... | xxix |
|--------------------------|------|

|                 |      |
|-----------------|------|
| Bevezetés ..... | xxxi |
|-----------------|------|

|                                                                                 |         |
|---------------------------------------------------------------------------------|---------|
| Mi egy csapat vagyunk, az Olvasó és én.....                                     | xxxii   |
| A könyv áttekintése.....                                                        | xxxiii  |
| Első kötet.....                                                                 | xxxiii  |
| 1. rész: Bevezetés a C#-ba és a .NET platformba .....                           | xxxiii  |
| 1. fejezet: A .NET filozófiája.....                                             | xxxiii  |
| 2. fejezet: C#-alkalmazások készítése .....                                     | xxxiv   |
| 2. rész: A C# alapvető építőelemei .....                                        | xxxiv   |
| 3. fejezet: A C# alapvető építőelemei, I. rész .....                            | xxxiv   |
| 4. fejezet: A C# alapvető építőelemei, II. rész.....                            | xxxiv   |
| 5. fejezet: Egységbe zárt osztálytípusok definiálása .....                      | xxxv    |
| 6. fejezet: A származtatás és a polimorfizmus.....                              | xxxv    |
| 7. fejezet: Strukturált hibakezelés.....                                        | xxxv    |
| 8. fejezet: Az objektumok életciklusa.....                                      | xxxv    |
| 3. rész: Haladó programozási szerkezetek a C#-ban .....                         | xxxvi   |
| 9. fejezet: Interfészek használata .....                                        | xxxvi   |
| 10. fejezet: Gyűjtemények és generikus típusok .....                            | xxxvi   |
| 11. fejezet: Metódusreferenciák, események és lambdák.....                      | xxxvi   |
| 12. fejezet: Indexerek, operátorok és mutatók.....                              | xxxvii  |
| 13. fejezet: C# 2008 nyelvi újdonságai .....                                    | xxxvii  |
| 14. fejezet: Bevezetés a nyelvbe ágyazott<br>lekérdezésekbe (LINQ) .....        | xxxvii  |
| 4. rész: Programozás .NET-szerelvényekkel .....                                 | xxxvii  |
| 15. fejezet: A .NET-szerelvények.....                                           | xxxviii |
| 16. fejezet: Típusreflexió, késői kötés és<br>attribútumalapú programozás ..... | xxxviii |
| 17. fejezet: Folyamatok, alkalmazástartományok és<br>objektumkörnyezetek .....  | xxxviii |
| 18. fejezet: Többszálú alkalmazások készítése .....                             | xxxviii |
| 19. fejezet: A köztes nyelv (CIL) és<br>a dinamikus szerelvények.....           | xxxix   |

|                                                                        |       |
|------------------------------------------------------------------------|-------|
| Második kötet .....                                                    | xxxix |
| 5. rész: Bevezetés a .NET alaposztálykönyvtáraiba .....                | xxxix |
| 20. fejezet: Fájlműveletek és elkülönített tárolás .....               | xxxix |
| 21. fejezet: Bevezetés az objektumsorosítás világába .....             | xl    |
| 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat .....                   | xl    |
| 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat .....                | xl    |
| 24. fejezet: A LINQ API programozása .....                             | xl    |
| 25. fejezet: A WCF .....                                               | xli   |
| 26. fejezet: A WF .....                                                | xli   |
| 6. rész: Felhasználói felületek .....                                  | xli   |
| 27. fejezet: Windows Forms-programozás .....                           | xli   |
| 28. fejezet: A WPF és az XAML .....                                    | xlII  |
| 29. fejezet: Programozás WPF-vezérlőelemekkel .....                    | xlII  |
| 30. fejezet: WPF 2D grafikus renderelés,<br>erőforrások és témák ..... | xlII  |
| 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével .....     | xlIII |
| 31. fejezet: ASP.NET weboldalak készítése .....                        | xlIII |
| 32. fejezet: ASP.NET-vezérlőelemek,<br>témák és mesteroldalak .....    | xlIII |
| 33. fejezet: ASP.NET állapotkezelési technikák .....                   | xlIII |
| 8. rész: Függelék .....                                                | xlIV  |
| A függelék: A COM és a .NET együttműködése .....                       | xlIV  |
| B függelék: Platformfüggetlen .NET-fejlesztés a Monóval .....          | xlIV  |
| Öt szabadon letölthető fejezet – még több információ .....             | xlV   |
| A könyv forráskódjának igénylése .....                                 | xlV   |
| A lehetséges javítások .....                                           | xlVI  |
| Elérhetőségem .....                                                    | xlVI  |

## 1. rész: Bevezetés a C#-ba és a .NET platformba

### 1. A .NET filozófiája..... 3

|                                             |   |
|---------------------------------------------|---|
| Az előzmények áttekintése .....             | 4 |
| Egy C/Win32 API programozó élete .....      | 4 |
| Egy C++/MFC-programozó élete .....          | 4 |
| Egy Visual Basic 6.0 programozó élete ..... | 5 |
| Egy Java/J2EE-programozó élete .....        | 5 |
| Egy COM-fejlesztő élete .....               | 6 |
| Egy Windows DNA programozó élete .....      | 7 |

|                                                                     |    |
|---------------------------------------------------------------------|----|
| A .NET-megoldás .....                                               | 8  |
| Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS) .....     | 9  |
| Az alaposztálykönyvtárak szerepe .....                              | 10 |
| A C# előnyei .....                                                  | 11 |
| További .NET-alapú programozási nyelvek .....                       | 13 |
| Élet egy többnyelvű világban .....                                  | 14 |
| A .NET-szerelvények áttekintése .....                               | 15 |
| Egyfájlos és többfájlos szerelvények .....                          | 17 |
| A közös köztes nyelv (CIL) szerepe .....                            | 18 |
| A köztes nyelv előnyei .....                                        | 20 |
| A köztes nyelv fordítása platformfüggő utasítássá .....             | 21 |
| A .NET-típusmetaadat szerepe .....                                  | 22 |
| A szerelvény manifesztumának szerepe .....                          | 23 |
| A közös típusrendszer (CTS) .....                                   | 23 |
| Az egységes típusrendszer osztálytípusai .....                      | 24 |
| Az egységes típusrendszer interfésztípusai .....                    | 25 |
| Az egységes típusrendszer struktúrátípusai .....                    | 25 |
| Az egységes típusrendszer felsorolt típusai .....                   | 26 |
| Az egységes típusrendszer metódusreferencia-típusai .....           | 27 |
| Az egységes típusrendszer típustagjai .....                         | 27 |
| Valódi CTS-adattípusok .....                                        | 28 |
| A közös nyelvi specifikáció (CLS) .....                             | 29 |
| A CLS-megfeleltetés biztosítása .....                               | 31 |
| A közös nyelvi futtatórendszer (CLR) .....                          | 32 |
| A szerelvény, a névtér és a típus megkülönböztetése .....           | 33 |
| A Microsoft-névterek szerepe .....                                  | 37 |
| Programozott hozzáférés a névterekhez .....                         | 38 |
| Hivatkozás külső szerelvényekre .....                               | 39 |
| Szerelvények feltárása az ildasm.exe segítségével .....             | 40 |
| A köztes nyelvi kód megtekintése .....                              | 41 |
| A típusmetaadat megtekintése .....                                  | 42 |
| Szerelvény metaadatának (manifesztumának) megtekintése .....        | 42 |
| Szerelvények feltárása a Lutz Roeder's Reflector segítségével ..... | 43 |
| A .NET-futtatókörnyezet telepítése .....                            | 44 |
| A .NET platformfüggetlen természete .....                           | 44 |
| Összefoglalás .....                                                 | 46 |

## 2. C#-alkalmazások fordítása ..... 49

|                                                                                      |    |
|--------------------------------------------------------------------------------------|----|
| A .NET 3.5 keretrendszer fejlesztőkészletének szerepe .....                          | 49 |
| A Visual Studio 2008 parancssor .....                                                | 50 |
| C#-alkalmazások fordítása csc.exe használatával.....                                 | 51 |
| Bemeneti és kimeneti célok megadása .....                                            | 52 |
| Hivatkozás külső szerelvényekre .....                                                | 54 |
| Hivatkozás több külső szerelvényre .....                                             | 55 |
| Több forrásfájl fordítása .....                                                      | 55 |
| A C#-reakciófájlok használata .....                                                  | 56 |
| .NET-alkalmazás készítése TextPad használatával .....                                | 59 |
| A C#-kulcsszószínezés bekapcsolása .....                                             | 59 |
| A *.cs fájl szűrő konfigurálása .....                                                | 60 |
| A csc.exe integrálása .....                                                          | 61 |
| Futtatási parancsok társítása mentőelemekhez .....                                   | 62 |
| .NET-alkalmazás készítése Notepad++ használatával.....                               | 64 |
| Az automatikus kiegészítési lista testreszabása .....                                | 65 |
| .NET-alkalmazás készítése SharpDevelop használatával.....                            | 66 |
| Egy egyszerű tesztprojekt létrehozása .....                                          | 67 |
| .NET-alkalmazás készítése Visual C# 2008 Express használatával.....                  | 69 |
| A Visual C# Express néhány jellegzetessége .....                                     | 70 |
| .NET-alkalmazás készítése Visual Studio 2008 használatával .....                     | 71 |
| A Visual Studio 2008 néhány jellegzetessége.....                                     | 72 |
| A .NET-keretrendszer kiválasztása a New Project<br>párbeszédablak segítségével ..... | 72 |
| A Solution Explorer használata .....                                                 | 73 |
| Hivatkozás külső szerelvényekre .....                                                | 74 |
| Projekt tulajdonságok megtekintése .....                                             | 74 |
| A Class View.....                                                                    | 75 |
| Az Object Browser .....                                                              | 76 |
| Integrált támogatás a kódújratervezéshez.....                                        | 76 |
| Kódbővítések és a környezetbe foglalás technológiája .....                           | 79 |
| A Visual Class Designer.....                                                         | 81 |
| Object Test Bench.....                                                               | 84 |
| Az integrált .NET Framework 3.5 dokumentációs rendszer .....                         | 85 |
| További .NET-fejlesztőeszközök .....                                                 | 87 |
| Összefoglalás.....                                                                   | 88 |

## 2. rész: A C# alapvető építőelemei

### 3. A C# alapvető építőelemei, I. rész ..... 91

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| Egy egyszerű C#-program anatómiája .....                                   | 91  |
| Variációk Main() metódusra.....                                            | 93  |
| Alkalmazás-hibakód megadása .....                                          | 94  |
| A parancssori argumentumok feldolgozása .....                              | 96  |
| Parancssori paraméterek megadása Visual Studio 2008-ban.....               | 98  |
| Érdekes kitérő: a System.Environment osztály néhány<br>további tagja ..... | 99  |
| A System.Console osztály .....                                             | 100 |
| A Console osztály alapvető bemenete és kimenete.....                       | 101 |
| A konzolkimenet formázása.....                                             | 102 |
| Numerikus adatok formázása .....                                           | 103 |
| Numerikus adatok formázása a parancssori<br>alkalmazásokon túl.....        | 105 |
| A System-adattípusok és gyorsazonosítók a C#-ban .....                     | 106 |
| Változódeklaráció és inicializálás .....                                   | 108 |
| A beépített adattípusok példányosítása.....                                | 110 |
| Az adattípusok hierarchiája.....                                           | 110 |
| A numerikus adattípusok tagjai .....                                       | 112 |
| A System.Boolean tagjai.....                                               | 113 |
| A System.Char tagjai .....                                                 | 114 |
| Értékek értelmezése sztringadatokból .....                                 | 114 |
| A System.String típus.....                                                 | 115 |
| Egyszerű sztringkezelés .....                                              | 116 |
| Sztringösszefűzés.....                                                     | 117 |
| Vezérlőkértékek.....                                                       | 118 |
| Szó szerinti sztringek definiálása .....                                   | 119 |
| A sztringek és az egyenlőség.....                                          | 120 |
| A sztring megváltoztathatatlan.....                                        | 121 |
| A System.Text.StringBuilder típus .....                                    | 123 |
| System.DateTime és System.TimeSpan .....                                   | 125 |
| Szűkítő és bővítő adattípus-konverziók.....                                | 126 |
| Hibacsapda állítása a szűkítő adatkonverzióknak .....                      | 128 |
| Projektszintű túlcsoportolás-ellenőrzés beállítása.....                    | 131 |
| Az unchecked kulcsszó .....                                                | 132 |
| A System.Convert szerepe.....                                              | 132 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Ciklus készítése a C#-ban.....                               | 133 |
| A for ciklus.....                                            | 133 |
| A foreach ciklus.....                                        | 134 |
| A while és a do/while ciklusszerkezetek.....                 | 135 |
| Elágazó szerkezetek és a reláció/egyenlőség-operátorok ..... | 136 |
| Az if/else utasítás .....                                    | 136 |
| A switch utasítás .....                                      | 138 |
| Összefoglalás.....                                           | 139 |

#### 4. A C# alapvető építőelemei, II. rész..... 141

|                                                                  |     |
|------------------------------------------------------------------|-----|
| Metódusok és paramétermódosítók .....                            | 141 |
| Az alapértelmezett paraméterátadási viselkedés .....             | 142 |
| Az out módosító.....                                             | 144 |
| A ref módosító.....                                              | 145 |
| A params módosító .....                                          | 147 |
| A tag túlterhelésének megértése .....                            | 149 |
| Tömbök kezelése a C#-ban .....                                   | 151 |
| A C# tömbinicializáló szintaxisa .....                           | 152 |
| Objektumtömb definiálása .....                                   | 153 |
| Munka a többdimenziós tömbökkel.....                             | 154 |
| Tömbök mint paraméterek (és visszatérési értékek).....           | 156 |
| A System.Array ösosztály.....                                    | 156 |
| Az Enum típus .....                                              | 159 |
| Az enum alapjául szolgáló tároló vezérlése.....                  | 160 |
| Felsorolások deklarálása és használata.....                      | 160 |
| A System.Enum típus.....                                         | 162 |
| Felsorolás név/érték párpainak dinamikus feltárása.....          | 162 |
| A struktúrátípus .....                                           | 165 |
| Struktúraváltozók létrehozása .....                              | 167 |
| Érték- és referenciátípusok.....                                 | 169 |
| Érték- és referenciátípusok, valamint az értékadó operátor ..... | 170 |
| Referenciátípusokat tartalmazó értéktípusok .....                | 172 |
| Referenciátípusok átadása értékként.....                         | 174 |
| Referenciátípusok átadása referenciaként .....                   | 176 |
| Érték- és referenciátípusok: további részletek.....              | 177 |
| Nullázható típusok a C#-ban.....                                 | 179 |
| Munka a nullázható típusokkal .....                              | 180 |
| A ?? operátor.....                                               | 181 |
| Összefoglalás.....                                               | 182 |



## 5. Beágyazott osztálytípusok definiálása ..... 183

|                                                       |     |
|-------------------------------------------------------|-----|
| Bevezetés a C# osztálytípusaiba.....                  | 183 |
| Objektumok memóriaterületének lefoglalása             |     |
| a new kulcsszóval .....                               | 186 |
| A konstruktor .....                                   | 187 |
| Az alapértelmezett konstruktor szerepe.....           | 188 |
| Egyedi konstruktor definiálása .....                  | 189 |
| Az újradefiniált alapértelmezett konstruktor .....    | 190 |
| A this kulcsszó szerepe .....                         | 191 |
| Konstruktorhívások láncolása a this kulcsszóval.....  | 193 |
| A konstruktorok lefutásának sorrendje .....           | 196 |
| A static kulcsszó .....                               | 198 |
| Statikus metódusok (és mezők) definiálása .....       | 199 |
| Statikus adatok definiálása .....                     | 200 |
| Statikus konstruktorok definiálása.....               | 203 |
| Statikus osztályok definiálása .....                  | 205 |
| Az objektumorientált programozás alappillérei .....   | 207 |
| Az egységbe zárás.....                                | 208 |
| A származtatás .....                                  | 209 |
| A polimorfizmus .....                                 | 210 |
| Hozzáférés-módosítók a C#-ban.....                    | 212 |
| Az alapértelmezett hozzáférés-módosítók .....         | 213 |
| Hozzáférés-módosítók és beágyazott típusok .....      | 214 |
| Az első pillér: az egységbe zárás a C#-ban .....      | 215 |
| Egységbe zárás tradicionális lekérdezők és            |     |
| módosítók használatával .....                         | 216 |
| Egységbe zárás típustulajdonságok használatával ..... | 218 |
| A tulajdonságok belső reprezentációja .....           | 220 |
| A get/set utasítástulajdonságok láthatósági           |     |
| szintjeinek vezérlése .....                           | 222 |
| Csak olvasható és csak írható tulajdonságok .....     | 223 |
| Statikus tulajdonságok .....                          | 224 |
| Konstans adatok .....                                 | 226 |
| Írásvédett mezők .....                                | 227 |
| Statikus írásvédett mezők.....                        | 228 |
| Részleges típusok .....                               | 229 |

|                                                        |     |
|--------------------------------------------------------|-----|
| A C#-forráskód dokumentálása XML segítségével .....    | 231 |
| Az XML-fájl generálása .....                           | 234 |
| XML-megjegyzések átalakítása az NDoc-on keresztül..... | 235 |
| Az elvégzett munka eredményének láthatóvá tétele.....  | 237 |
| Összefoglalás.....                                     | 238 |

## **6. A származtatás és a polimorfizmus..... 239**

|                                                                               |     |
|-------------------------------------------------------------------------------|-----|
| A származtatás alapvető szerkezete .....                                      | 239 |
| Az osztálytípus szülőosztálya.....                                            | 240 |
| Több őszosztály .....                                                         | 242 |
| A sealed kulcsszó.....                                                        | 243 |
| Osztálydiagramok készítése a Visual Studio 2008-ban.....                      | 245 |
| A második pillér: a származtatás .....                                        | 246 |
| Őszosztály konstruktorának meghívása a base kulcsszóval .....                 | 249 |
| Családi titok megtartása: a protected kulcsszó.....                           | 251 |
| Lezárt osztály hozzáadása .....                                               | 252 |
| A tartalmazás és a delegálás .....                                            | 254 |
| A beágyazott típus definiálása .....                                          | 255 |
| A harmadik pillér: a polimorfizmus támogatása .....                           | 258 |
| A virtual és override kulcsszavak.....                                        | 259 |
| Virtuális tagok felülbírállása a Visual Studio 2008 segítségével.....         | 261 |
| A virtuális tagok lezárása .....                                              | 262 |
| Az absztrakt osztályok .....                                                  | 263 |
| Polimorfikus interfész építése.....                                           | 265 |
| Tagok eltakarása .....                                                        | 270 |
| Az őszosztály, a leszármazott osztály és<br>a típusmódosítási szabályok ..... | 272 |
| Az as kulcsszó.....                                                           | 274 |
| Az is kulcsszó .....                                                          | 274 |
| A fő szülőosztály: a System.Object.....                                       | 275 |
| A System.Object.ToString() felüldefiniálása .....                             | 279 |
| A System.Object.Equals() felüldefiniálása.....                                | 280 |
| A System.Object.GetHashCode() felüldefiniálása.....                           | 281 |
| A módosított Person osztály tesztelése.....                                   | 282 |
| A System.Object statikus tagjai.....                                          | 283 |
| Összefoglalás.....                                                            | 284 |

## 7. A strukturált kivételkezelés ..... 285

|                                                                       |     |
|-----------------------------------------------------------------------|-----|
| Hibák, programhibák és kivételek .....                                | 285 |
| A kivételkezelés szerepe a .NET-ben .....                             | 286 |
| A .NET-kivételkezelés .....                                           | 288 |
| A System.Exception osztály .....                                      | 288 |
| A lehető legegyszerűbb példa .....                                    | 290 |
| Általános kivétel dobása .....                                        | 292 |
| Kivételek elkapása .....                                              | 294 |
| A kivétel tulajdonságainak beállítása .....                           | 295 |
| A TargetSite tulajdonság .....                                        | 295 |
| A StackTrace tulajdonság .....                                        | 297 |
| A HelpLink tulajdonság .....                                          | 297 |
| A Data tulajdonság .....                                              | 298 |
| Rendszerszintű kivételek (System.SystemException) .....               | 301 |
| Alkalmazásszintű kivételek (System.ApplicationException) .....        | 302 |
| Egyedi kivétel készítése, 1. rész .....                               | 302 |
| Egyedi kivétel készítése, 2. rész .....                               | 305 |
| Egyedi kivétel készítése, 3. rész .....                               | 306 |
| Többszörös kivételek feldolgozása .....                               | 308 |
| Általános catch utasítások .....                                      | 310 |
| Kivételek továbbdobása .....                                          | 311 |
| Belső kivételek .....                                                 | 311 |
| A finally blokk .....                                                 | 313 |
| Ki mit dob? .....                                                     | 314 |
| A kezeletlen kivételek eredménye .....                                | 315 |
| Kezeletlen kivételek hibakeresése a Visual Studio használatával ..... | 315 |
| Összefoglalás .....                                                   | 317 |

## 8. Az objektumok élettartama..... 319

|                                                 |     |
|-------------------------------------------------|-----|
| Osztályok, objektumok és referenciák .....      | 319 |
| Az objektumok élettartama .....                 | 321 |
| A new kulcsszó köztes nyelve .....              | 322 |
| Objektumreferenciák null értékre állítása ..... | 323 |
| Az alkalmazás gyökérelemének szerepe .....      | 325 |
| Az objektumgenerációk .....                     | 327 |
| A System.GC típus .....                         | 328 |
| A szemétygyűjtés kikényszerítése .....          | 330 |

|                                                     |     |
|-----------------------------------------------------|-----|
| Véglegesíthető objektumok készítése .....           | 333 |
| A System.Object.Finalize() felüldefiniálása .....   | 334 |
| A véglegesítő folyamat részletei .....              | 337 |
| Eldobható objektumok készítése .....                | 337 |
| A C# using kulcsszó egy újabb alkalmazása .....     | 340 |
| Véglegesíthető és eldobható típusok készítése ..... | 342 |
| A formalizált felszabadítási minta .....            | 343 |
| Összefoglalás .....                                 | 346 |

### 3. rész: Haladó programozási szerkezetek a C#-ban

#### 9. Interfészek használata ..... 349

|                                                                        |     |
|------------------------------------------------------------------------|-----|
| Az Interface típus .....                                               | 349 |
| Az interfésztípusok és absztrakt ósosztályok<br>összehasonlítása ..... | 351 |
| Saját interfészek definiálása .....                                    | 354 |
| Az interfész implementálása .....                                      | 356 |
| Interfésztagok hívása az objektumon .....                              | 358 |
| Az interfészreferenciák megszerzése: az as kulcsszó .....              | 360 |
| Az interfészreferenciák megszerzése: az is kulcsszó .....              | 360 |
| Interfészek mint paraméterek .....                                     | 361 |
| Interfészek mint visszatérési értékek .....                            | 364 |
| Interfésztípusok tömbje .....                                          | 364 |
| Interfészimplementálás a Visual Studio 2008 segítségével .....         | 366 |
| Névütközés feloldása explicit interfészimplementációval .....          | 367 |
| Interfészhierarchia tervezése .....                                    | 371 |
| Többszörös öröklés interfésztípusokkal .....                           | 372 |
| Felsorolható típusok készítése (IEnumerable és IEnumerator) .....      | 375 |
| Iterátor metódus készítése a yield kulcsszóval .....                   | 379 |
| Nevesített iterátor létrehozása .....                                  | 380 |
| Az iterátormetódusok belső reprezentációja .....                       | 381 |
| Klónozható objektumok készítése (ICloneable) .....                     | 383 |
| Egy összetettebb klónozási példa .....                                 | 385 |
| Összehasonlítható objektumok készítése (Comparable) .....              | 389 |
| Többszörös rendezési sorrend megadása (Comparable) .....               | 392 |
| Egyedi tulajdonságok, egyedi rendezési típusok .....                   | 394 |
| A Callback interfész .....                                             | 395 |
| Összefoglalás .....                                                    | 399 |

## 10. Gyűjtemények és generikus típusok ..... 401

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| A System.Collections névtér interfészei .....                   | 401 |
| Az ICollection interfész .....                                  | 403 |
| Az IDictionary interfész .....                                  | 403 |
| Az IDictionaryEnumerator interfész .....                        | 404 |
| Az IList interfész .....                                        | 404 |
| A System.Collections névtér osztálytípusai .....                | 405 |
| Az ArrayList típus használata .....                             | 406 |
| A Queue típus használata .....                                  | 408 |
| A Stack típus használata .....                                  | 409 |
| A System.Collections.Specialized névtér .....                   | 410 |
| A bedobozolás, kidobozolás és a System.Object kapcsolata .....  | 412 |
| A dobozolással járó probléma .....                              | 415 |
| A típusbiztonság és az erősen típusos gyűjtemények .....        | 416 |
| Egyedi gyűjtemény készítése .....                               | 417 |
| Az erősen típusos gyűjtemények és a dobozolás .....             | 419 |
| A System.Collections.Generic névtér .....                       | 421 |
| A List<T> típus vizsgálata .....                                | 423 |
| Saját generikus metódusok létrehozása .....                     | 426 |
| A típusparaméterek kikövetkeztetése .....                       | 427 |
| Generikus struktúrák és osztályok létrehozása .....             | 429 |
| A default kulcsszó a generikus kódban .....                     | 430 |
| Saját generikus gyűjtemény készítése .....                      | 432 |
| A saját generikus gyűjtemény korlátai .....                     | 433 |
| Típusparaméter korlátozása a where kulcsszó használatával ..... | 435 |
| Az operátorkorlátozások hiánya .....                            | 438 |
| Generikus ősosztály létrehozása .....                           | 439 |
| Generikus interfész létrehozása .....                           | 440 |
| Összefoglalás .....                                             | 442 |

## 11. Metódusreferenciák, események és lambdák ..... 443

|                                                                   |     |
|-------------------------------------------------------------------|-----|
| A.NET-metódusreferencia .....                                     | 443 |
| A metódusreferencia definiálása C#-ban .....                      | 445 |
| A System.MulticastDelegate és System.Delegate alaposztályok ..... | 448 |
| A legegyszerűbb metódusreferencia .....                           | 449 |
| A metódusreferencia-objektum vizsgálata .....                     | 452 |

|                                                                                       |     |
|---------------------------------------------------------------------------------------|-----|
| A Car típus utólagos módosítása metódusreferenciákkal .....                           | 453 |
| Többes küldés engedélyezése .....                                                     | 457 |
| Cél eltávolítása a metódusreferencia hívási listájáról .....                          | 459 |
| Egy összetettebb metódusreferencia .....                                              | 460 |
| Metódusreferenciák mint paraméterek .....                                             | 461 |
| Az átirányító kód elemzése .....                                                      | 464 |
| Metódusreferenciák kovarianciája .....                                                | 465 |
| Generikus metódusreferencia készítése .....                                           | 468 |
| A generikus metódusreferencia szimulációja<br>generikus típusok nélkül .....          | 469 |
| C#-események .....                                                                    | 471 |
| Az event kulcsszó .....                                                               | 472 |
| Eseménykezelés a felszín alatt .....                                                  | 474 |
| A bejövő események figyelése .....                                                    | 476 |
| Esemény beregisztrálásának egyszerűsítése<br>a Visual Studio 2008 használatával ..... | 477 |
| Egy „korrekt” esemény .....                                                           | 478 |
| A generikus EventHandler<T> metódusreferencia .....                                   | 480 |
| Névtelen metódusok a C#-ban .....                                                     | 481 |
| „Külső” változók elérése .....                                                        | 483 |
| A metóduscsoport átalakítása .....                                                    | 484 |
| A C# 2008 lambda operátora .....                                                      | 486 |
| A lambda kifejezések részletezése .....                                               | 489 |
| Argumentumok feldolgozása több utasításon keresztül .....                             | 490 |
| A CarDelegate példa áttervezése lambda<br>kifejezések használatával .....             | 492 |
| Lambda kifejezések több paraméterrel<br>(vagy paraméterek nélkül) .....               | 494 |
| Összefoglalás .....                                                                   | 496 |

## **12. Indexelők, operátorok és mutatók ..... 497**

|                                               |     |
|-----------------------------------------------|-----|
| Az indexelő metódus .....                     | 497 |
| Objektumok indexelése sztringértékekkel ..... | 500 |
| Az indexelő metódusok túlterhelése .....      | 501 |
| Az indexelő metódusok belső ábrázolása .....  | 502 |
| Többdimenziós indexelők .....                 | 503 |
| Indexelő definíciók interfésztípusokon .....  | 503 |

|                                                            |     |
|------------------------------------------------------------|-----|
| Az operátor-túlterhelés.....                               | 504 |
| Kétooperandusú operátorok túlterhelése.....                | 506 |
| Mi a helyzet a += és -= operátorokkal?.....                | 508 |
| Egyoperandusú operátorok túlterhelése.....                 | 509 |
| Az egyenlőségvizsgáló operátorok túlterhelése.....         | 510 |
| Összehasonlító operátorok túlterhelése.....                | 511 |
| A túlterhelt operátorok belső ábrázolása.....              | 512 |
| Zárszó az operátor-túlterhelésről.....                     | 514 |
| Egyedi típuskonverziók.....                                | 515 |
| A numerikus konverzió.....                                 | 515 |
| Konverzió az egymásból származó osztálytípusok között..... | 516 |
| Egyedi konverziós rutinok készítése.....                   | 517 |
| A Square típus további explicit konverziói.....            | 520 |
| Implicit konverziós rutinok definiálása.....               | 521 |
| Az egyedi konverziós rutinok belső ábrázolása.....         | 523 |
| Mutatótípusok használata.....                              | 524 |
| Az unsafe kulcsszó.....                                    | 526 |
| A * és az & operátor.....                                  | 528 |
| A nem felügyelt (és felügyelt) cserefüggvény.....          | 528 |
| Mezőelérés mutatókkal (a -> operátor).....                 | 530 |
| A stackalloc kulcsszó.....                                 | 530 |
| Típuspéldány rögzítése a fixed kulcsszóval.....            | 531 |
| A sizeof kulcsszó.....                                     | 532 |
| A C# előfordítói direktívái.....                           | 533 |
| Kódrégiók megadása.....                                    | 534 |
| Feltételes kódfordítás.....                                | 535 |
| Összefoglalás.....                                         | 537 |

### 13. A C# 2008 nyelv szolgáltatásai ..... 539

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| Implicit típusú lokális változók.....                           | 539 |
| A var használata foreach szerkezetekben.....                    | 541 |
| Az implicit típusú változókra vonatkozó korlátozások.....       | 542 |
| Implicit típusú lokális tömbök.....                             | 543 |
| Az implicit típusú adatok erősen tipizáltak.....                | 544 |
| Az implicit típusú lokális változók előnye.....                 | 545 |
| Automatikus tulajdonságok.....                                  | 546 |
| Interakció automatikus tulajdonságokkal.....                    | 548 |
| Az automatikus tulajdonságok elérésének korlátozása.....        | 549 |
| Az automatikus tulajdonságok és az alapértelmezett értékek..... | 549 |

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| Bővítő metódusok.....                                             | 551        |
| A bővítő metódusok definiálása .....                              | 552        |
| A bővítő metódusok meghívása példányszinten .....                 | 553        |
| A bővítő metódusok statikus meghívása .....                       | 555        |
| A bővítő metódus hatóköre.....                                    | 555        |
| A bővítő metódusokat definiáló típusok importálása .....          | 557        |
| IntelliSense a bővítő metódusokhoz.....                           | 558        |
| Bővítő könyvtárak létrehozása és használata .....                 | 559        |
| Interfésztípusok kiterjesztése bővítő metódusokkal.....           | 560        |
| Részleges metódusok .....                                         | 562        |
| A részleges metódusok első pillantásra .....                      | 563        |
| A részleges metódusok használata.....                             | 565        |
| Az objektuminicializáló szintaxis.....                            | 566        |
| Egyedi konstruktor meghívása az inicializációs szintaxissal ..... | 568        |
| A belső típusok inicializálása .....                              | 569        |
| Gyűjteményinicializálók .....                                     | 571        |
| Névtelen típusok .....                                            | 572        |
| A névtelen típusok belső ábrázolása .....                         | 573        |
| A ToString() és a GetHashCode() megvalósítása .....               | 575        |
| Az egyenlőség szemantikája névtelen típusoknál.....               | 576        |
| Névtelen típusokat tartalmazó névtelen típusok .....              | 578        |
| Összefoglalás.....                                                | 579        |
| <br><b>14. A LINQ. Bevezetés .....</b>                            | <b>581</b> |
| A LINQ szerepe .....                                              | 581        |
| A LINQ-kifejezések jellemzői                                      |            |
| (típusizáltság és kiterjeszthetőség) .....                        | 583        |
| Alapvető LINQ-szerelvények .....                                  | 583        |
| Bevezetés a LINQ lekérdező kifejezésekbe .....                    | 585        |
| A LINQ és az implicit típusú lokális változók.....                | 588        |
| A LINQ és a bővítő metódusok .....                                | 590        |
| A késleltetett futtatás szerepe.....                              | 591        |
| Az azonnali futtatás szerepe .....                                | 592        |
| A LINQ és a generikus gyűjtemények.....                           | 593        |
| A LINQ-kifejezés alkalmazása .....                                | 594        |
| A LINQ és a nem generikus gyűjtemények.....                       | 595        |
| Adatok szűrése az OfType<T>() használatával.....                  | 596        |



|                                                                                                           |     |
|-----------------------------------------------------------------------------------------------------------|-----|
| A LINQ-lekérdezésoperátorok belső ábrázolása .....                                                        | 597 |
| Lekérdező kifejezések létrehozása lekérdezésoperátorok<br>segítségével. Ismétlés .....                    | 598 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a lambdák használatával.....                  | 599 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a névtelen metódusok használatával .....      | 601 |
| Lekérdező kifejezések létrehozása az Enumerable típus és<br>a nyers metódusreferenciák használatával..... | 602 |
| A LINQ-lekérdezésoperátorok .....                                                                         | 603 |
| Elemek számának meghatározása<br>az Enumerable segítségével.....                                          | 604 |
| Egyszerű tesztprojekt létrehozása .....                                                                   | 605 |
| A kiválasztó szintaxis alapjai.....                                                                       | 606 |
| Adatok egy részhalmazának kinyerése.....                                                                  | 608 |
| Új adattípusok leképezése.....                                                                            | 609 |
| Az eredményhalmaz-elemek sorrendjének megfordítása.....                                                   | 610 |
| Kifejezések rendezése .....                                                                               | 610 |
| Különbségek keresése.....                                                                                 | 611 |
| LINQ-lekérdezések: szigetek a szigetekeken? .....                                                         | 612 |
| Lekérdezéseredmények transzformálása Array típusúvá .....                                                 | 614 |
| Összefoglalás.....                                                                                        | 615 |

## 4. rész: Programozás .NET-szerelvényekkel

### 15. A .NET-szerelvények ..... 619

|                                                       |     |
|-------------------------------------------------------|-----|
| Egyedi névtér definiálása .....                       | 619 |
| A típus teljesen meghatározott neve .....             | 621 |
| Definiálás álnevek használatával.....                 | 623 |
| Beágyazott névterek létrehozása .....                 | 624 |
| A Visual Studio 2008 „alapértelmezett névtére” .....  | 625 |
| A .NET-szerelvények szerepe .....                     | 626 |
| A szerelvények a kód újrafelhasználását segítik ..... | 626 |
| A szerelvények típushatárokat definiálnak.....        | 627 |
| A szerelvények verzióval rendelkező egységek.....     | 627 |
| A szerelvények önleírók .....                         | 627 |
| A szerelvények konfigurálhatók.....                   | 628 |

|                                                                                    |     |
|------------------------------------------------------------------------------------|-----|
| A .NET-szerelvények felépítése.....                                                | 628 |
| A Win32-fejléc .....                                                               | 629 |
| A CLR-fejléc.....                                                                  | 630 |
| A köztes nyelvi kód, a típusmetaadat és<br>a szerelvénymanifesztum .....           | 631 |
| Opcionális szerelvény-erőforrások.....                                             | 631 |
| Egyfájlos és többfájlos szerelvények .....                                         | 632 |
| Egy állományból álló szerelvény készítése és használata.....                       | 634 |
| A manifesztum .....                                                                | 638 |
| A köztes nyelvi kód .....                                                          | 639 |
| A típusmetaadat.....                                                               | 640 |
| C#-ügyfélalkalmazás készítése .....                                                | 641 |
| Visual Basic ügyfélalkalmazás készítése .....                                      | 643 |
| A nyelvek közti származtatás .....                                                 | 644 |
| Több állományból álló szerelvény készítése és használata.....                      | 645 |
| Az ufo.netmodule fájl.....                                                         | 647 |
| Az airvehicles.dll fájl .....                                                      | 647 |
| A többfájlos szerelvény használata .....                                           | 648 |
| Privát szerelvények .....                                                          | 649 |
| A privát szerelvény azonosítója.....                                               | 650 |
| A keresési sorrend (probing) megadása .....                                        | 651 |
| Privát szerelvények beállítása .....                                               | 651 |
| Konfigurációs fájlok és a Visual Studio 2008.....                                  | 654 |
| A .NET Framework konfiguráló segédprogramja .....                                  | 655 |
| Megosztott szerelvények .....                                                      | 657 |
| Erős nevek megértése.....                                                          | 658 |
| A CarLibrary.dll ellátása erős névvel.....                                         | 660 |
| Erős nevek hozzárendelése a Visual Studio 2008 segítségével.....                   | 662 |
| Megosztott szerelvények telepítése/eltávolítása<br>a globális szerelvénytárba..... | 662 |
| A megosztott szerelvények felhasználása.....                                       | 664 |
| A SharedCarLibClient manifesztuma .....                                            | 665 |
| A megosztott szerelvények konfigurálása .....                                      | 666 |
| Az aktuális megosztott szerelvény befagyasztása.....                               | 667 |
| A megosztott szerelvény 2.0.0.0 verziójának elkészítése.....                       | 668 |
| Dinamikus átirányítás a megosztott szerelvények<br>meghatározott verzióiba .....   | 670 |
| A .NET Framework konfiguráló programja – ismét .....                               | 672 |

|                                                   |     |
|---------------------------------------------------|-----|
| A GAC belső felépítése .....                      | 673 |
| Kibocsátó házirenddel rendelkező szerelvény ..... | 674 |
| A kibocsátó házirend letiltása .....              | 676 |
| A <codeBase> elem .....                           | 677 |
| A System.Configuration névtér .....               | 679 |
| A gépi szintű konfigurációs fájl.....             | 680 |
| Összefoglalás.....                                | 681 |

## **16. Típusreflexió, késői kötés és attribútumalapú programozás..... 683**

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| A metaadatok szükségessége.....                                            | 683 |
| Az EngineState felsorolás (részleges) metaadata .....                      | 685 |
| A Car típus (részleges) metaadata .....                                    | 686 |
| A TypeRef .....                                                            | 687 |
| A definiáló szerelvény leírása .....                                       | 687 |
| A hivatkozott szerelvények leírása.....                                    | 688 |
| Sztringliterálok leírása.....                                              | 688 |
| A reflexió .....                                                           | 689 |
| A System.Type osztály .....                                                | 690 |
| Típusreferencia megszerzése a System.Object.GetType()<br>segítségével..... | 691 |
| Típusreferencia megszerzése a System.Type.GetType()<br>segítségével.....   | 692 |
| Típusreferencia készítése a typeof() segítségével.....                     | 693 |
| Egyedi metaadat-nézegető létrehozása .....                                 | 693 |
| A metódusok reflexiója .....                                               | 693 |
| Mezők és tulajdonságok reflexiója.....                                     | 694 |
| A típus által megvalósított interfészek reflexiója .....                   | 695 |
| Egyéb érdekességek megjelenítése .....                                     | 695 |
| A Main() metódus megvalósítása .....                                       | 695 |
| Generikus típusok reflexiója.....                                          | 697 |
| Metódusparaméterek és visszatérési értékek reflexiója .....                | 698 |
| Dinamikusan betöltődő szerelvények .....                                   | 700 |
| Megosztott szerelvények .....                                              | 702 |
| A késői kötés .....                                                        | 704 |
| A System.Activator osztály .....                                           | 705 |
| Paraméter nélküli metódusok hívása.....                                    | 706 |
| Paraméterekkel rendelkező metódusok hívása .....                           | 707 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| Programozás attribútumokkal .....                            | 708 |
| Attribútumok feldolgozása .....                              | 709 |
| Attribútumok alkalmazása a C# nyelvben.....                  | 710 |
| Attribútumok konstruktorparamétereinek meghatározása.....    | 712 |
| Az Obsolete attribútum működés közben.....                   | 713 |
| C#-attribútum rövidített formája.....                        | 713 |
| Egyedi attribútum készítése.....                             | 714 |
| Egyedi attribútumok alkalmazása.....                         | 715 |
| Nevesített tulajdonság szintaxisa .....                      | 716 |
| Az attribútumok használatának korlátozása .....              | 716 |
| Alkalmazásszintű (és modulszintű) attribútumok .....         | 718 |
| A Visual Studio 2008 AssemblyInfo.cs fájlja .....            | 718 |
| Attribútumok korai kötés használatával .....                 | 720 |
| Attribútumok késői kötés használatával.....                  | 721 |
| Reflexió, késői kötés és egyedi attribútumok kifejtése ..... | 723 |
| Kiterjeszthető alkalmazás készítése .....                    | 724 |
| A CommonSnappableTypes.dll létrehozása .....                 | 725 |
| C# beépülő modul létrehozása .....                           | 726 |
| Visual Basic beépülő modul készítése .....                   | 726 |
| Kiterjeszthető Windows Forms-alkalmazás készítése .....      | 727 |
| Összefoglalás.....                                           | 731 |

## **17. Folyamatok, alkalmazástartományok és objektumkörnyezetek..... 733**

|                                                        |     |
|--------------------------------------------------------|-----|
| A tradicionális Win32-es folyamat áttekintése .....    | 733 |
| A szálak áttekintése .....                             | 734 |
| Folyamatok kezelése a .NET platformban .....           | 736 |
| A futó folyamatok listázása .....                      | 739 |
| Egy adott folyamat vizsgálata .....                    | 740 |
| Egy folyamat szálkészletének vizsgálata.....           | 741 |
| Egy folyamat modulkészletének vizsgálata .....         | 743 |
| Folyamatok programozott indítása és leállítása .....   | 745 |
| A .NET-alkalmazástartomány .....                       | 746 |
| Folyamat alkalmazástartományainak felsorolása .....    | 749 |
| Új alkalmazástartományok programozott létrehozása..... | 750 |
| Alkalmazástartományok programozott eltávolítása .....  | 752 |
| Az objektumkontextus határai.....                      | 754 |
| Kontextusváltó és kötött típusok.....                  | 755 |

|                                                                     |     |
|---------------------------------------------------------------------|-----|
| Kötött objektum meghatározása .....                                 | 756 |
| Egy objektum kontextusának vizsgálata .....                         | 757 |
| A folyamat, alkalmazástartomány és a kontextus összefoglalása ..... | 759 |
| Összefoglalás .....                                                 | 759 |

## 18. Többszálú alkalmazások létrehozása ..... 761

|                                                                               |     |
|-------------------------------------------------------------------------------|-----|
| A folyamat, az alkalmazástartomány, a kontextus és<br>a szál kapcsolata ..... | 761 |
| A konkurencia problémája .....                                                | 763 |
| A szálszinkronizáció szerepe .....                                            | 764 |
| A .NET-metódusreferenciák rövid áttekintése .....                             | 764 |
| A metódusreferenciák aszinkron viselkedése .....                              | 767 |
| A BeginInvoke() és az EndInvoke() metódusok .....                             | 767 |
| A System.IAsyncResult interfész .....                                         | 768 |
| Metódushívás aszinkron módon .....                                            | 769 |
| A hívó szál szinkronizálása .....                                             | 770 |
| Az AsyncCallback metódusreferencia szerepe .....                              | 771 |
| Az AsyncResult osztály szerepe .....                                          | 773 |
| Egyedi állapotadatok átadása és fogadása .....                                | 774 |
| A System.Threading névtér .....                                               | 775 |
| A System.Threading.Thread osztály .....                                       | 777 |
| Hozzáférés az aktuális szál statisztikáihoz .....                             | 778 |
| A Name tulajdonság .....                                                      | 779 |
| A Priority tulajdonság .....                                                  | 780 |
| Második szál létrehozása kódból .....                                         | 781 |
| A ThreadStart metódusreferencia használata .....                              | 782 |
| A ParameterizedThreadStart metódusreferencia használata .....                 | 784 |
| Előtérben és háttérben futó szálak .....                                      | 786 |
| A konkurencia kérdése .....                                                   | 787 |
| Szinkronizáció a C# lock kulcsszavával .....                                  | 790 |
| Szinkronizáció a System.Threading.Monitor típussal .....                      | 792 |
| Szinkronizáció a System.Threading.Interlocked típussal .....                  | 793 |
| Szinkronizáció a [Synchronization] attribútummal .....                        | 795 |
| Programozás időzített visszahívással .....                                    | 796 |
| A CLR ThreadPool .....                                                        | 798 |
| A BackgroundWorker komponens .....                                            | 800 |
| A BackgroundWorker típus használata .....                                     | 801 |
| Adatfeldolgozás a BackgroundWorker típussal .....                             | 803 |
| Összefoglalás .....                                                           | 805 |

**19. A CIL és a dinamikus szerelvények szerepe ..... 807**

|                                                                  |     |
|------------------------------------------------------------------|-----|
| A CIL-programozás természete.....                                | 807 |
| CIL-direktívák, -attribútumok és -vezérlőkódok .....             | 808 |
| A CIL-direktívák szerepe.....                                    | 809 |
| A CIL-attribútumok szerepe .....                                 | 810 |
| A CIL-vezérlőkódok szerepe.....                                  | 810 |
| A CIL-vezérlőkódok/CIL-mnemonikok megkülönböztetése .....        | 810 |
| Verem írása/olvasása: A CIL veremalapú természete.....           | 811 |
| A round-trip engineering .....                                   | 813 |
| A CIL-kódcímkék szerepe .....                                    | 817 |
| Fejlesztés a CIL-lel: egy *.il fájl módosítása .....             | 818 |
| A CIL-kód fordítása ilasm.exe használatával .....                | 820 |
| CIL-kód fordítása SharpDevelop használatával.....                | 821 |
| A peverify.exe szerepe .....                                     | 823 |
| CIL-direktívák és -attribútumok .....                            | 823 |
| Külsőleg hivatkozott szerelvények meghatározása CIL-ben.....     | 823 |
| Az aktuális szerelvény definiálása CIL-ben.....                  | 824 |
| Névtér definiálása CIL-ben.....                                  | 825 |
| Osztálytípusok definiálása CIL-ben .....                         | 826 |
| Interfészek definiálása és implementálása CIL-ben .....          | 827 |
| Struktúrák definiálása CIL-ben.....                              | 828 |
| Felsorolt típusok definiálása CIL-ben .....                      | 829 |
| Generikus típusok definiálása a CIL-ben .....                    | 829 |
| A CILTypes.il fájl fordítása.....                                | 830 |
| .NET-alaposztálykönyvtár, C#- és CIL-adattípus-leképezések ..... | 832 |
| Típustagok definiálása CIL-ben.....                              | 833 |
| Adatmezők definiálása CIL-ben .....                              | 833 |
| Típuskonstruktorok definiálása CIL-ben.....                      | 834 |
| Tulajdonságok definiálása CIL-ben.....                           | 835 |
| Tagparaméterek definiálása .....                                 | 835 |
| A CIL-vezérlőkódok.....                                          | 836 |
| A .maxstack direktíva.....                                       | 839 |
| Lokális változók deklarálása CIL-ben.....                        | 840 |
| Paraméterek leképzése lokális változókba CIL-ben.....            | 841 |
| A rejtett „this” referencia .....                                | 842 |
| Ciklusok reprezentációja a CIL-ben .....                         | 843 |

---

|                                                                   |            |
|-------------------------------------------------------------------|------------|
| .NET-szerelvény készítése köztes nyelven.....                     | 844        |
| A CILCars.dll létrehozása.....                                    | 844        |
| A CILCarClient.exe létrehozása.....                               | 847        |
| Dinamikus szerelvények .....                                      | 850        |
| A System.Reflection.Emit névtér .....                             | 851        |
| A System.Reflection.Emit.ILGenerator szerepe.....                 | 852        |
| Dinamikus szerelvény generálása .....                             | 853        |
| A szerelvény- és a modulkészlet generálása.....                   | 856        |
| A ModuleBuilder típus szerepe .....                               | 857        |
| A HelloWorld típus és a sztringtípusú tagváltozó generálása ..... | 859        |
| A konstruktorok generálása .....                                  | 859        |
| A SayHello() metódus generálása .....                             | 861        |
| Dinamikusan létrehozott szerelvény használata.....                | 861        |
| Összefoglalás.....                                                | 863        |
| <b>Tárgymutató .....</b>                                          | <b>865</b> |
| <b>A szakmai lektorról .....</b>                                  | <b>879</b> |
| <b>A szerzőről.....</b>                                           | <b>881</b> |





# Köszönetnyilvánítás

Jóllehet egyedül az én nevem jelenik meg a könyv borítóján, de a munka soha nem készülhetett volna el jó néhány tehetséges ember segítségével nélkül. A következőkben azokhoz szólok, akik lehetővé tették, hogy ezt a könyvet megírjam.

Mindenekelőtt köszönet illeti az Apress összes munkatársát, akikkel már több éve van szerencsém együtt dolgozni. Rendkívüli emberek, akik képesek voltak az én eredeti Word-dokumentumaimat tökéletes prózává varázsolni. Hálás köszönet érte. Már most nagyon várom, hogy mikor dolgozhatunk ismét együtt egy következő könyvön (de azért előtte szabadságolom magam egy időre).

Külön köszönöm technikai szerkesztőm, Gavin munkáját, aki annyi bölcs tanáccsal segített, hogy úgy érzem, ez a kiadás sokkal színvonalasabb lett, mint az előzőek. Mint mindig, az esetleges gépelési hibákért vagy technikai tévedésekért egyedül én vagyok a felelős.

Végül, de nem utolsósorban, köszönöm családomnak, barátaimnak és munkatársaimnak, hogy elviselték mogorva viselkedésemet, amely a kézirat elkészítésének utolsó szakaszaiban jó néhányszor jellemzett engem.



# Bevezetés

Ez a könyv nagyjából 2001 nyara óta létezik, ha nem is ebben a formában: a C# és a .NET platform első kiadása ugyanis akkor jelent meg, a .NET 1.0 Beta 2 verziójával karöltve. Nagy örömmel és hálával tölt el, hogy munkám kedvező fogadtatásra lelt a sajtóban, és ami még fontosabb, az olvasók körében is. Az évek során jelölték a Jolt Award döntőjére (vesztettem... kellemetlen) és a 2003-as Referenceware Excellence Awardon a programozási könyvek kategóriájában (nyertem?... de jó!)

Azóta azon dolgoztam, hogy a könyvet frissítsem és aktualizáljam a .NET platform újabb verzióinak megfelelően. Eközben limitált darabszámmal megjelentettem egy speciális kiadványt, amely bemutatta a .NET 3.0 verzióját (Windows Presentation Foundation, Windows Communication Foundation, és a Windows Workflow Foundation), valamint áttekintést nyújtott olyan különböző technológiákról, amelyek még megjelenésre vártak, és amelyeket most LINQ néven ismerünk.

Jelenleg a könyv negyedik kiadását tartja a kezében az Olvasó. Ez a kiadás egyrészt tömören megismétli a korábbi ismereteket, azaz a .NET 3.5-ös verziójában található legfontosabb változások magyarázatát, másrészt számos teljesen új fejezetet is tartalmaz, de ezen túlmenően több eddigi rész is jelentősen kibővült.

Ahogy a korábbiakban megszokhattuk, ez a kiadás is egyszerű és érthető nyelven ismerteti a C# nyelvet és a .NET-alaposztálykönyvtárakat. Soha nem értettem azokat a szakírókat, akik úgy adják elő mondandójukat, hogy az jobban hasonlít egy GRE-szöszedetről írt tanulmányra, mint egy szakkönyvre. Munkám továbbra is arra fekteti a hangsúlyt, hogy a benne található információk segítségével jól használható szoftverrendszereket lehessen készíteni, így nem töltök túl sok időt az ezoterikus részletek tanulmányozásával, hiszen ezekkel csak nagyon kevés olvasóm fog valaha is foglalkozni.

## Mi egy csapat vagyunk, az Olvasó és én

A szakírók az emberek igényes csoportjának írnak (nekem már csak tudnom kell – egy vagyok közülük). Tisztában vagyunk azzal, hogy egy szoftver elkészítése – bármelyik platformmal történjen is (.NET, J2EE, COM stb.) – rendkívül aprólékos munka, ugyanakkor nagyon specifikus a részleg, a cég, az ügyfél vagy a tárgykör szerint. Hiszen lehet, hogy épp az elektronikus kiadói iparágban dolgozunk, vagy éppenséggel rendszert fejlesztünk az államnak, esetleg a helyi önkormányzatnak, a NASA-nak vagy egy katonai részlegnek. Én a magam részéről gyermekeket oktató szoftvert fejlesztettem, számos n-rétegű rendszert, valamint projekteket a gyógyászati és a pénzügyi ágazatokban. Majdnem nulla az esélye, hogy annak a kódnak, amelyet egyikünk ír a munkájához, köze lesz ahhoz a kódhoz, amelyet a másikunk ír a saját munkájához (kivéve, ha véletlen épp együtt dolgoztunk korábban!).

Ezért szándékosan kerülöm az olyan mintapéldákat, amelyeket konkrét iparághoz vagy programozási feladathoz köthetnénk: a C#, az objektumorientált programozás és a .NET 3.5 alapsztály könyvtárait ipari alkalmazásoktól független példákon keresztül mutatom be. Ahelyett, hogy minden példám adatok tömkelegét szolgáltatná, bérlistát számolna, vagy ehhez hasonlókkal szolgálna, az én példáimhoz mindenkinek van valami köze: autókról szólnak (ráadásként néhány geometriai struktúrával és alkalmazottakkal). És ez az a pont, ahol összetetalálkozunk.

Az *én feladatom*, hogy legjobb tudásom szerint elmagyarázzam a C# programozási nyelvet és a .NET platform alapvető jellegzetességeit. Ugyanígy mindent megteszek azért, hogy ellássam a kedves Olvasót különféle eszközökkel és stratégiákkal, amelyek lehetővé teszik, hogy a könyv alapján folytatni lehessen a téma tanulmányozását.

Az *olvasó feladata* pedig, hogy a megszerezett tudást képes legyen alkalmazni saját programozási feladatában. Persze egyértelmű, hogy a projektek nem állatnevekkel ellátott autók körül forognak, de hát pont erről szólna az alkalmazott ismeret. Biztos lehet benne az Olvasó, hogy ha egyszer megértette a könyvben bemutatott elveket, ismereteinek birtokában olyan .NET-alkalmazásokat írhat, amelyek használhatók saját programozási környezetükben.

# A könyv áttekintése

A *Pro C# 2008 and the .NET 3.5 Platform* (A C# és a Microsoft .NET 3.5.) negyedik kiadása logikailag nyolc különálló részből áll, s ezek mindegyike további fejezeteket tartalmaz. Jó néhány témakör, például az alapvető C#-szerkezetek, az objektumorientált programozás és a platformfüggetlen .NET fejlesztése kibővült a korábbiakhoz képest, saját fejezetet kapott. Az új kiadás a .NET 3.0–3.5 programozás tulajdonságait (LINQ, WCF, WPF, WF stb.) összefoglaló fejezeteket is tartalmaz.

---

**Megjegyzés** A magyar változat kiadásának előkészítése során a SZAK Kiadó úgy döntött, hogy az eredeti könyv tartalmát két kötetben jelenteti meg. Ezt a viszonylag nagy terjedelem indokolja. Az alábbiakban mind a két kötet tartalmát ismertetjük, így biztosítva, hogy az is képet kapjon a mű egészéről, akinek csak az egyik kötet kerül a kezébe. (Ezt a Bevezetést változatlan formában mind a két kötetben szerepeltetjük.)

---

## Első kötet

### 1. rész: Bevezetés a C#-ba és a .NET platformba

Az első rész bemutatja a .NET platform alapvető természetét és számos olyan fejlesztőeszközt (köztük több nyílt forráskódút), amelyeket .NET-alkalmazások fejlesztéséhez használhatunk. Mindeközben néhány alap C# programozási nyelvi részlettel és a .NET-típusrendszerrel is megismerkedünk.

#### 1. fejezet: A .NET filozófiája

Az első fejezet voltaképpen a könyv további részeinek a gerincét képezi. A hagyományos Windows-fejlesztések világának vizsgálatával kezdjük, és feltárjuk a korábbi hátrányokat is. A fejezet elsődleges célja azonban az, hogy megismerjünk jó néhány .NET-központú építőkövet, például az egységes nyelvi futtatót (Common Language Runtime – CLR), a közös típusrendszert (Common Type System – CTS), az egységes nyelvi specifikációt (Common Language Specification – CLS) és az alaposztálykönyvtárakat. Egyszerű betekintést nyerhetünk a C# programozási nyelvbe és a .NET-szerelvényformátumába, valamint áttekintést kapunk a .NET platform platformfüggetlen természetéről (a B függelék részletesen tárgyalja ezt a témakört).

## 2. fejezet: C#-alkalmazások készítése

A fejezet célja, hogy különféle eszközök és technikák ismertetésével bevezessen minket a C#-forráskódfájl fordításának folyamatába. Először a parancssoros fordító (`csc.exe`) és a C#-eredményfájlok használatát ismerhetjük meg. A fejezet további részében pedig számos forráskódszerkesztőt és integrált fejlesztési környezetet (integrated development environments – IDEs) tanulmányozhatunk, többek között a következőket: TextPad, Notepad++, Sharp-Develop, Visual C# 2008 Express és a Visual Studio 2008. Ezenkívül számos olyan további programozási eszközt ismertetünk, amelyet bármely .NET-fejlesztőnek azonnal elő kell tudnia húzni a farzsebéből.

## 2. rész: A C# alapvető építőelemei

A 2. részben tárgyalt témakörök kiemelten fontosak, hiszen az itt megjelenő tudnivalókat folyamatosan használjuk függetlenül attól, hogy milyen jellegű .NET-szoftvert szándékozunk fejleszteni (webes alkalmazás, asztali grafikus alkalmazás, forráskódkönyvtárak, Windows-szolgáltatások stb.). Itt derül ki, hogy hogyan működnek a C# nyelv alapvető építőkövei, beleértve az objektumorientált programozás (OOP) részleteit is. Ez a rész ismerteti továbbá, hogy hogyan kell kezelni a futásidejű kivételeket, és bevezet minket a .NET szemétgyűjtő szolgáltatásának részleteibe is.

### 3. fejezet: A C# alapvető építőelemei, I. rész

Ebben a fejezetben kezdjük el részletesen tanulmányozni a C# programozási nyelvet. Megismerkedünk a `Main()` metódus szerepével és számos részlettel a .NET platform belső adattípusaival kapcsolatban, beleértve a szövegesadatkezelést a `System.String` és a `System.Text.StringBuilder` névterek alkalmazásával. Megvizsgáljuk továbbá az iterációs és a döntési szerkezeteket, a szűkítő és a bővítő műveleteket, valamint az `unchecked` kulcsszó használatát.

### 4. fejezet: A C# alapvető építőelemei, II. rész

A 4. fejezet befejezi az alapvető C#-szemléletek meghatározását, a típustúlterhelt metódusok létrehozásától az `out`, `ref` és `params` kulcsszavakon keresztüli paraméterek definiálásáig. Vizsgáljuk, hogyan kell adathalmazt létrehozni és kezelni, hogyan kell olyan adattípust készíteni, amelynek `null` is lehet az értéke (a `?` és a `??` operátorokkal), és megmutatjuk az *értéktípus* és a *referenciatípus* közötti különbséget is.

## 5. fejezet: Egységbe zárt osztálytípusok definiálása

Ebben a fejezetben az objektumorientált programozással (OOP) ismerkedhetünk meg a C# programozási nyelv használatán keresztül. Először meghatározzuk az OOP pilléreit (beágyazás, származtatás és polimorfizmus), majd a fejezet további részében azt nézzük meg, hogyan kell konstruktorok, tulajdonságok, statikus tagváltozók, konstansok és csak olvasható fájlok használatával robusztusosztály-típusokat létrehozni. Végezetül ismertetjük a részleges típus definícióját és a C# XML-kódjának dokumentációs szintaktikáját.

## 6. fejezet: A származtatás és a polimorfizmus

Ebben a részben az OOP maradék pilléreit (származtatás és polimorfizmus) vizsgáljuk meg; ezek a segítségével összetartozó osztálytípusok családját hozhatjuk létre. Megnézzük a virtuális metódusok és absztrakt metódusok (és az absztrakt alaposztályok) szerepét, valamint a *polimorf interfész* természetét. Végül, de nem utolsósorban, ez a fejezet ismerteti a .NET platform legfontosabb alaposztályának, a `System.Object` osztálynak a szerepét is.

## 7. fejezet: Strukturált hibakezelés

Ez a fejezet elsősorban azt mutatja be, mit kell tenni a strukturális kivételkezelés segítségével a futási időben keletkezett anomáliákkal. Nemcsak az ilyen jellegű problémák megoldására szolgáló C#-kulcsszavakat ismerjük itt meg (`try`, `catch`, `throw` és `finally`), hanem az alkalmazásszintű és a rendszerszintű kivételek közötti különbséget is. Megismerhetünk továbbá többféle, a Visual Studio 2008-ban található eszközt, amelyek segítségével hibakeresést végezhetünk a figyelmen kívül hagyott kivételek között.

## 8. fejezet: Az objektumok életciklusa

Ernek a résznek az utolsó fejezete azt tárgyalja, hogy hogyan kezeli a CLR a memóriát a .NET szemétyűjtőjének a segítségével. Megérthetjük az alkalmazásgyökerek, az objektumnemzedékek és a `System.GCType` szerepét. Az alapok megismerése után, a fejezet további része az *eldobható objektumokkal* (az `IDisposable` interfészen keresztül) és a véglegesítő folyamattal (a `System.Object.Finalize()` metóduson keresztül) foglalkozik.

### 3. rész: Haladó programozási szerkezetek a C#-ban

A könyv 3. része ismerteti a C# nyelv haladóbb jellegű (de ugyancsak nagyon fontos) elveit. Az interfészek és metódusreferenciák vizsgálatával befejezzük a .NET típusrendszerével való ismerkedést. A továbbiakban betekintést nyerhetünk a generikus típusok és a számos új C# 2008 nyelvi elem szerepébe, valamint a LINQ-ba is.

#### 9. fejezet: Interfészek használata

Ennek a fejezetnek az anyagát az interfészalapú programozást is magába foglaló objektumalapú fejlesztés alkotja. Megismerhetjük belőle, hogy hogyan kell olyan típusokat definiálni, amelyek többszörös viselkedést támogatnak, hogyan lehet ezeket a viselkedéseket futásidőben feltérképezni, és hogyan lehet szelektíven elrejtetni bizonyos viselkedéseket az *explicit interfészmegvalósítás* segítségével. Továbbá a számos előre definiált .NET-interfész típus megismeréséhez el kell sajátítanunk azoknak az egyedi interfészeknek a használatát, amelyek ad hoc jellegű eseményarchitektúra készítéséhez szükségesek.

#### 10. fejezet: Gyűjtemények és generikus típusok

Ez a fejezet a `System.Collections` névtér gyűjtemény típusainak tanulmányozásával kezdődik; ezek már a .NET platform kezdeti kiadásának is részei voltak. Noha a .NET 2.0 megjelenése óta a C# programozási nyelv lehetővé teszi generikus típusok használatát, ahogy a későbbiekben kiderül, a generikus programozás lényegesen megnöveli az alkalmazás teljesítményét és a típusbiztonságot. Nemcsak azt vizsgáljuk meg, hogy milyen különböző generikus típusok léteznek a `System.Collections.Generic` névtérben, de azt is, hogyan készíthetjük el saját generikus metódusainkat és típusainkat (megszorításokkal vagy anélkül).

#### 11. fejezet Metódusreferenciák, események és lambdák

A 11. fejezetnek az a célja, hogy érthetővé tegye a *metódusreferencia*-típusokat. Leegyszerűsítve: a .NET-metódusreferencia olyan objektum, amely az alkalmazás más metódusaira mutat. Ez a minta olyan rendszerek készítését teszi lehetővé, amelyek több objektum összekapcsolódását engedik meg kétirányú együttműködés segítségével. A .NET-metódusreferenciák vizsgálata után megismerkedünk a .NET event kulcsszavával, amelyet arra használunk, hogy egyszerűsítsük a nyers metódusreferencia programozásának kezelését. Végezetül megvizsgáljuk a C# 2008 lambda-operátort ( $\Rightarrow$ ), és feltárjuk a kapcsolatot a metódusreferenciák, a névtelen metódusok és a lambda-kifejezések között.



## 12. fejezet: Indexerek, operátorok és mutatók

Ez a fejezet a C# programozási ismereteinket mélyíti el számos haladó programozási technika segítségével. Megismerhetjük, hogyan kell operátorokat túlterhelni, és hogyan kell a típusaink számára egyedi konverziós rutinokat írni (akár implicit, akár explicit formában). Megtanulhatjuk azt is, hogy hogyan kell létrehozni és használni a *típusindexelőket*, és hogyan kell kezelni a C-stílusú mutatókat a „nem biztonságos” kódkörnyezetben.

## 13. fejezet: C# 2008 nyelvi újdonságai

A .NET 3.5 kiadásában a C# nyelv számos új programozási eszközzel bővült, amelyek közül jó néhány arra szolgál, hogy lehetővé tegye a LINQ API használatát (erről a 14. fejezetben többet is megtudhatunk). Megismerjük a lokális változók implicit típussal történő ellátásának a szerepét, a részleges metódus, az automatikus tulajdonságok, a bővítő metódusok, a névtelen típusok és az objektuminicializáló szintaxis használatát.

## 14. fejezet: Bevezetés a nyelvbe ágyazott lekérdezésekbe (LINQ)

Ez a fejezet ismerteti a LINQ-et (Language Integrated Query – nyelvi integrált lekérdezés), amelyet nyugodtan nevezhetünk a .NET 3.5 legérdekesebb részének. Ahogy a fejezet későbbi részéből kiderül, a LINQ segítségével lehetőségünk nyílik erősen típusos *lekérdezéseket* írni, amelyeket számos LINQ-célpontra alkalmazhatunk, hogy a szó legtágabb értelmében „adatot” kezelhessünk. Megtanuljuk a LINQ objektumokra történő alkalmazását, ennek segítségével a LINQ-kifejezéseket adattárolókra (tömbök, kollekciók, egyedi típusok) fogjuk tudni alkalmazni. Ez a tudás nélkülözhetetlen lesz akkor, amikor azt vizsgáljuk meg, hogyan kell LINQ-kifejezéseket relációs adatbázisokon (a LINQ az ADO-n keresztül) és XML-dokumentumokon (a LINQ az XML-en keresztül) használni.

## 4. rész: Programozás .NET-szerelvényekkel

A negyedik rész a .NET-szerelvényformátum részleteit taglalja. Nemcsak azt tanulhatjuk meg, hogyan kell .NET-kódkönyvtárakat telepíteni és konfigurálni, de azt is, hogy mi a .NET bináris kép belső összeállítása. Ebben a részben lesz szó a .NET-attribútumok szerepéről és a többszálú alkalmazások létrehozásának módjáról is. A későbbi fejezetek néhány haladó témát érintenek, például az objektumkörnyezetet, a köztes nyelvi kódot és a dinamikus szerelvényeket.

## 15. fejezet: A .NET-szerelvények

Első megközelítésben a *szerelvény* nem más, mint egy felügyelt \*.dll vagy \*.exe bináris fájl leírására szolgáló kifejezés. Ám a .NET-szerelvények története sokkal több ennél. Ebben a fejezetben megtudjuk, mi az egyfájlos és a többfájlos szerelvények közötti különbség, és, hogy ezeket hogyan kell készíteni és telepíteni. Megvizsgáljuk, hogyan kell privát és megosztott szerelvényeket konfigurálni XML-alapú \*.config fájlokkal és kibocsátó házirenddel rendelkező szerelvényekkel. Mindeközben megtudhatjuk, hogy milyen a GAC (Global Assembly Cache – globális szerelvénytár) belső struktúrája, és mi a .NET keretrendszer konfigurációs segédprogramjainak a szerepe.

## 16. fejezet: Típusreflexió, késői kötés és attribútumalapú programozás

A 16. fejezetben folytatjuk az ismerkedést a .NET-szerelvényekkel. A System.Reflection névtér használatával kapcsolatban azt vizsgáljuk, a futási időben hogyan valósul meg a típusmeghatározás folyamata. Az ilyen jellegű típusok használatával lehetőségünk van olyan alkalmazások létrehozására, amelyek futás közben képesek a szerelvények metaadatait olvasni. Kiderül, hogyan lehet a *késői kötés* segítségével futási időben dinamikusan betölteni és létrehozni típusokat. A fejezet utolsó témaköre – mind a szabványos, mind az egyedi – .NET-attribútumok szerepének ismertetése. Az egyes témakörök hasznosságának szemléltetésére a fejezet konklúziójaként bemutatunk egy kiterjeszthető Windows Forms-alkalmazást.

## 17. fejezet: Folyamatok, alkalmazástartományok és objektumkörnyezetek

A szerelvényekről szóló tudnivalók után ez a fejezet a betöltött .NET végrehajtható fájlok felépítését ismerteti. Az elsődleges cél a folyamatok, az alkalmazástartományok és a kontextusbeli határok közötti kapcsolat illusztrálása. Ezek alapozzák meg a következő fejezet témáját: a többszálú alkalmazások készítésének módját.

## 18. fejezet: Többszálú alkalmazások készítése

Ez a fejezet azt mutatja be, hogy hogyan kell többszálú alkalmazásokat készíteni, és számos olyan technikát ismertet, amelyeket a szálbiztos kód készítéséhez alkalmazhatunk. A fejezet elején átvizsgáljuk a .NET metódusreferencia-típusait, hogy megértsük, valójában hogyan támogatják a metódusreferenciák

az aszinkronfüggvény-hívásokat. Majd megvizsgáljuk a `System.Thread` névtérben található típusokat. Ezek között számos olyan van (`Thread`, `ThreadStart`), amelynek segítségével könnyen lehet újabb végrehajtási szálakat készíteni. Végül megismerkedünk a `BackgroundWorker` típussal, amely nagyban megkönnyíti a felhasználói felületről történő szálkészítést.

## 19. fejezet: A köztes nyelv (CIL) és a dinamikus szerelvények

A 4. rész utolsó fejezetének szerepe kettős. Először a korábbinál részletesebben megvizsgáljuk CIL szintaktikáját és szemantikáját, majd a `System.Reflection.Emit` névtér szerepét mutatjuk be. Ezeknek a típusoknak a használatával lehetőségünk nyílik olyan szoftver készítésére, amely képes .NET-szerelvényeket futásidőben generálni a memóriába. Az olyan szerelvényeket, amelyeket a memóriában definiálunk és futtatunk, *dinamikus szerelvényeknek* nevezzük.

## Második kötet

## 5. rész: Bevezetés a .NET alaposztálykönyvtáraiba

Ha elérkeztünk ehhez a részhez, akkor már biztos tudásunk van a C# nyelvről és a .NET- szerelvényformátumok részleteiről. Az ötödik rész továbbfejleszti ezeket az ismereteket azáltal, hogy számos általánosan használt alaposztálybeli szolgáltatást mutat be, többek között a fájl I/O-t és az adatbáziselérést az ADO.NET segítségével. Ez a rész tárgyalja továbbá az elosztott alkalmazások készítésének témakörét a Windows Communication Foundation (WCF) segítségével, valamint azokat a munkafolyamat-engedélyezett alkalmazások készítését, amelyek a Windows Workflow Foundation (WF) API-t használják.

## 20. fejezet: Fájlműveletek és elkülönített tárolás

A `System.IO` névtér segítségével lehetőségünk van a számítógép fájl- és könyvtárszerkezetének elérésére. Ebben a fejezetben megismerjük, hogyan kell programozottan könyvtárrendszert létrehozni (és törölni), valamint, adatokat különböző folyamatba (fájlalapú, sztringalapú, memóriaalapú stb.) be-, illetve kimotozni. A fejezet utolsó része megvizsgálja az *elkülönített tárolást*, amelynek segítségével lehetőségünk van biztonságos helyen külön tárolni az adatokat függetlenül attól, hogy a célgép biztonsági beállításaitól. Hogy megértsük a `System.IO.Isolated` API bizonyos elveit, áttekintést kell kapnunk a kódhozzáférés-szabályozásról (Code Access Security – CAS).

## 21. fejezet: Bevezetés az objektumsorosítás világába

Ez a fejezet a .NET-platform objektumsorosító szolgáltatásába nyújt betekintést. Egyszerűen szólva a *sorosítás* segítségével lehetőségünk nyílik arra, hogy egy objektum (vagy a kapcsolódó objektumok egy halmazának) állapotát egy folyamban tároljuk a későbbi használat céljából. A *deszerializálás* (a sorosítás ellentétes művelete) az a folyamat, amelynek segítségével kivehetjük az objektumot a folyamból, és a memóriában az alkalmazás számára használható formába hozhatjuk. Az alapok elsajátítása után megtanulhatjuk, hogyan kell egyedivé tenni a sorosítási folyamatot az *ISerializable* interface és a .NET-attribútumok halmazának segítségével.

## 22. fejezet: ADO.NET 1. rész: Az élő kapcsolat

Az adatbázisokról szóló két fejezet közül az elsőben az ADO.NET programozási API-ról kaptunk ismereteket. Ez a rész bevezeti a .NET adatszolgáltató szerepét, valamint azt, hogy hogyan kell a relációs adatbázissal kommunikálni az ADO.NET *élő kapcsolatán* keresztül. Ez a kapcsolati objektumokat, a parancsobjektumokat és az adatolvasó objektumokat jelenti. Ezenkívül ez a fejezet mutatja be az egyedi adatbázis létrehozásának módját, és ismerteti azokat az adatelérési könyvtárakat, amelyeket majd a könyv további részében használunk.

## 23. fejezet: ADO.NET 2. rész: A bontott kapcsolat

Ez a fejezet az adatbázis-kezelés különböző módokon történő manipulálási lehetőségeit tárgyalja tovább, jelen esetben az ADO.NET *bontott kapcsolatán* keresztül. Megismerkedhetünk a *DataSet* típus és az adatfeldolgozó objektumok szerepével, valamint számos Visual Studio 2008 eszközzel, amelyek nagyban megkönnyítik az adatvezérelt alkalmazások létrehozását. Mindeközben rávilágítunk arra, hogyan kell a *DataTable*-objektumokat felhasználni felületi elemekhez kapcsolni, olyanokhoz, mint például a Windows Forms API *GridView* típusa.

## 24. fejezet: A LINQ API programozása

A 24. fejezet a LINQ programozási modellt mutatja be, pontosabban a LINQ „az objektum irányába” részét. Megvizsgáljuk, hogyan kell LINQ-lekérdezéseket alkalmazni relációs adatbázisokon, *DataSet*-objektumokon és XML-dokumentumokon. Mindeközben megtanuljuk az *adatokontextus*-objektum, valamint az *sqlmetal.exe* segédprogram szerepét, és megvizsgáljuk a Visual Studio 2008 különböző LINQ-támogató funkcióit.

## 25. fejezet: A WCF

A .NET 3.0 bevezetett egy teljesen új API-t, a WCF-et, amely lehetőséget ad arra, hogy szimmetrikus módon, függetlenül az alatta található rétegek felépítésétől elosztott alkalmazást fejlesszünk. Ez a fejezet feltárja a WCF-szolgáltatások, -gazdagépek és -kliensek készítésének lehetőségeit. A WCF-szolgáltatások rendkívül flexibilisek, a gazdagépek és a kliensek XML-alapú konfigurációs fájlokat használhatnak, hogy deklaratív módon adjanak meg címeket, kötéseket és szerződéseket.

## 26. fejezet: A WF

A .NET 3.0 a WCF mellett bevezette a WF API-t is, amely lehetőséget ad arra, hogy *munkafolyamatokat* definiáljunk, futtassunk és monitorozzunk. Ennek segítségével komplex üzleti folyamatok modellezhetők. Megtanuljuk a WF általános célját, ahogy azt is, hogy mi a szerepe az aktivitásoknak, a munkafolyamat tervezőnek, a munkafolyamat futtatási motornak, és a munkafolyamat-engedélyezett forráskód könyvtáraknak.

# 6. rész: Felhasználói felületek

Általánosan hibás feltételezés a .NET platformmal kapcsolatban az, hogy ez a keretrendszer kizárólag a webalapú felhasználói felületekkel foglalkozik (ez feltehetően a „.NET” kifejezés miatt van, hiszen felidézi az „Internet” szót, azaz a „webprogramokat”). A .NET kiemelkedő módon támogatja a webalkalmazások készítését, a könyv ezen része azonban a hagyományos felhasználói felületekre koncentrál két grafikus felület keretrendszerén, a Windows Forms és a WPF használatán keresztül.

## 27. fejezet: Windows Forms-programozás

Az eredeti asztali grafikus felhasználói felület, amely a .NET platformmal együtt kapható, a *Windows Forms*. Ez a fejezet ismerteti a felhasználói felület keretrendszerének szerepét, és bemutatja, hogyan kell főablakokat, párbeszédablakokat és menürendszereket készíteni. Feltárja továbbá az űrlapszármaztatás szerepét, és azt, hogy hogyan kell kétdimenziós adatot rajzolni a *System.Drawing* névtér segítségével. A témakör illusztrálására a fejezet végén készítünk egy (korlátozott funkcionalitással bíró) rajzolóalkalmazást.

## 28. fejezet: A WPF és az XAML

A .NET 3.0 teljesen új grafikus felhasználói felületet vezetett be, amelyet *WPF*-nek nevezett el. Röviden, a WPF kimagaslóan interaktív és médiában gazdag front endek készítését teszi lehetővé asztali alkalmazásokhoz (és indirekt módon webalkalmazásokhoz). A Windows Formsszal ellentétben, ez a túltelített felhasználóifelület-keretrendszer számos kulcsfontosságú szolgáltatást (2D-s és 3D-s grafika, animációk, gazdag dokumentumok stb.) integrál egyetlen egységesített objektummodellbe. Ebben a fejezetben megkezdjük az ismerkedést a WPF-fel és a kiterjeszthető alkalmazásjelölő nyelvvel (Extendable Application Markup Language – XAML). Megtanuljuk, hogyan kell XAML-mentes, csak XAML-t használó és a kettő kombinációjából felépülő WPF-programokat létrehozni. A fejezet végén készítünk egy egyedi XAML-nézegetőt, amelyet a további WPF-fel foglalkozó fejezetekben is használni fogunk.

## 29. fejezet: Programozás WPF-vezérlőelemekkel

Ebben a fejezetben megtudjuk, hogy hogyan kell dolgozni a WPF-vezérlőelem tartalommodelljével, és érintünk számos vezérlőelemekkel kapcsolatos témát is, például a függőségi tulajdonságokat és irányított eseményeket. Jó néhány WPF-vezérlőelem használatát bemutatjuk, s szintén ebben a fejezetben magyarázzuk el az elrendezéskezelő, -vezérlő utasítások és a WPF-adatkötés modelljének a használatát.

## 30. fejezet: WPF 2D grafikus renderelés, erőforrások és témák

Az utolsó fejezet a WPF tárgyalását három látszólag független téma vizsgálatával zárja. A WPF grafikus renderelő szolgáltatásának általában szüksége van arra, hogy egyedi erőforrásokat definiáljunk hozzá. Ezeknek az erőforrásoknak a használatával lehetőségünk van egyedi WPF-animációk készítésére, és grafikát, erőforrásokat és animációkat alkalmazva egyedi témák készítését fogjuk tudni megoldani WPF-alkalmazások számára. A különböző témák összegzésekképpen a fejezet utolsó részében azt illusztráljuk, hogyan kell futásidőben egyedi grafikus témákat alkalmazni.

## 7. rész: Webes alkalmazások fejlesztése ASP.NET segítségével

A 7. rész az ASP.NET programozási API használatával készített webes alkalmazásokat vizsgálja. Az ASP.NET szándékosan úgy lett kialakítva, hogy az asztali felhasználói felület létrehozását modellezze, úgy, hogy szabványos HTTP kérésekre/válaszokra helyez egy eseményvezérelt, objektumorientált keretrendszert.

### 31. fejezet: ASP.NET weboldalak készítése

Ez a fejezet vezet be bennünket az ASP.NET használatával történő webes alkalmazásfejlesztésbe. A kiszolgálóoldali szkripteket felváltják az igazi objektumorientált nyelvek (mint a C#, VB.NET és hasonló). Bemutatjuk egy ASP.NET weboldal készítését, valamint az alatta található programozási modellt és az ASP.NET egyéb kulcsfontosságú elveit, például azt, hogy hogyan kell webkiszolgálót választani, valamint a `web.config` fájlok használatát.

### 32. fejezet: ASP.NET-vezérlőelemek, témák és mesteroldalak

Ez a fejezet azokkal a vezérlőelemekkel foglalkozik, amelyek a belső vezérlőtöltik fel adattal. Megismerjük az alapvető ASP.NET webes vezérlőelemeket, többek között a ellenőrző vezérlőelemet, a belső oldali navigációs vezérlőelemeket és a különböző adatkötő műveleteket. Ugyancsak ennek a fejezetnek a feladata, hogy bemutassa a *mesteroldalak* és az ASP.NET témamotorjának a szerepét, amely a hagyományos stíluslapok szerveroldali megfelelője.

### 33. fejezet: ASP.NET állapotkezelési technikák

Ez a fejezet kiterjeszti az eddigi ASP.NET-ismereteinket, ugyanis megvizsgálja, milyen különböző módokon lehet állapotot kezelni a .NET alatt. Ahogy a klasszikus ASP, az ASP.NET is egyszerűen teszi lehetővé sűtik létrehozását, ahogy alkalmazásszintű és munkafolyamat-szintű változók használatát is. Az ASP.NET azonban bevezet egy új állapotkezelő technikát: az alkalmazás-gyorsítótárat. Ha már tudjuk, milyen módokon lehet állapotot kezelni az ASP.NET alatt, megismerkedünk a `System.HttpApplication` alapsztály szerepével (a `Global.asax` fájlban belül), valamint azzal, hogy hogyan kell dinamikusan megváltoztatni a webalkalmazásunk futásidejű viselkedését a `web.config` fájl segítségével.

## 8. rész: Függelék

A könyv utolsó része két fontos témát vizsgál meg, amelyek valójában nem illeszkedtek szervesen a könyv felépítésébe, ezért a függelékbe kerültek. A függelék kiteljesíti a C#-pal és .NET platformmal kapcsolatos ismereteinket, megvizsgáljuk ugyanis, hogyan kell régebbi kódokat integrálni a .NET-alkalmazásainkba, valamint hogyan kell a .NET-fejlesztést a Windows operációs rendszer családján kívül használni.

### A függelék: A COM és a .NET együttműködése

Azok, akik már programoztak Windows operációs rendszer alatt a .NET platform használata előtt, nagy valószínűséggel ismerik a Component Object Modelt (COM). Noha a COM-nak és a .NET-nek semmi köze egymáshoz (azon túl, hogy mindkettő a Microsoft szerzeménye), a .NET platform teljes névteret biztosít ahhoz (`System.Runtime.InteropServices`), hogy .NET-szoftverek COM-komponenseket használjanak, és fordítva. Ez a függelék az együttműködési réteget mutatja be egészen részletesen; ez a téma azért nagyon fontos, hogy a korábban megírt programjainkat át tudjuk menteni az újonnan készített .NET-alkalmazásainkba.

### B függelék: Platformfüggetlen .NET-fejlesztés a Monóval

Végül, de nem utolsó sorban a B függelék a .NET nyílt forráskódú implementációjával, a *Monóval* foglalkozik. A Mono segítségével lehetőségünk van gazdag tulajdonságokkal rendelkező .NET-alkalmazás létrehozására, telepítésére és futtatására különböző operációs rendszereken, mint például a Max OS X, Solaris, AIX és számos Linux-disztribúció. Mivel a Mono nagyjából kompatibilis a Microsoft .NET platformjával, így már tisztában vagyunk azazal, hogy mit tud nyújtani számunkra. Ezért a függelék a Mono installálását tárgyalja, valamint azt, hogy, milyen fejlesztői eszközök állnak a rendelkezésünkre, és hogyan működik a Mono futtatómotorja.



## Öt szabadon letölthető fejezet – még több információ

Azoknak, akiknek a 33 fejezet és a két függelék nem lenne elég, szabadon letölthetnek további öt fejezetet. A könyv korábbi verziói három olyan fejezetet tartalmaztak, amelyeket a Windows Forms fejlesztésének szenteltünk (egyedi vezérlők vizsgálatával egyetemben), egy másik fejezetet a .NET remoting rétegének (System.Runtime.Remoting és társai), és a végül egy fejezet a hagyományos XML-webszolgáltatások készítésével foglalkozott az ASP.NET Web Service projekt sablonjainak a segítségével.

A könyv jelen kiadása nem tartalmazza ezt az öt fejezetet. Ennek az a legfőbb oka, hogy a .NET 3.0-ban a WCF és a WFP API-k rendre a .NET-remoting/XML-webszolgáltatások és a Windows Forms API-k örökösei. Ha mélyebben szeretnénk elmerülni a Windows Forms világában (azon túl, amit a 27. fejezet nyújtani tud), vagy meg akarjuk nézni, hogyan kell használni a (örökségül hagyott) .NET-remotingot és az XML-webszolgáltatás-API-t, egyszerűen csak meg kell nézni az Apress honlapján a megfelelő részt:

<http://apress.com/book/view/1590598849>.

Az itt található linke kattintva digitális formában letölthetjük a könyv további fejezeit, ha sikerült válaszolni a könyv szövegéből véletlenszerűen feltett kérdésekre. (A letölthető fejezetek csak angol nyelven érhetők el – a SZAK Kiadó megjegyzése).

## A könyv forráskódjának igénylése

A könyvben szereplő összes forráskód-mintapélda (az öt szabadon letölthető további fejezetben levők is) szabadon és azonnal elérhető az Apress honlapjának Source Code/Download részlegéről. Egyszerűen gépeljük be a <http://www.apress.com> címet, válasszuk ki a Source Code/Download linket, és keressük meg a megfelelő címet, és, töltsük le az önmagát kicsomagoló \*.zip fájlt. A forráskódokat fejezetenként osztottuk fel.

Figyelem: a fejezetekben a Forráskód megjelölés, mint amilyen az alábbi is, arra utal, hogy a szóban forgó példákat be lehet tölteni a Visual Studio 2008-ba további vizsgálódások és módosítások céljából:

---

**Forráskód** Ez a forráskód-megjelölés egy kiválasztott könyvtárra hivatkozik.

---

Egyszerűen nyissuk meg a \*.sln fájlt a megfelelő alkönyvtárból. Ha nem használjuk a Visual Studio 2008-at (lásd a 2. fejezetet további integrált fejlesztői környezetekért), akkor manuálisan töltjük be a felkínált forráskódfájlokat a választott fejlesztői eszközbe.

---

**Megjegyzés** A magyar nyelvű változatban a forráskódok megjegyzéseit lefordítottuk. A letölthető forráskódok fordítása nem állt módunkban, így azok teljesen angol nyelvűek. Ilyen módon inkább megfelelnek a letölthető kódok használati céljának – a SZAK Kiadó megjegyzése.

---

## A lehetséges javítások

Elképzelhető, hogy a könyv végigolvasása során nyelvtani vagy forráskódbeli hibákat vél felfedezni a kedves Olvasó, bár remélem, hogy erre nem kerül sor. Ha mégis, elnézést kérek. Az aktuális hibalistát az Apress weboldáról lehet igényelni (amely szintén a könyv honlapján található meg), és ugyancsak itt található az elérhetőségem, amelyen értesíteni tudnak.

## Elérhetőségem

Ha bármilyen kérdés merül fel a könyvhöz tartozó forráskódokkal kapcsolatban, vagy szükség van valamelyik példa tisztázására, vagy egyszerűen csak véleményt szeretne nyilvánítani a .NET platformról, nyugodtan küldjön levelet a következő e-mail címre (hogy biztosak legyünk benne, hogy a levele nem kerül a kéretlen levelek közé, kérem, tegye a tárgymezőbe a „C#FE” szöveget): [atroelsen@Intertech.com](mailto:atroelsen@Intertech.com).

Annak ellenére, hogy megpróbálok minden levélre lehetőség szerint válaszolni, sajnos, ahogy mindnyájan, időnként nagyon elfoglalt vagyok. Ha nem válaszolok egy-két héten belül, akkor ez nem jelenti azt, hogy nem akarok foglalkozni a problémával, csak épp nagyon nem érek rá (vagy ha szerencsém van, valahol épp nyaralok).

Nos, akkor, köszönöm, hogy megvásárolta a könyvemet (vagy legalábbis, hogy belenéz a könyvesboltban, miközben azon gondolkodik, hogy megvegye-e). Remélem, élvezni fogja a kedves Olvasó a tanulmányozását, és hasznosítani tudja az újonnan szerzett ismereteket.

Minden jót!  
Andrew Troelsen

# 1. rész

## Bevezetés a C#-ba és a .NET platformba



## ELSŐ FEJEZET

# A .NET filozófiája

Ma már a programozók számára elengedhetetlen, hogy a néhány évente vagy akár még gyakrabban továbbképzéseken vegyenek részt, hiszen mindig naprakésznek kell lenniük a legújabb technológiákból. A szoftverfejlesztés gyöngyszemeinek kikiáltott nyelveket (C++, Visual Basic 6.0, Java), keretrendszereket (OWL, MFC, ATL, STL), architektúrákat (COM, COBRA, EJB) és API-kat (mint a .NET Windows-űrlapjai és a GDI+ könyvtárai) végül mindig háttérbe szorítja valami jobb, de legalábbis valami új. Bár a belső adatbázisunk frissítésekor nyilván csalódottságot érzünk, ám mindez elkerülhetetlen. Éppen ezért könyvünknek az a legfontosabb célja, hogy betekintsünk a Microsoft szoftverfejlesztéssel kapcsolatos aktuális kínálatának részleteibe: a .NET platformba és a C# programozási nyelvbe.

Ebben a fejezetben elsődleges feladatunk az, hogy tisztázzuk a könyv további részeinek megértéséhez szükséges alapfogalmakat. Köztük számos .NET-hez kapcsolódó téma: például a szerelvények, a köztes nyelvi kód (CIL) és a JIT-fordítás tárgyalását is nyomon követhetjük. A C# programozási nyelv legfontosabb sajátosságainak ismertetése mellett a .NET-keretrendszer különböző megjelenési formái – mint a nyelvek közös futtatórendszere (CLR), a közös típusrendszer (CTS) és a közös nyelvi specifikáció (CLS) – között létrejövő kapcsolatra is fény derül.

Ezeket kívül az olykor BCL-nek, esetleg FCL-nek (keretrendszer-osztálykönyvtár) rövidített .NET-alaposztálykönyvtárak funkcionalitását is megvizsgáljuk. Végezetül áttekintjük a .NET platform nyelvagnosztikus és platformfüggetlen természetét is. (Tudniillik a .NET nem korlátozódik a Windows operációs rendszerre.) Ezeket a témákat a továbbiakban részletesen is kifejthetjük.

## Az előzmények áttekintése

A .NET univerzum sajátosságainak számbavétele előtt hasznos lehet néhány olyan tényezőt összegezni, amely a Microsoft jelenlegi platformjának megalakítását előidézte. A tisztánlátáshoz kezdjük a fejezetet egy rövid, „fájdalommentes” történelemórával, hogy felelevenítsük az előzményeket, és megértjük azok korlátait. (Elvégre az első lépés a megoldáshoz vezető úton, ha beismerjük, hogy problémával állunk szemben.) Miután tettünk egy gyors körutazást a múltban, figyelmünket a C# és a .NET-platform számtalan előnyének a megismerésére fordíthatjuk.

### Egy C/Win32 API programozó élete

Azt szokták mondani, hogy a Windows operációs rendszerek családjába tartozó szoftverek fejlesztése a C programozási nyelv és a Windows alkalmazásprogramozói felületének (API) együttes használatából áll. Igaz, hogy számos alkalmazást sikeresen létrehoztak ezzel a hagyományos módszerrel, néhányan mégis vitatják, hogy az alkalmazások nyers API-val történő felépítése összetett feladat lenne.

Az első nyilvánvaló probléma az, hogy a C nagyon tömör nyelv. A C# fejlesztői kénytelenek kézi memóriakezeléssel, csúnya pointeraritmetikával és ugyancsak csúnya szintaktikai konstrukciókkal megküzdenni. Ráadásul, mivel a C strukturált nyelv, hiányzik belőle az objektumorientált megoldások előnye (valaki el tudja mondani, mi az a *spagettikód*?). Amikor a Win32 API segítségével definiált globális függvények és adattípusok ezreit hozzáadjuk egy eleve túl részletes nyelvhez, nem csoda, hogy olyan sok hibás alkalmazást kapunk.

### Egy C++/MFC-programozó élete

A csupasz C/API fejlesztésének egyik nagy horderejű újítása a C++ programozási nyelv használata volt. A C++ sok szempontból a C-re helyezett objektumorientált *rétegnek* tekinthető. Így annak ellenére, hogy a C++ programozóinak hasznára vannak az objektumorientált programozás közismert alappillérei (egységbezárás, származtatás és polimorfizmus), továbbra is ki vannak szolgáltatva a C nyelv nehézségeinek (pl. kézi memóriakezelés, csúnya pointeraritmetika és csúnya szintaktikai konstrukciók).

Összetettsége ellenére, manapság is sok C++-keretrendszer létezik. Például a Microsoft Foundation Classes (MFC) olyan C++-osztály-készletet biztosítanak a fejlesztők számára, amely megkönnyíti a Win32-alkalmazások felépítését. Az MFC legfőbb szerepe, hogy számos osztály, makró és kódgeneráló eszköz (varázsló) mögött a natív Win32 API egy részét beburkolja. Az MFC-keretrendszer hasznos támogatását figyelmen kívül hagyjuk (ugyanúgy, mint sok egyéb C++-alapú eszközrendszert), be kell látnunk, hogy a C++-programozás nehézkes, és több hibalehetőséget rejt magában, tekintve, hogy a C-ből ered.

## Egy Visual Basic 6.0 programozó élete

Az egyszerűsége törekvés jegyében sok programozó a C(++)-alapú keretrendszerekről olyan barátságosabb és kifinomultabb nyelvekre tért át, mint például a Visual Basic 6.0 (VB6). A VB6 annak köszönheti népszerűségét, hogy egyszerűen lehet vele összetett felhasználói felületeket, kódkönyvtárakat (pl. COM-kiszolgálók) és adatelérési logikát készíteni. A VB6 sokkal jobban elrejt szem elől a csupasz Win32 API bonyolultságát, mint az MFC, számos beépített forráskód-varázsló, valódi adattípusok, osztályok és VB-specifikus függvények segítségével.

A VB6 legnagyobb hibája (amelyet a .NET-platform megjelenése orvosolt), hogy nem teljesen objektumorientált nyelv, hanem inkább „objektumtudatos”. A VB6 például nem engedi a programozónak, hogy „ez egy...” („is-a”) kapcsolatokat hozzanak létre a típusok között (azaz nem engedi a klasszikus származtatást), és nincs belső támogatása paraméterezett osztályok készítéséhez. Ezenkívül a VB6 nem képes többszálú alkalmazások létrehozására sem, kivéve, ha alacsony szintű Win32 API hívásokat szeretnénk (ám ez legjobb esetben összetett, legrosszabb esetben pedig veszélyes).

## Egy Java/J2EE-programozó élete

A Java megérkezett. A Java objektumorientált programozási nyelv, amelynek szintaktikai gyökerei a C++-ból erednek. Amint azzal sokan tisztában vannak, a Java előnyei messze túlszárnyalják platformfüggetlenségi támogatását. A Java (mint nyelv) a C++ számos kellemetlen szintaktikai jellemvonását kiküszöbölte. A Java (mint platform) a programozók számára számtalan előre megadott „csomagot” biztosít, amelyek különféle típusdefiníciókat tartal-

maznak. Ezekkel a típusokkal a Java-programozók képesek 100%-ban tiszta Java-alkalmazásokat létrehozni, amelyek teljes adatbázis-összekapcsolhatósággal, üzenetközvetítési támogatással, webes ügyféloldallal és gazdag asztali felhasználói felülettel rendelkeznek.

Habár a Java nagyon elegáns nyelv, potenciális problémája, hogy használatakor jellemzően előlről hátrafelé kell haladni a fejlesztés folyamán. A Java tulajdonképpen kismértékben nyújt nyelvi integritást, minthogy ez ellentmond elsődleges feladatának – a Java egy egyszerű, bármilyen célra felhasználható programozási nyelv. A valóságban azonban forráskódok milliányi sora létezik, s ezeket előnyös lenne Java-kódból felhasználni. Sajnos a probléma megoldását a Java nem könnyíti meg. Amíg ugyanis korlátozott képességet mutat a nem Java-s API-k eléréséhez, addig csak kismértékben támogatja a valódi nyelvközi integrációt.

## Egy COM-fejlesztő élete

A Component Object Model (COM – komponensobjektum-modell) volt a Microsoft előző alkalmazásfejlesztési keretrendszere. A COM architektúrája lényegében azt fejezi ki, hogy „[h]a az osztályainkat a COM szabályai szerint építjük fel, *újrafelhasználható bináris kódblokkot* kapunk”.

Egy bináris COM-kiszolgálóban az a nagyszerű, hogy nyelvfüggetlen módon hozzá lehet férni. Így a C++-programozók olyan COM-osztályokat hozhatnak létre, amelyeket a VB6 felhasználhat. A Delphi-programozók használhatnak olyan COM-osztályokat, amelyeket C segítségével hoztak létre, és így tovább. A COM nyelvfüggetlensége azonban némileg korlátozott. Egy meglévő COM-osztályból például nem lehet újabbat származtatni (mivel a COM nem támogatja a klasszikus származtatást). Ehelyett, a sokkal nehezebb „van neki egy...” („has-a”) kapcsolat segítségével kell újra felhasználni a COM-osztály típusait.

A COM másik előnye az, hogy transzparens tárhellyel rendelkezik. Az olyan szerkezetek használatával, mint az alkalmazásazonosító (AppIDs), a csonk, a proxyk és a COM futási környezet, a programozóknak nem kell socketekkel, távoli eljáráshívásokkal és egyéb alacsony szintű részletekkel dolgozniuk. Nézzük meg például a következő VB6-COM-ügyfélkódot:



```
' A MyCOMClass típus bármely  
' COM-alapú nyelven megírható, és a hálózaton bárhol  
' elhelyezhető (helyi számítógépen is).  
Dim obj as MyCOMClass  
Set obj = New MyCOMClass ' AppID használatának a helye.  
obj.DoSomeWork
```

Habár a COM-ot nagyon sikeres objektummodellnek is tarthatjuk, valójában elképesztően bonyolult (legalábbis addig, amíg nem fordítunk több hónapot az átkötések vizsgálatára – elsősorban C++-programozóként). A COM-binárisok fejlesztésének egyszerűsítésére számos COM-alapú keretrendszer létezik. Ilyen például az ATL (Active Template Library), amely olyan C++-osztályok, -sablonok és -makrók készletével szolgál, amelyek megkönnyítik a COM-típusok létrehozását.

Sok egyéb nyelv ugyancsak elrejt szem elől a COM-infrastruktúra nagy részét. A nyelvi támogatás azonban önmagában nem elég a COM komplexitásának elrejtéséhez. Még ha egy olyan viszonylag egyszerű COM-alapú nyelvet választunk, mint a VB6, akkor is meg kell küzdenünk a sérülékeny regisztrációs bejegyzésekkel és számos telepítéssel kapcsolatos problémával (gyűjtőnéven, némileg komikusan, *DLL-pokol*).

## Egy Windows DNA programozó élete

Tovább bonyolítja a helyzetet az internet. Az elmúlt néhány évben a Microsoft jó néhány további internetalapú szolgáltatást épített be az operációs-rendszer-családjába és termékeibe. Sajnos a COM-alapú Windows DNA segítségével történő webes alkalmazások létrehozása ugyancsak meglehetősen bonyolult.

Ez a bonyolultság részben annak a ténynek köszönhető, hogy a Windows DNA számos technológia és nyelv használatát igényli (ASP, HTML, XML, JavaScript, VBScript és COM[+], valamint olyan adatelérési API-t, mint az ADO). Problémát jelent, hogy szintaktikai szempontból a technológiák nagy része nem kapcsolódik egymáshoz. A JavaScript szintaxisa például sokban hasonlít a C-re, míg a VBScript a VB6 részhalmaza. A COM+-futtatókörnyezetre tervezett COM-kiszolgálók teljesen eltérő megjelenésűek, mint azok az ASP-oldalak, amelyek hívják őket. Mindez a technológiák teljesen zavaros keverékét eredményezi.

Ezenkívül, és ez talán sokkal fontosabb, minden nyelvnek és/vagy technológiának megvan a saját típusrendszere (és ez lehet, hogy semmiben sem hasonlít egy másikéra). Túl azon, hogy minden egyes API rendelkezik saját előre gyártott kódgyűjteménnyel, az alapvető adattípusokat nem lehet mindig ugyanúgy kezelni. A `COMBSTR` az ATL-ben nem teljesen ugyanaz, mint egy `string` a VB6-ban; egyiknek sincs semmi köze a `C char*` adattípusához.

## A .NET-megoldás

Az előző rövid történelemóra tanulsága az volt, hogy a Windows-programozók élete bizony nem egyszerű. A .NET-keretrendszer azonban radikálisan egyszerűsíti és megkönnyíti a programozó életét. A .NET a „Változtassunk meg mindent!” megoldást kínálja („elnézést kérek, de ne az üzenet közvetítőjét okoljuk az üzenetért”). Ahogy a könyv további részeiben látni fogjuk, a .NET-keretrendszer a rendszerfelépítés teljesen új modellje a Windows operációs rendszercsaládban éppúgy, mint számos nem Microsoft-alapú operációs rendszerben (mint például a Mac OS X és a különféle Unix-/Linux-termékek). Előzetesen íme a .NET néhány alapvető jellemzőjének összefoglalása:

- *Együttműködési képesség a meglévő forráskóddal.* Ez (természetesen) nagyon hasznos. A meglévő COM-binárisok keverednek (azaz együttműködnek) az újabb .NET bináris fájlokkal, és viszont. A `PInvoke` lehetővé teszi a C-alapú könyvtárak (beleértve az operációs rendszer API) hívását a .NET-kódból.
- *Teljes nyelvi integráció.* A .NET támogatja a nyelvközi származtatást, a nyelvközi kivételkezelést és a nyelvközi hibakeresést.
- *Közös futtatórendszer az összes .NET-alapú nyelv számára.* Ez a rendszer definiál egy olyan típuskészletet, amelyet minden .NET-alapú nyelv megért.
- *Átfogó alaposztálykönyvtár.* Ez a könyvtár teszi lehetővé a natív API-hívások komplexitásának elkerülését, és olyan konzisztens objektummodellt kínál, amelyet az összes .NET-alapú nyelv használ.

- *Nincs több COM-átkötés.* Az `IClassFactory`-nak, az `IUnknown`-nak, az `IDispatch`-nek, az IDL-kód és a nem túl szimpatikus `variant`-kompatibilis adattípusoknak (`BSTR`, `SAFEARRAY` stb.) nincs helyük egyetlen .NET-binárisban sem.
- *Egy valóban leegyszerűsített telepítési modell.* .NET alatt nem szükséges bejegyezni a bináris egységeket a rendszerleíró adatbázisba. Ráadásul a .NET lehetővé teszi, hogy egyetlen gépen ugyanazon .dll több verziója legyenek jelen.

Ahogy a fenti felsorolásból látszik, a .NET-platformnak semmi köze a COM-hoz (azonfelül, hogy mindkét keretrendszer a Microsofttól származik). A .NET- és a COM-típusok valójában csak az együttműködési réteg segítségével tudnak kommunikálni.

---

**Megjegyzés** A .NET együttműködési réteg leírása a 2 kötet A függelékében található.

---

## Bevezetés a .NET platform építőelemeibe (CLR, CTS és CLS)

A .NET néhány előnyét már megismerve nézzük át azt a három alapvető, egymással összefüggő egységet, amelyek mindezt lehetővé teszik: a CLR-t, a CTS-t és a CLS-t. Egy programozó szemszögéből a .NET felfogható futtató-környezetként és átfogó alapsztálykönyvtárként. A futtató réteg más néven *közös nyelvi futtatórendszer*, vagyis a CLR. A CLR elsődleges feladata, hogy megkeresse, betöltse és kezelje helyettünk a .NET-típusokat. Ezenkívül olyan alacsony szintű részletekre is figyel, mint a memóriakezelés, alkalmazástartományok létrehozása, végrehajtási szálak és objektumkontextus-határok, valamint különböző biztonsági ellenőrzések végrehajtása.

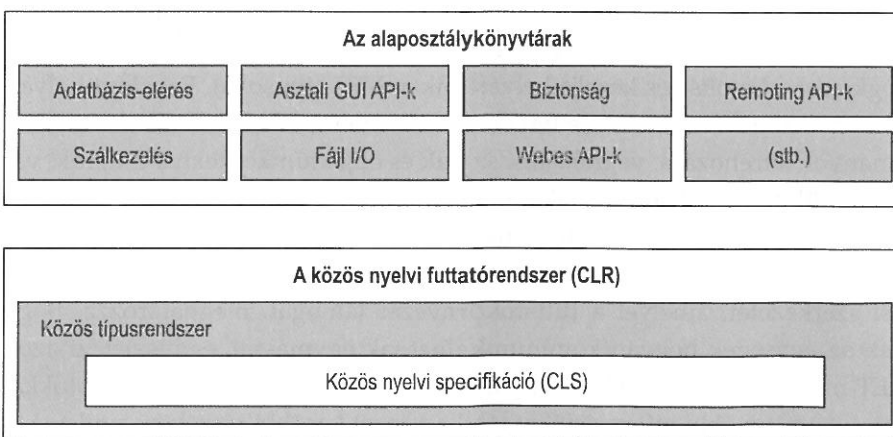
A .NET-platform másik építőeleme a *közös típusrendszer*, azaz a CTS. A CTS-specifikáció teljes körűen leírja az összes lehetséges adattípust és programozási szerkezetet, amelyet a futtatókörnyezet támogat, meghatározza, hogy ezek az egységek hogyan kommunikálhatnak egymással, és részletezi azok .NET-metaadat-formátumban való megjelenésének módját (a metaadatokkal kapcsolatos további információkat lásd a fejezet későbbi részében; a teljes körű leírást pedig lásd a 16. fejezetben).

Egy adott .NET-alapú nyelv nem szükségszerűen támogat minden egyes CTS-ben definiált funkciót. A *közös nyelvi specifikáció* (CLS) egy olyan kapcsolódó specifikáció, amely definiálja azoknak a közös típusoknak és programozási szerkezeteknek a részhalmazát, amelyeket minden .NET programozási nyelv elfogad. Ezért, ha olyan .NET-típusokat építünk fel, amelyek csak CLS-kompatibilis funkciókkal rendelkeznek, nyugodtak lehetünk afelől, hogy az összes .NET-alapú nyelv használni tudja azokat. Ellenben, ha olyan adattípust vagy programozási szerkezetet használunk, amely a CLS határain kívül esik, nem biztos, hogy az összes .NET programozási nyelv képes lesz kommunikálni a .NET-kódkönyvtárunkkal.

## Az alapsztálykönyvtárak szerepe

A CLR- és CTS/CLS-specifikációkon kívül, a .NET-platform az összes .NET programozási nyelv számára biztosít egy alapsztálykönyvtárat. Ez az alapsztálykönyvtár nemcsak beágyaz különböző primitíveket, mint a végrehajtási szálak, fájl input/output (I/O), grafikus renderelés és különféle külső hardvereszközökkel történő kommunikáció, de számos, leginkább az élet-szerű alkalmazások igényeihez mért szolgáltatást is támogat.

Az alapsztálykönyvtárak például olyan típusokat definiálnak, amelyek megkönnyítik az adatbázis-hozzáférést, az XML-dokumentumok manipulációját, a programozott adatvédelmet és a webes (csakúgy, mint a hagyományos asztali és konzolos) front endek létrehozását. A CLR, CTS, CLS és az alapsztálykönyvtár közötti kapcsolatot, az 1.1. ábra szemlélteti.



1.1. ábra: A CLR, CTS, CLS és az alapsztálykönyvtár kapcsolata

## A C# előnyei

Mivel a .NET radikálisan eltér a korábbi technológiáktól, a Microsoft kifejezetten ehhez az új platformhoz hozott létre egy új programozási nyelvet, a C#-t (kiejtve: [si: sa:p]). A C# alapszintaxisa *nagyon* hasonlít a Java szintaxisára. A C# azonban nem a Java egyik módosulata. Mind a C#, mind a Java a C programozási nyelv családjába (C, Objective C, C++ stb.) tartozik, ezért hasonló szintaxissal rendelkeznek. Ahogy a Java sok tekintetben a C++ letisztult verziója, a C#-ot ugyanígy a Java letisztult verziójának tekinthetjük.

Ettől függetlenül az igazság az, hogy a C# szintaktikai felépítése nagyrészt a Visual Basic 6.0 és a C++ alapján készült. Ahogy például a VB6, a C# is támogatja a tulajdonságok létrehozását (ellentétben a tradicionális getter és setter metódusokkal) és a metódusok deklarálásának képességét változó számú argumentummal (argumentumtömbök révén). A C++-hoz hasonlóan, a C# lehetővé teszi az operátorok túlterhelését, valamint struktúrák, felsorolt típusok és visszahívható függvények létrehozását (metódusreferenciák révén).

---

**Megjegyzés** Ahogy a 13. fejezetben látni fogjuk, a C# 2008 számos olyan konstrukciót átvett, amelyek eredetileg különböző funkcionális nyelvekből (pl. LISP vagy Haskell) származnak. Ráadásul, a LINQ megjelenésével (további információkért lásd a 14. és a II. kötetben található 24. fejezetet) a C# több olyan programozási konstrukciót támogat, amelyek meglehetősen egyedivé teszik a programnyelvek palettáján. Mindamellettt probléma a C#-pal, hogy valóban hatással voltak rá a C-alapú nyelvek.

---

Mivel a C# több nyelv keresztezéséből született, szintaktikailag legalább annyira, ha nem még inkább letisztultnak tekinthető, mint a Java, olyan egyszerű, mint a VB6, és annyira hatékony és rugalmas, mint a C++ (a vele járó apró csúnyaságok nélkül). Íme, egy lista a C# alapvető tulajdonságairól, amelyek a nyelv valamennyi verzióját jellemzik:

- Nincs szükség mutatókra. A C#-programoknak jellemzően nincs szükségük közvetlen mutatómanipulálásra (de, ha mindenképpen szükséges, le lehet erre a szintre menni).
- Automatikus memóriakezelés a szemétygyűjtésen keresztül. Így a C# nem támogatja a `delete` kulcsszót.
- Formális szintaktikai konstrukcióként megjelenik az osztály, az interfész, a struktúra, a felsorolás és a metódusreferenciák.

- A C++-hoz hasonló operátor-túlterhelés az egyedi típusok esetében, csak egyszerűbben (pl. a meggyőződés arról, hogy a „return \*this to allow chaining” nem a mi problémánk).
- Az attribútumalapú programozás támogatása. Az ilyen típusú fejlesztés lehetővé teszi, hogy a típusokat és a tagjaikat magyarázatokkal lásuk el viselkedésük további leírásához.

A .NET 2.0 verziójával (kb. 2005-től) a C# programozási nyelv számos újdonsággal frissült, amelyek közül az alábbiak a legfontosabbak:

- Generikus típusok és tagok felépítésének a képessége. A generikus típusok használatával rendkívül hatékony és típusbiztos forráskódokat építhetünk fel, amelyek a generikus elemmel való kommunikáció során meghatározott helyőrzőket definiálnak.
- Névtelen metódusok támogatása, amelyek lehetővé teszik, hogy bárhol, ahol metódusreferenciára van szükség, inline függvényeket alkalmazzunk.
- A metódusreferencia/esemény modell számtalan egyszerűsítése, köztük kovariancia, kontravariancia és metóduscsoport-átalakítás.
- Egyetlen típus több kódfájlon keresztüli definiálásának képessége (vagy ha szükséges, memórián belüli ábrázolása) a `partial` kulcsszó segítségével.

A .NET 3.5 egyértelműen még több funkcionalitással látta el a C# programozási nyelvet (pontosabban a C# 2008 verziót), köztük az alábbiakkal:

- A különféle adatformátumokkal való kommunikációhoz szükséges erősen típusos lekérdezések (LINQ) támogatása.
- Névtelen típusok támogatása, amellyel sokkal inkább a típusok „alakját”, mint viselkedését modellezhetjük.
- Létező típusok funkcionalitásának kiterjesztése, bővítő metódusok segítségével.
- A lambda operátor (`=>`) bevezetése, amely még inkább leegyszerűsíti a .NET-metódusreferencia-típussal végzett munkát.
- Új objektuminicializáló szintaxis, amely lehetővé teszi a tulajdonságértékek beállítását az objektum létrehozásakor.

A C# nyelvvel kapcsolatban talán az a legfontosabb, hogy csak olyan forráskódot tud létrehozni, amely .NET-futtatókörnyezetben működik (C# segítségével nem tudunk natív COM-kiszolgálót vagy nem felügyelt Win32 API alkalmazást felépíteni). A .NET-futtatókörnyezethez készült forráskód leírására hivatalosan a *felügyelt programkód* kifejezést használjuk. A felügyelt programkódot tartalmazó bináris egységet *szerelvénynek* nevezzük. (A szerelvényekkel kapcsolatos további információkért lásd a „.NET-szerelvények áttekintése” című részt alább.) Azt a forráskódot ellenben, amelyiket a .NET-futtatókörnyezet közvetlenül nem tudja hosztolni, *nem felügyelt programkódnak* nevezzük.

## További .NET-alapú programozási nyelvek

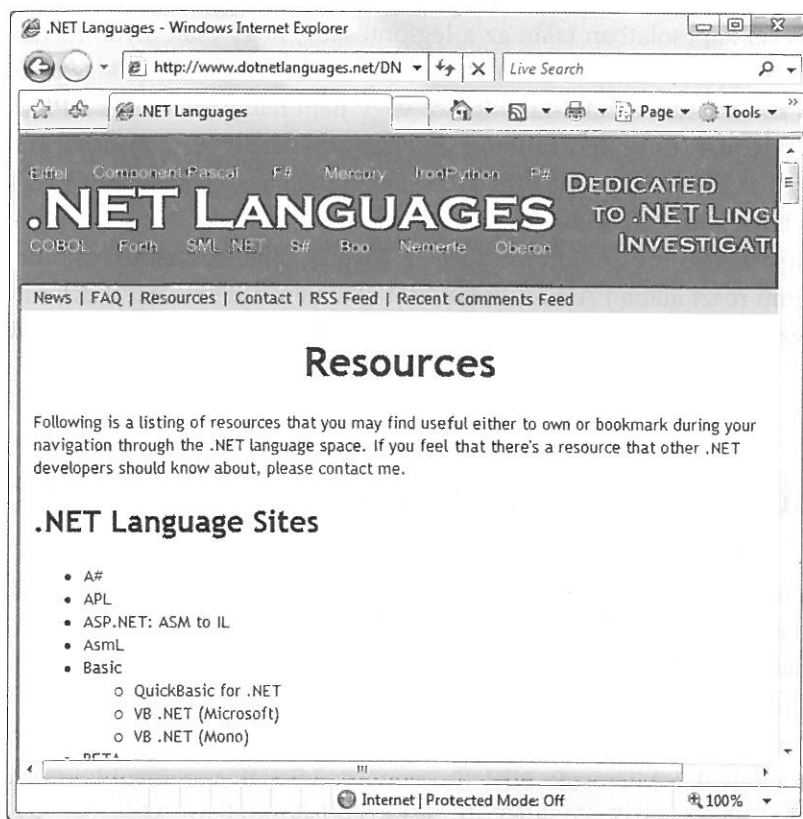
.NET-alkalmazásokat nem csak a C# nyelv segítségével építhetünk fel. Amikor 2000-ben, a Microsoft fejlesztői konferenciáján először bemutatták a nagyközönségnek a .NET-platformot, több gyártó is bejelentette, hogy felépítik a saját fordítóprogramjuk .NET-alapú verzióját.

Azóta nyelvek tucatjai estek át a .NET-felvilágosodáson. Azon az öt nyelven kívül, amelyek a Microsoft .NET Framework 3.5 SDK verzióval kerültek forgalomba (C#, Visual Basic .NET, J#, C++/CLI [korábban „Managed Extensions for C++”] és JScript .NET), léteznek .NET-fordítóprogramok Smalltalkra, COBOL-ra és Pascalra is. Jóllehet könyvünk (szinte) kizárólag csak a C#-pal foglalkozik, érdemes ellátogatni a következő webhelyre (a Microsoft fenntartja az URL változtatásának jogát):

<http://www.dotnetlanguages.net>.

Ha a főoldal tetején lévő Resources hivatkozásra kattintunk, számtalan .NET programozási nyelvet és azokhoz kapcsolódó hivatkozásokat tartalmazó listát kapunk, ezekre kattintva pedig különféle fordítókat tölthetünk le (lásd 1.2. ábra).

Ha elsősorban a C# szintaxisa alapján szeretnénk .NET-programokat létrehozni, tanácsos felkeresni ezt a webhelyet, itt biztosan találunk ugyanis a célnak megfelelő .NET-nyelvet (esetleg a LISP .NET-et?).



1.2. ábra: A *www.DotNetLanguages.net* az egyik olyan webhely, amely ismert .NET programozási nyelveket dokumentál

## Élet egy többnyelvű világban

Amikor a fejlesztők ráébredtek a .NET nyelvagnosztikai természetére, számos kérdés merült fel. Talán az egyik leggyakoribb kérdés ez lehetett: „Ha az összes .NET-nyelv felügyelt programkódot fordít, miért van szükség több fordítóra?”. Ezt a kérdést többféleképpen meg lehet válaszolni. Először is, mi programozók *nagyon* válogatósak vagyunk, ha bármilyen programozási nyelv kiválasztásáról van szó. Néhányan a pontosvesszős és kapcsos zárójeles nyelveket szeretik, a lehető legkevesebb nyelvi kulcsszóval. Mások azokat a nyelveket szeretik, amelyek az ember számára olvashatóbb szintaktikai tokeneket tartalmaznak (mint a Visual Basic). Megint mások pedig a nagyszámítógépes tapasztalataikat szeretnék érvényesíteni, amíg áttérnek a .NET-platformra (a COBOL .NET-en keresztül).



Legyünk őszinték. Ha a Microsoft létrehozott volna egy „hivatalos” .NET-nyelvet, amely a BASIC nyelvcsaládból származik, vajon az összes programozó elégedett volna? Vagy, ha az egyetlen „hivatalos” .NET-nyelv a Fortran szintaxisán alapulna, nyilván sokan lennének, akik a .NET-et végképp semmibe vennék. Mivel a .NET-futtatókörnyezetben nem fontos, hogy milyen nyelv segítségével építették fel a felügyelt kódot, a .NET-programozók hűek maradhatnak ahhoz a szintaxishoz, amelyet korábban választottak, és a lefordított szerelvényeket probléma nélkül megoszthatják a csapattagokkal, a részlegekkel és a külső szervezetekkel (tekintet nélkül arra, hogy a másik fél milyen .NET-nyelvet használ).

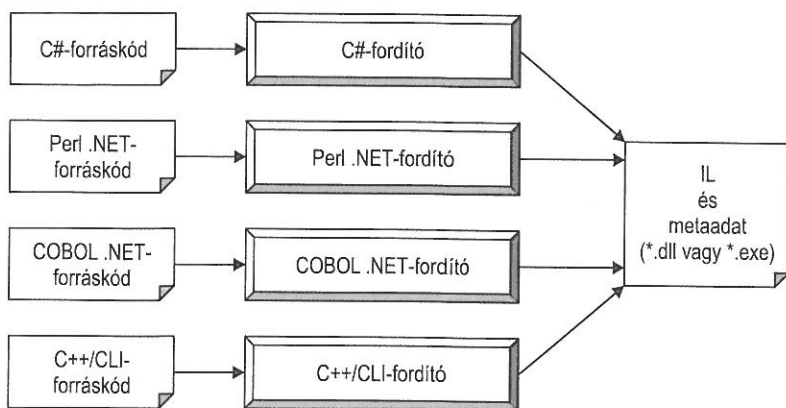
A különféle .NET-nyelveknek egyetlen, egységes szoftvermegoldásba történő integrálásának másik előnye, hogy minden egyes programozási nyelvnek megvannak a maga erősségei és gyengeségei. Egyes programozási nyelvek például remekül támogatják a fejlett matematikai feldolgozást. Mások pedig elsősorban a pénzügyi számításokat, a logikai számításokat, a nagy-számítógépekkel való kommunikációt stb. Ha az egyik programozási nyelv erősségeit vesszük, majd azokat egyesítjük a .NET-platform előnyeivel, voltaképpen mindenki jól jár.

A valóságban persze elég nagy az esély arra, hogy a szoftverek felépítését a kiválasztott .NET-nyelv segítségével fogjuk végezni. Ha azonban egy .NET-nyelv szintaxisát megismerjük, nagyon könnyű lesz egy másikét is megtanulni. Ez szintén meglehetősen hasznosnak tűnik, különösen a szoftvertanácsadóknak. Ha a választott nyelvünk például a C#, de egy olyan ügyfélnél dolgozunk, ahol csak Visual Basic .NET-et használhatunk, a .NET-keretrendszer funkcionalitását így is kihasználhatjuk, és az alapkód általános szerkezetét nagyobb nehézségek nélkül megérthetjük.

## A .NET-szerelvények áttekintése

Függetlenül attól, melyik .NET-nyelven programozunk, fogadjuk el, hogy noha a .NET bináris fájlok kiterjesztései megegyeznek a COM-kiszolgálók és a nem felügyelt Win32 bináris fájlok kiterjesztéseivel (\*.dll vagy \*.exe), nincs bennük semmiféle belső hasonlóság. A \*.dll .NET bináris fájlok például nem exportálnak metódusokat a COM-futtatókörnyezettel való kommunikációhoz (mivel a .NET *nem* COM). Továbbá a .NET bináris fájlok nem használják COM-típuskönyvtárakat, és nincsenek bejegyezve a rendszerleíró adatbázisba.

De talán a legfontosabb, hogy a .NET bináris fájlok platformfüggő utasítások helyett platformfüggetlen *köztes nyelvet (IL)* és típusmetaadatokat tartalmaznak. Az 1.3. ábra az eddigieket mutatja be.



1.3. ábra: Az összes .NET-alapú fordító IL-utasításokat és metaadatokat ad ki

**Megjegyzés** Az „IL” rövidítést illetően meg kell említeni, hogy a .NET kifejlesztése során az IL helyett hivatalosan a Microsoft köztes nyelv (MSIL) kifejezést használták. A .NET végleges kiadásakor azonban már közös köztes nyelvként (CIL) említették. Ebből következően a .NET-irodalomban az IL, az MSIL és a CIL tehát pontosan ugyanarra utal. Az aktuális terminológiát követve a továbbiakban a „CIL” rövidítést fogjuk alkalmazni.

Amikor .NET-alapú fordító segítségével hozunk létre egy \*.dll vagy egy \*.exe kiterjesztésű fájlt, az eredményül kapott modul *szerelvényt* alkot. (A .NET-szerelvényeket részletesen a 15. fejezetben tárgyaljuk.) A .NET futtató környezet részletes tárgyalása előtt meg kell ismerkednünk az új fájlformátum néhány alapvető tulajdonságával.

A szerelvények tehát köztes nyelvi kódokat tartalmaznak, amelyek elvben a Java-bájt kódhoz hasonlítanak, hiszen addig nem változnak platformfüggő gépi utasítássá, amíg ez nem feltétlenül szükséges. A „feltétlenül szükséges” pont jellemzően az, amikor a .NET-futtatókörnyezet a CIL-utasításblokk (például egy metódusimplementáció) használatára hivatkozik.

A CIL-utasításokon kívül a szerelvények *metaadatokat* is tartalmaznak, amelyek a binárisban található típusok karakterisztikáját részletezik. Ha van például egy *sportscar* nevű osztályunk, a típusmetaadat olyan részleteket ír le, mint a *sportsCar* ösosztálya, amelynek interfészeit (ha léteznek) a *sportscar* implementálja, valamint az összes olyan tagot, amelyeket a *sportscar* típusa támogat.

A .NET-metaadatok hatalmas előrelépést jelentenek a COM-metaadatokhoz képest. Talán már közismert, hogy a COM-binárisok jellemzően hozzárendelt típuskönyvtárakat használnak (ez valamivel több, mint az interfészleíró nyelv [IDL] forráskódjának bináris verziója). A COM-típusinformációkkal az a probléma, hogy nem biztos, hogy megvannak, illetve, hogy az IDL-kód nem tudja dokumentálni a kívülről hivatkozott kiszolgálókat, amelyek az aktuális COM-kiszolgáló megfelelő működéséhez szükségesek. Ezzel szemben a .NET-metaadatok mindig jelen vannak, és az adott .NET-alapú fordító automatikusan generálja őket.

Végezetül, a CIL és a típusmetaadatokon kívül maguk a szerelvények is használnak metaadatot, amelyet *manifesztumnak* nevezünk. A manifesztum a szerelvény aktuális változatáról szóló és kulturális információkat tartalmaz (a sztring- és a képerőforrások lokalizációjához), illetve egy listát, amely a megfelelő végrehajtáshoz szükséges összes külsőleg hivatkozott szerelvényt felsorolja. A következő néhány fejezetben a szerelvények típusainak, metaadatainak és önleíró adatainak vizsgálatához szükséges különféle eszközöket ismerhetjük meg.

## Egyfájlos és többfájlos szerelvények

Az esetek nagy többségében egyszerű egy-egy kapcsolat van a .NET-szerelvény és a bináris fájl (\*.dll vagy \*.exe) között. Ebből következően, ha egy .NET \*.dll létrehozásán dolgozunk, úgy vehetjük, hogy a bináris és a szerelvény egy és ugyanaz. Hasonlóképpen, ha egy végrehajtható asztali alkalmazást készítünk, az \*.exe fájlt egyszerűen tekinthetjük magának a szerelvénynek. (Később azonban, a 15. fejezetben majd kiderül, hogy ez így nem pontos.) Szaknyelven, ha a szerelvény egy \*.dll vagy egy \*.exe modulból áll, azt *egyfájlos szerelvénynek* nevezzük. Az egyfájlos szerelvények az összes szükséges CIL-t, metaadatot és hozzárendelt manifesztumot önálló, egyedi és pontosan definiált csomagban tartalmazzák.

A *többfájlos szerelvények* viszont számos .NET bináris fájlból állnak, amelyeket *moduloknak* nevezünk. A többfájlos szerelvény esetében az egyik modul (*elsődleges modul*) tartalmazza a szerelvény manifesztumát (esetleg CIL-utasításokat és különféle típusok metaadatait). A többi modul egy modulszintű manifesztummalsal, CIL-utasításokkal és típusmetaadatokkal rendelkezik. Nyilvánvaló, hogy az elsődleges modul dokumentálja a szükséges másodlagos modulok csoportját a szerelvény manifesztumában.

Miért hozunk létre többfájlos szerelvényeket? Ha egy szerelvényt különálló modulokra osztunk fel, jóval rugalmasabb telepítési beállításokhoz jutunk. Ha például a felhasználó egy olyan távoli szerelvényre hivatkozik, amelyet szeretne letölteni a számítógépére, a futtatókörnyezet csak a szükséges modulokat fogja letölteni. Ezért felépíthetjük úgy is a szerelvényünket, hogy azokat a típusokat, amelyekre ritkábban van szükség (mint például a `HardDriveRef` nevű osztály), külön modul tárolja.

Ellenben, ha az összes típust egyfájlos szerelvénybe helyezzük, a végfelhasználó jó néhány felesleges adatot fog letölteni (és ez nyilvánvalóan időpocsékolás). Egy szerelvény tehát valóban egy vagy több kapcsolódó modul *logikai csoportosítását* tartalmazza, amelyeket önálló egységként telepítünk és módosítunk.

## A közös köztes nyelv (CIL) szerepe

A következőkben vizsgáljuk meg kicsit közelebbről a köztes nyelvi kódot, a típusmetaadatot és a szerelvény-manifesztumot. A CIL olyan nyelv, amely bármilyen platformfüggő gépiutasítás-készlet felett áll. A következő C#-kód például egy hétköznapi számológépet modellez. Ne foglalkozzunk most a pontos szintaxissal, de figyeljünk a `Calc` osztály `Add()` metódusának formátumára:

```
// Calc.cs
using System;

namespace CalculatorExample
{
    // Az osztály tartalmazza az alkalmazás belépési pontját.
    class Program
    {
        static void Main()
        {
            Calc c = new Calc();
            int ans = c.Add(10, 84);
            Console.WriteLine("10 + 84 is {0}.", ans);

            // Várakozás, amíg a felhasználó megnyomja az Enter
            // billentyűt.
            Console.ReadLine();
        }
    }
}
```

```
// C#-számológép.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
```

Ha ezt a kódfájlt a C#-fordítóval (csc.exe) lefordítjuk, olyan egyfájlos \*.exe szerelvényt kapunk, amely tartalmaz manifesztumot, CIL-utasításokat és metaadatokat, és ezek minden szempontból leírják a Calc és a Program osztályt.

---

**Megjegyzés** A C#-fordító segítségével történő kódfordítást, illetve a grafikus IDE-k, például a Visual Studio, a Visual C# Express és a SharpDevelop, használatát a 2. fejezetben vizsgáljuk meg részletesen.

---

Ha ezt a szerelvényt például az ildasm.exe segítségével szeretnénk megnyitni (lásd alább), láthatjuk, hogy az Add() metódus köztes nyelven, a következőképpen jelenik meg:

```
.method public hidebysig instance int32 Add(int32 x,
int32 y) cil managed
{
    // A kód mérete 9 (0x9)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: nop
    IL_0001: ldarg.1
    IL_0002: ldarg.2
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007
    IL_0007: ldloc.0
    IL_0008: ret
} // a Calc::Add metódus vége
```

(Az eredményül kapott metódust a 19. fejezetben találjuk a CIL programozási nyelv alapjainak tárgyalásáról szóló részben.) A hangsúly most azon van, hogy a C#-fordító nem platformfüggetlen gépi utasításokat ad ki, hanem CIL-re fordít.

Mindez, ahogy láttuk, az összes .NET-alapú fordítóra igaz. Ennek illusztrálására, tételezzük fel, hogy ugyanezt az alkalmazást C# helyett Visual Basic .NET segítségével is létrehoztuk:

```
' Calc.vb
Imports System

Namespace CalculatorExample
' A VB "Module" is a class that contains only
' static members.
Module Program
    Sub Main()
        Dim c As New Calc
        Dim ans As Integer = c.Add(10, 84)
        Console.WriteLine("10 + 84 is {0}.", ans)
        Console.ReadLine()
    End Sub
End Module

Class Calc
    Public Function Add(ByVal x As Integer, ByVal y As Integer)
        As Integer
        Return x + y
    End Function
End Class
End Namespace
```

Ha megnézzük az Add() metódus köztes nyelvét, ott hasonló utasításokat találhatunk (a VB .NET-fordító [vbc.exe] által optimalizálva):

```
.method public instance int32 Add(int32 x, int32 y) cil managed
{
    // A kód mérete 8 (0x8)
    .maxstack 2
    .locals init (int32 v_0)
    IL_0000: ldarg.1
    IL_0001: ldarg.2
    IL_0002: add.ovf
    IL_0003: stloc.0
    IL_0004: br.s IL_0006
    IL_0006: ldloc.0
    IL_0007: ret
} // a Calc::Add metódus vége
```

---

**Forráskód** A Calc.cs és a Calc.vb kódfájlokat az 1. fejezet alkönyvtára tartalmazza.

---

## A köztes nyelv előnyei

Adódik a kérdés: miért volt jó, hogy köztes nyelvre fordítottuk a forráskódot, és nem közvetlenül egy specifikus utasításkészletbe. Ennek egyik oka a nyelvi integráció. Ahogy már láttuk, minden egyes .NET-alapú fordító közel azonos CIL-utasításokat hoz létre. Így az összes nyelv egy jól meghatározott bináris térben kommunikálhat egymással.

Ráadásul, mivel a CIL platformfüggetlen, és maga a .NET-keretrendszer is az, így a Java-fejlesztők számára már jól ismert előnyöket biztosítja (vagyis egy szimpla kódbázis fut több operációs rendszeren). A C# nyelvre létezik nemzetközi szabvány, és a .NET-platform, valamint implementációinak a nagy része már több nem Windows operációs rendszeren is rendelkezésre áll (további információkért lásd a fejezet későbbi részét). Ellentétben a Javával, a .NET lehetővé teszi, hogy a kívánt nyelven hozzunk létre alkalmazásokat.

## A köztes nyelv fordítása platformfüggetti utasítássá

Mivel a szerelvények inkább CIL-utasításokat, mintsem platformfüggetti utasításokat tartalmaznak, a köztes nyelvi kódot használat előtt, menet közben kell lefordítani. Azt az egységet, amely a CIL-kódot CPU-utasításokká alakítja, *azonnal (JIT) fordítónak* nevezzük, néha tréfásan *Jitternek*. A .NET-futtatókörnyezet minden futó processzor számára fenntart egy JIT-fordítót, s ezek mindegyikét az alapplatformra optimalizálták.

Ha például egy kézieszközbe (pl. Pocket PC-be) szánt alkalmazást készítünk, a megfelelő Jitter kitűnően fut a kevés memóriával rendelkező környezetben. Viszont, ha egy kiszolgálóra (ahol memóriában ritkán van hiány) telepítjük a szerelvényt, a Jittert a sok memóriával rendelkező környezethez optimalizáljuk. Elegendő tehát, ha a fejlesztők megírják azt a kódot, amely JIT-fordítással hatékonyan lefordítható, és a gépeken különböző architektúrákkal is végrehajtható.

Ráadásul, mivel a Jitterek a CIL-utasításokat megfelelő gépi kódokká fordítják, az eredményeket a memóriában a céloperációs rendszernek megfelelő formátumban gyorsítótárazzák. Így, ha a `PrintDocument()` metódust hívjuk meg, a CIL-utasításokat a rendszer az első híváskor platformfüggetti utasításokká fordítja, és a későbbi felhasználás érdekében a memóriában tárolja. Ebből következően, ha legközelebb meghívjuk a `PrintDocument()` metódust, a CIL-utasítást nem kell újra lefordítani.

---

**Megjegyzés** Azt is megtehetjük, hogy amikor a .NET Framework 3.5 SDK-ban lévő `ngen.exe` parancssori eszköz segítségével telepítjük az alkalmazást, végrehajtunk egy JIT-előfordítást a szerelvényen. Ezzel tökéletesíthetjük a grafikai intenzív alkalmazások indulási idejét.

---

## A .NET-típusmetaadat szerepe

A .NET-szerelvények a CIL-utasításokon kívül olyan teljes és szabatos metaadatot tartalmaznak, amely a binárisban definiált összes típust (osztály, struktúra, felsorolt típus stb.), valamint azok tagjait (tulajdonságok, metódusok, események stb.) is leírja. Szerencsére a legfrissebb és legnagyobb típusmetaadat előállításuk mindig a fordító (és nem a programozó) feladata. Mivel a .NET-metaadat rendkívül aprólékos, a szerelvények teljesen önleíró egységek.

A .NET-típusmetaadat formátumának szemléltetésére, nézzük meg a korábban megvizsgált, C# `Calc` osztály `Add()` metódusa számára generált metaadatot (az `Add()` metódus VB .NET verziója számára generált metaadat is hasonló):

TypeDef #2 (02000003)

```
-----
TypDefName: CalculatorExample.Calc (02000003)
Flags      : [NotPublic] [AutoLayout] [Class]
[AnsiClass] [BeforeFieldInit] (00100001)
Extends    : 01000001 [TypeRef] System.Object
Method #1 (06000003)
```

```
-----
MethodName: Add (06000003)
Flags      : [Public] [HideBySig] [ReuseSlot] (00000086)
RVA        : 0x00002090
ImplFlags  : [IL] [Managed] (00000000)
CallConvtn: [DEFAULT]
hasThis
ReturnType: I4
  2 Arguments
    Argument #1: I4
    Argument #2: I4
  2 Parameters
    (1) ParamToken : (08000001) Name : x flags: [none] (00000000)
    (2) ParamToken : (08000002) Name : y flags: [none] (00000000)
```

A metaadatot a .NET-futtatókörnyezet számos célra felhasználja ugyanúgy, ahogy a különféle fejlesztőeszközök. Az IntelliSense szolgáltatás például – ezt az olyan eszközök, mint a Visual Studio 2008 tartalmazzák – lehetővé teszi, hogy a szerelvények metaadata a tervezés folyamán olvasható legyen. A metaadatot különféle objektumböngésző segédprogramok, hibakereső eszközök és maga a C#-fordító is használja. Kétségtelen, hogy a metaadat a gerince számos .NET-technológiának, köztük a WCF-nek, az XML-webszolgáltatásoknak vagy a .NET remoteingnak, a reflexiónak, a késői kötésnek és az objektumsorositásnak. (A 16. fejezet tartalmazza a .NET-metaadat szerepének formalizálását.)



## A szerelvény manifesztumának szerepe

Végül, de nem utolsósorban, ahogy már szó volt róla, a .NET-szerelvények olyan metaadatot tartalmaznak, amely magát a szerelvényt írja le (szakszóval *manifesztum*, *önleírás*). Egyéb részletek mellett, a manifesztum rögzíti az aktuális szerelvény megfelelő működéséhez szükséges összes külső szerelvényt, a szerelvény verziószámát, a szerzői jogra vonatkozó adatokat stb. Ugyanúgy, mint a típusmetaadatot, a szerelvény manifesztumát is a fordító generálja. Íme a fejezetben korábban bemutatott `calc.csc` kódfájl fordításakor generált manifesztum néhány releváns részlete (tételezzük fel, hogy utasítottuk a fordítót, hogy a szerelvényt `calc.exe`-nek nevezze el):

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
.assembly Calc
{
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module Calc.exe
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
```

Röviden, a manifesztum dokumentálja a `Calc.exe` számára szükséges külső szerelvények listáját (az `.assembly extern` direktíva révén), valamint a szerelvény különböző jellemzőit (verziószám, modulnév stb.). (Az önleíró adatok hasznosságáról a 15. fejezetben lesz szó részletesebben.)

## A közös típusrendszer (CTS)

Egy adott szerelvény bármennyi különböző típust tartalmazhat. A .NET világában a *típus* általános kifejezés, amely a halmaz (osztály, interfész, struktúra, felsorolt típus, metódusreferencia) egy tagjára utal. Amikor egy .NET-alapú nyelv segítségével hozunk létre megoldásokat, valószínűleg sok ilyen típussal találkozunk. Például akkor, amikor a szerelvényünk olyan osztályt definiál, amely néhány interfészt implementál.

Előfordulhat, hogy az egyik interfészmetódus bemeneti paramétere egy felsorolt típus lesz, a hívónak pedig egy struktúrát ad vissza.

Ahogy láttuk, a közös típusrendszer (CTS) olyan formális specifikáció, amely azt dokumentálja, hogyan kell a típusokat úgy definiálni, hogy azokat a közös nyelvi futtatórendszer (CLR) befogadja. Jellemzően kizárólag azokat a fejlesztőket érdekli a közös típusrendszer belső működése, akik a .NET-platformot megcélzó eszközöket és/vagy fordítóprogramokat készítenek. Minden .NET-programozó számára fontos azonban, hogy megtanulja, hogyan kell a közös típusrendszer által definiált öt típussal bármilyen választott nyelven dolgozni. Az alábbiakban következzen egy rövid áttekintés.

## Az egységes típusrendszer osztálytípusai

Minden .NET-alapú nyelv támogatja legalább az *osztálytípus* fogalmát, amely az objektumorientált programozás (OOP) alappillére. Egy osztálynak bármennyi tagja (pl. tulajdonságok, metódusok és események) és adatpontja (mezője) lehet. A C#-ban a `class` kulcsszóval deklaráljuk az osztályokat:

```
// C# osztálytípus.
class Calc
{
    public int Add(int x, int y)
    { return x + y; }
}
```

A C# segítségével készült CTS-osztályok felépítését az 5. fejezetben vizsgáljuk meg; az 1.1. táblázatban felsoroljuk az osztálytípusok néhány jellemzőjét.

| Osztályjellemző                             | Valós jelentése                                                                                                                                                                                                                                     |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lezárt-e az osztály, vagy sem?              | A lezárt osztályok nem lehetnek más osztályok alaposztályai.                                                                                                                                                                                        |
| Megvalósít az osztály bármilyen interfészt? | Az <i>interfész</i> olyan absztrakt tagok gyűjteménye, amelyek kapcsolatot biztosítanak az objektum és az objektumfelhasználó között. Az egységes típusrendszer lehetővé teszi, hogy egy osztály vagy szerkezet bármennyi interfészt megvalósítson. |
| Az osztály absztrakt vagy valós?            | Az <i>absztrakt osztályokat</i> nem lehet közvetlenül létrehozni, de egységes viselkedést biztosítanak a származtatott típusoknak. A <i>valós osztályokat</i> közvetlenül létre lehet hozni.                                                        |

| Osztályjellemző                  | Valós jelentése                                                                                                                                                                                    |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Milyen az osztály „láthatósága”? | Minden osztályt egy láthatósági attribútummal kell konfigurálni. Alapvetően ez a jellemző határozza meg, hogy használhatják-e külső szerelvények is az osztályt, vagy csak a definiáló szerelvény. |

1.1. táblázat: Az egységes típusrendszer osztályjellemzői

## Az egységes típusrendszer interfésztípusai

Az *interfész* nem más, mint absztrakt tagok definícióinak megnevezett gyűjteménye, amelyet egy adott osztály vagy struktúra támogathat (megvalósíthat). Eltérően a COM-tól, a .NET-interfészek *nem* egy közös ősi interfészből származnak (mint az `unknown`). A C#-ban az interfésztípusokat az `interface` kulcsszó segítségével definiálhatjuk:

```
// A C# interfész típust rendszerint publikus
// típusként deklaráljuk, így más szerelvények típusai
// is megvalósíthatják a viselkedését.
public interface IDraw
{
    void Draw();
}
```

Az interfészeknek önmagukban nincs sok hasznuk. Viszont, amikor egy osztály vagy struktúra megvalósít egy adott interfészt a maga egyedi módján, egy interfész-hivatkozás révén polimorf módon hozzáférhetünk a funkcionálitáshoz. (Az interfészalapú programozást a 9. fejezetben tárgyaljuk.)

## Az egységes típusrendszer struktúratípusai

A struktúra fogalmát ugyancsak formalizálták az egységes típusrendszerben. Ha a C-t ismerjük, akkor egyértelmű, hogy ezek a felhasználói típusok (UDT-k) a .NET világában is tovább élnek (bár egy kicsit másképp viselkednek). Egyszerűen fogalmazva, a *struktúrát* egy értékalapú szemantikával rendelkező pehelysúlyú osztálytípusnak is tekinthetjük. (A struktúrákkal kapcsolatos további részletekért, lásd a 4. fejezetet.) A struktúrák jellemzően geometriai és matematikai adatok modellezésére alkalmasak, és a C#-ban a `struct` kulcsszó segítségével lehet létrehozni őket:

```
// C# struktúratípus.
struct Point
{
    // A struktúrák mezőket tartalmazhatnak.
    public int xPos, yPos;

    // A struktúrák paraméterezett konstruktorokat tartalmazhatnak.
    public Point(int x, int y)
    { xPos = x; yPos = y; }

    // A struktúrák metódusokat definiálhatnak.
    public void Display()
    {
        Console.WriteLine("{0}, {1}", xPos, yPos);
    }
}
```

## Az egységes típusrendszer felsorolt típusai

A *felsorolt típus* olyan praktikus programozási szerkezet, amely lehetővé teszi a név/érték párok csoportosítását. Például tételezzük fel, hogy olyan videojáték-alkalmazást hozunk létre, amely lehetővé teszi a játékosok számára, hogy három karakterkategóriából válasszanak (Wizard – varázsló, Fighter – harcos vagy Thief – tolvaj). A numerikus értékek nyomon követése helyett, amelynek célja az egyes lehetőségek egyesével történő bemutatása lenne, egyedi felsorolt típusokat is létrehozhatunk, az enum kulcsszó segítségével:

```
// C# felsorolt típus.
enum CharacterType
{
    18 CHAPTER 1 n THE PHILOSOPHY OF .NET
    8849CH01.qxd 10/1/07 10:30 AM Page 18
    Wizard = 100,
    Fighter = 200,
    Thief = 300
}
```

Az alapértelmezés szerint minden egyes elemet 32 bites egész tárol; módosíthatjuk azonban a tárolót, ha szükséges (pl. amikor egy Pocket PC-hez hasonló, kevés memóriás eszközre programozunk). Az egységes típusrendszer emellett azt igényli, hogy a felsorolt típusok egy közös, `System.Enum` őosztályból származzanak. Ahogy a 4. fejezetből később kiderül: ez az alaposztály egy sor olyan érdekes tagot definiál, amelyek révén a mögöttes név/érték párokat programozással lehet kibontani, manipulálni és átalakítani.

## Az egységes típusrendszer metódusreferencia-típusai

A *metódusreferencia* a típusbiztos C-stílusú függvénytmutató .NET-megfelelője. A legfőbb különbség az, hogy a .NET-metódusreferencia a `System.MulticastDelegate`-ből származó *osztály*, nem pedig egy nyers memóriacím egyszerű mutatója. A C#-ban a metódusreferenciákat a `delegate` kulcsszóval deklaráljuk:

```
// Ez a C# metódusreferencia típus bármely metódusra mutathat,  
// amely egész számot ad vissza, és két egész szám a bemenete.  
delegate int BinaryOp(int x, int y);
```

A metódusreferenciák egyrészt akkor nagyon hasznosak, ha valamelyik egyed számára lehetőséget szeretnénk teremteni arra, hogy egy másiknak hívást továbbítson, másrészt a .NET-eseményarchitektúra alapjainak megteremtésekor. A metódusreferenciák a többes küldéshez valódi támogatással rendelkeznek (pl. egy kérés több címzettnek való továbbításához), valamint az aszinkron függvényhíváshoz (részletesen lásd a 11. és 18. fejezetben).

## Az egységes típusrendszer típustagjai

Az egységes típusrendszerben formalizált összes típus közül a legtöbbnek bármennyi *tagja* is lehet. Szaknyelven szólva, a *típustagot* a halmaz korlátozza (konstruktor, véglegesítő, statikus konstruktor, beágyazott típus, operátor, metódus, tulajdonság, indexelő, mező, írásvédett mező, konstans, esemény).

Az egységes típusrendszer különféle „díszítőelemeket” definiál, amelyeket hozzárendelhetünk egy adott taghoz. Minden tag rendelkezik például egy adott láthatósági jellemzővel (pl. nyilvános, privát, védett stb.). Néhány tag absztrakt tagként deklarálható, amely a származtatott típusokra polimorf viselkedést kényszerít, valamint virtuális tagként, amely alapértelmezett (de felüldefiniálható) megvalósítást definiál. Sőt a legtöbb tagot statikusként (osztályszinten kötött) vagy példányként (objektumszinten kötött) is konfigurálhatjuk. A típustagok létrehozását a következő néhány fejezet során vizsgáljuk meg.

---

**Megjegyzés** A C# nyelv támogatja a generikus típusok és tagok létrehozását is (lásd a 10. fejezetben).

---

## Valódi CTS-adattípusok

Az utolsó szempontként említhető az egységes típusrendszer vizsgálatában, hogy alapvető adattípusok jól meghatározott halmazát hozza létre. Habár minden nyelv rendelkezik egyedi kulcsszóval a valódi CTS-adattípusok deklarálására, az összes kulcsszó az `mscorlib.dll` nevű szerelvényben definiált típust adja meg. Nézzük meg az 1.2. táblázat alapján, hogy milyen kulcsszavakkal fejezzük ki a CTS-adattípusokat a különböző .NET-nyelvekben.

| CTS-adattípus               | VB .NET-kulcsszó      | C#-kulcsszó          | C++/CLI-kulcsszó                                          |
|-----------------------------|-----------------------|----------------------|-----------------------------------------------------------|
| <code>System.Byte</code>    | <code>Byte</code>     | <code>byte</code>    | <code>unsigned char</code>                                |
| <code>System.SByte</code>   | <code>SByte</code>    | <code>sbyte</code>   | <code>signed char</code>                                  |
| <code>System.Int16</code>   | <code>Short</code>    | <code>short</code>   | <code>short</code>                                        |
| <code>System.Int32</code>   | <code>Integer</code>  | <code>int</code>     | <code>int</code> vagy <code>long</code>                   |
| <code>System.Int64</code>   | <code>Long</code>     | <code>long</code>    | <code>__int64</code>                                      |
| <code>System.UInt16</code>  | <code>UShort</code>   | <code>ushort</code>  | <code>unsigned short</code>                               |
| <code>System.UInt32</code>  | <code>UInteger</code> | <code>uint</code>    | <code>unsigned int</code> vagy <code>unsigned long</code> |
| <code>System.UInt64</code>  | <code>ULong</code>    | <code>ulong</code>   | <code>unsigned __int64</code>                             |
| <code>System.Single</code>  | <code>Single</code>   | <code>float</code>   | <code>Float</code>                                        |
| <code>System.Double</code>  | <code>Double</code>   | <code>double</code>  | <code>Double</code>                                       |
| <code>System.Object</code>  | <code>Object</code>   | <code>object</code>  | <code>Object^</code>                                      |
| <code>System.Char</code>    | <code>Char</code>     | <code>char</code>    | <code>wchar_t</code>                                      |
| <code>System.String</code>  | <code>String</code>   | <code>String</code>  | <code>String^</code>                                      |
| <code>System.Decimal</code> | <code>Decimal</code>  | <code>decimal</code> | <code>Decimal</code>                                      |
| <code>System.Boolean</code> | <code>Boolean</code>  | <code>bool</code>    | <code>Bool</code>                                         |

1.2. táblázat: A valódi CTS-adattípusok

Mivel egy felügyelt nyelv egyedi kulcsszavai csak a `system` névtér valódi típusainak rövidített jelölései lehetnek, többé nem kell a numerikus adatok túlcsoportolásáról, illetve alácsopordulásáról tartanunk a különböző nyelveken, ahogy a sztringek és a logikai értékek belső megjelenésétől sem.

Nézzük meg a következő kódrészleteket, amelyek nyelvi kulcsszavak és formális CTS-típusok segítségével definiálnak 32 bites numerikus változókat C# és VB .NET nyelveken.

```
// Egész számok definíciója C# nyelvben.
int i = 0;
System.Int32 j = 0;

' Egész számok definíciója "ints" VB .NET-ben.
Dim i As Integer = 0
Dim j As System.Int32 = 0
```

## A közös nyelvi specifikáció (CLS)

A különböző nyelvek tehát ugyanazokat a programozási szerkezeteket egyedi, nyelvspecifikus kifejezésekkel fejezik ki. A C# nyelvben például a sztringek összefűzését a plusz operátorral (+) jelöljük, míg VB .NET-ben az és jelet (&) használjuk. Ha két különböző nyelv ugyanazt a programozási kifejezést használja is (pl. egy olyan függvényt, amelynek nincs visszatérési értéke), nagy az esély, hogy a szintaxis mégis elég eltérő lesz:

```
// C# metódus, amely nem ad vissza semmit.
public void MyMethod()
{
    // Érdekes kód...
}
' VB metódus, amely nem ad vissza semmit.
Public Sub MyMethod()
    ' Érdekes kód...
End Sub
```

Ezek az apró szintaktikai eltérések azonban nem fontosak a .NET-futtatókörnyezet szempontjából, mivel a megfelelő fordítók (ebben az esetben a csc.exe vagy a vb.exe) hasonló CIL-utasításokat adnak ki. A nyelvek viszont a működésük általános szintje alapján is eltérőek lehetnek.

Egy .NET-nyelv például vagy rendelkezik kulcsszóval az előjel nélküli adatok megjelenítésére, vagy nem, illetve vagy támogatja a mutatótípusokat, vagy nem. Ilyen lehetőségek mellett jó lenne, ha volna egy olyan referencia-adat, amellyel az összes .NET-alapú nyelv összhangban lenne.

A közös nyelvi specifikáció olyan szabálykészlet, amely részletesen leírja, hogy egy adott .NET-alapú fordítónak milyen minimális és teljes szolgáltatásokat kell támogatnia a közös nyelvi futtatórendszerben (CLR) hosztolt forráskódok létrehozásához, illetve ahhoz, hogy minden .NET-platformot célzó nyelv egységes módon férhessen hozzá. A közös nyelvi specifikáció több szempontból az egységes típusrendszer által definiált, teljes funkcionalitás *részhalmazának* tekinthető.

A közös nyelvi specifikáció végül is olyan szabálykészlet, amelyhez a fordító megalkotóinak igazodniuk kell, ha azt szeretnék, hogy a termékük zavartalanul működjön a .NET-univerzumban. Minden szabályhoz olyan egyszerű nevet rendeltek hozzá (pl. 6. CLS-szabály), amely leírja, hogy a szabály milyen hatással van azokra, akik a fordítót létrehozzák, és azokra, akik (valamilyen módon) kapcsolatba kerülnek vele. A közös nyelvi specifikáció alapja a 1. szabály:

- *1. szabály:* A CLS-szabályok a típusnak csak azokra a részeire vonatkoznak, amelyek a definiáló szerelvényen kívülről is hozzáférhetők.

Ebből a szabályból (helyesen) arra következtethetünk, hogy a többi CLS-szabály nem vonatkozik a .NET-típusok belső működésének létrehozásakor alkalmazott logikára. A típusoknak egyetlen szempontból kell összhangban lenniük a közös nyelvi specifikációval: a tagok definíciója (azaz elnevezési rend, paraméterek és visszatérési típusok) alapján. A tagok megvalósítási logikája bármennyi nem CLS-technikát használhat, hiszen a külvilág nem veszi észre a különbséget.

Ennek illusztrálására nézzük az alábbiakat. A következő `Add()` metódus nem CLS-kompatibilis, ugyanis a paraméterek és a visszatérési értékek előjel nélküli adatokat használnak (és ez nem CLS-követelmény):

```
class Calc
{
    // Előjel nélküli adatok hozzáférése nem CLS-kompatibilis!
    public ulong Add(ulong x, ulong y)
    { return x + y; }
}
```

Ha viszont belsőleg, egyszerűen előjel nélküli adatokat használtunk volna:



```

class Calc
{
    public int Add(int x, int y)
    {
        // Mivel az ulong változót csak belsőleg használjuk,
        // a kód továbbra is CLS-kompatibilis.
        ulong temp = 0;
        ...
        return x + y;
    }
},

```

akkor még mindig megfelelnek a CLS-szabályoknak, és nyugodtak lehetnénk afelől, hogy bármelyik .NET-nyelv képes meghívni az Add() metódust.

A közös nyelvi specifikáció persze számos egyéb szabályt is definiál az 1. szabályon kívül. Leírja például, hogy egy adott nyelvnek hogyan kell a sztringeket és a felsorolt típusokat (belül tárolt típus) belsőleg reprezentálni, hogyan definiáljunk statikus tagokat, stb. Szerencsére nem kell ezeket a szabályokat megjegyeznünk ahhoz, hogy profi .NET-fejlesztők legyünk. Összegezve, a közös nyelvi specifikáció és az egységes típusrendszer mélyebb ismerete csak az eszköz-, illetve fordítóalkotók számára szükséges.

## A CLS-megfeleltetés biztosítása

Ahogy azt a későbbiekben látni fogjuk, a C# számos, nem CLS-kompatibilis programszerkezetet definiál. Jó hír azonban, hogy egy .NET-attribútum segítségével utasíthatjuk a C#-fordítót, hogy ellenőrizze a kódunk CLS-megfeleltetését:

```

// Utasítja a C#-fordítót a kód CLS-megfeleltetésének ellenőrzésére.
[assembly: System.CLSCompliant(true)]

```

A [CLSCompliant] attribútum utasítja a C#-fordítót, hogy ellenőrizze, a kód minden egyes sora megfelel-e a CLS-szabályoknak. Ha szabályellenes dolgot talál, a fordító hibát jelez, és megmutatja a hibás kódot. (Az attribútumalapú programozás részleteit a 16. fejezetben tárgyaljuk.)

## A közös nyelvi futtatórendszer (CLR)

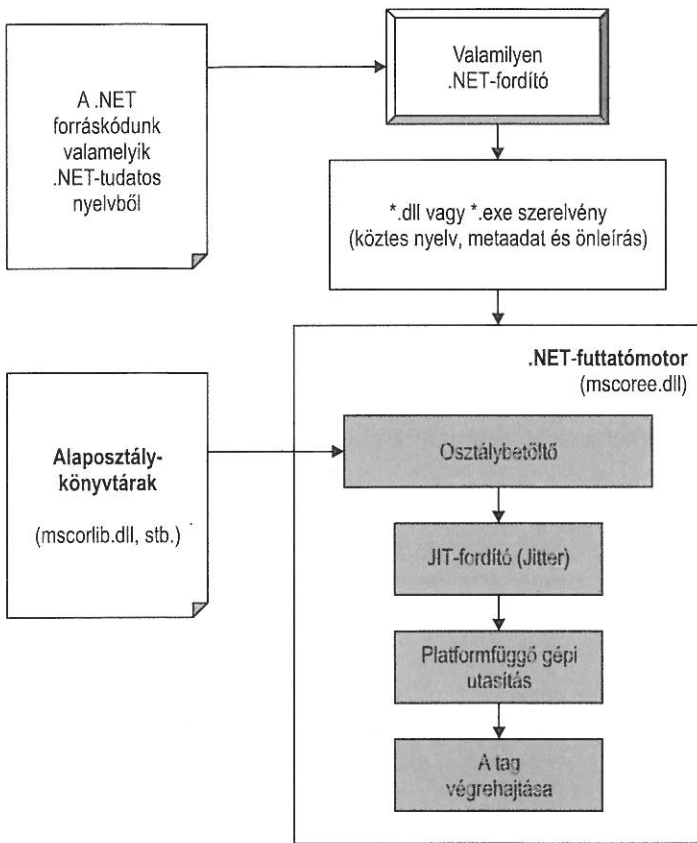
Az egységes típusrendszeren (CTS) és a közös nyelvi specifikáción (CLS) kívül a harmadik hárombetűs rövidítés, amelyről még beszélnünk kell, a közös nyelvi futtatórendszer (CLR). Programozási szempontból a *futtatórendszer* kifejezés olyan külső szolgáltatások gyűjteményére utal, amelyek a kód egy adott lefordított egységének a végrehajtásához szükségesek. Például, amikor a fejlesztők a Microsoft Foundation Classes (MFC) segítségével hoznak létre új alkalmazást, tisztában vannak azzal, hogy a programjuk igényli az MFC-alapkönyvtárat (azaz az `mfc42.dll`-t). A többi népszerű nyelvnek is megvan a megfelelő futtatórendszere. A VB6-programozóknak szintén egy vagy két futtatókörnyezeti modulhoz kell igazodniuk (pl. `msvbvm60.dll`). A Java-fejlesztők a Java virtuális géphez (JVM) kötődnek; stb.

A .NET-platform egy további futtatási rendszert kínál. A legnagyobb különbség a .NET-futtatórendszer és a többi, imént említett futtatórendszer között az, hogy a .NET-futtatórendszer rendelkezik egy jól meghatározott futtatási réteggel, amelyet az *összes* .NET-alapú nyelv és -platform használ.

A közös nyelvi futtatórendszer lényege fizikailag az `mscorlib.dll` könyvtárban (más néven Common Object Runtime Execution Engine) jelenik meg. Amikor egy szerelvényt használni szeretnénk, és hivatkozunk rá, az `mscorlib.dll` automatikusan betöltődik, és ez tölti be a memóriába a szerelvényt. A futtatórendszer számos feladatért felelős. Elsősorban megadja egy szerelvény helyét, és a binárisban lévő metaadatot olvasva megkeresi az igényelt típust. A CLR ezt követően a memóriában helyezi el a típust, a hozzárendelt CIL-utasítást platformfüggő gépi utasításokká fordítja, végrehajtja a szükséges biztonsági ellenőrzéseket, majd végrehajtja a szóban forgó kódot.

Az egyedi szerelvények betöltése és az egyedi típusok létrehozása mellett, a CLR, ha szükséges, a .NET-alaposztálykönyvtárban tárolt típusokkal is kommunikál. Habár a teljes alaposztálykönyvtár több különálló szerelvényre oszlik, a legfontosabb szerelvény (`mscorlib.dll`) sok alaptípust tartalmaz, amelyek között sok közös programozási feladat található az összes .NET-nyelv által használt alapvető adattípussal egyetemben. Amikor .NET-megoldásokat hozunk létre, automatikusan hozzáférünk ehhez a szerelvényhez.

Az 1.4. ábra a forráskód (amely az alaposztálykönyvtár-típusokat használja), egy adott .NET-fordító és a .NET-futtatómotor közötti munkafolyamot mutatja be.



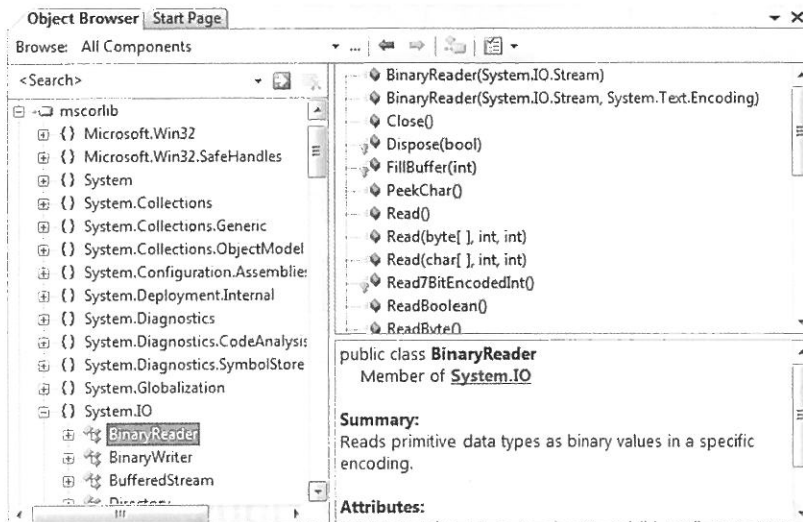
1.4. ábra: Az mscoree.dll működés közben

## A szerelvény, a névtér és a típus megkülönböztetése

Mindannyian elfogadjuk a forráskódkönyvtárak fontosságát. Az MFC-hez, J2EE-hez és az ATL-hez hasonló könyvtáraknak az a célja, hogy a meglévő forráskódok olyan jól meghatározott készletét biztosítsa a fejlesztők számára, amelyet az alkalmazásaikban felhasználhatnak. A C# nyelv azonban nem rendelkezik nyelvfüggő kódkönyvtárral. A C#-fejlesztők ehelyett a nyelvsemleges .NET-könyvtárakat használják. Annak érdekében, hogy a típusok rendezetten helyezkedjenek el az alaposztálykönyvtárakban, a .NET-platform messzemenően alkalmazza a *névtér* elvét.

Egyszerűen fogalmazva, a névtér az egy szerelvényben lévő, szemantikailag kapcsolódó típusok csoportja. A `System.IO` névtér például I/O-típusokat tartalmaz, a `System.Data` névtér pedig alapvető adatbázistípusokat definiál; stb. Hangsúlyoznunk kell, hogy egyetlen szerelvény (pl. az `microsoft.dll`) bármennyi névtérrel tartalmazhat, s ezek mindegyike bármennyi típussal rendelkezhet.

Ennek illusztrálására mutatja be az 1.5. ábra a Visual Studio 2008 Object Browser-ének egyik képernyőfotóját. Ezzel az eszközzel megvizsgálhatjuk azokat a szerelvényeket, amelyekre az aktuális projektünk hivatkozik, az egy adott szerelvényben lévő névtéreket, az egy névtérben lévő típusokat, valamint egy bizonyos típus tagjait. Az `microsoft.dll` tehát több különböző névtérrel tartalmaz, mindegyiket a saját, szemantikailag kapcsolódó típusaival.



1.5. ábra: Egy szerelvény bármennyi névtérrel tartalmazhat

A legfontosabb különbség e között a felfogás és egy MFC-hez hasonló, nyelv-függő könyvtár között, hogy bármelyik nyelv, amelyik a .NET-futtatórendszer veszi célba, *ugyanazokat* a névtéreket és *ugyanazokat* a típusokat használja. A következő három program ugyanazt a „Hello World” alkalmazást mutatja be C#, VB .NET és C++/CLI nyelveken:

```
// Hello world C# nyelven
using System;

public class MyApp
{
    static void Main()
    {
        Console.WriteLine("Hi from C#");
    }
}

' Hello world VB nyelven
Imports System

Public Module MyApp
    Sub Main()
        Console.WriteLine("Hi from VB")
    End Sub
End Module

// Hello world C++/CLI nyelven
#include "stdafx.h"
using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hi from C++/CLI");
    return 0;
}
```

Minden egyes nyelv használja a `System` névtérben definiált `Console` osztályt. Apróbb szintaktikai eltéréseken túl, ez a három alkalmazás nagyon sokban hasonlít egymásra, mind fizikailag, mind logikailag.

.NET-fejlesztőként az elsődleges célunk természetesen a számos .NET-névtérben definiált, temérdek típus megismerése. A legalapvetőbb névtér, amellyel találkozunk, a `System`. Ez a névtér biztosítja azoknak a típusoknak az alapvető készletét, amelyekre .NET-fejlesztőként időről időre szükségünk van. Valójában egyetlen működőképes C#-alkalmazást sem hozhatunk létre anélkül, hogy legalább ne hivatkoznánk a `System` névtérre, hiszen az alapvető adattípusokat (`System.Int32`, `System.String` stb.) ez a névtér definiálja. Az 1.3. táblázatban a kapcsolódó működések alapján csoportosított, néhány (de nem az összes) .NET-névtér látható.

| .NET-névtér                                                                                               | Valós jelentése                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System                                                                                                    | A System névtérben számos hasznos típust találunk, amelyek valós adatokkal, matematikai számításokkal, véletlenszám-generálással, környezeti változókkal és szemétygyűtéssel foglalkoznak, továbbá közösen használt kivételeket és attribútumokat. |
| System.Collections<br>System.Collections.Generic                                                          | Ezek a névterek számos tárolótípust definiálnak, valamint alaptípusokat és interfészeket, amelyek révén testre szabott gyűjteményeket építhetünk fel.                                                                                              |
| System.Data<br>System.Data.Odbc<br>System.Data.OracleClient<br>System.Data.OleDb<br>System.Data.SqlClient | Ezeket a névtereket a relációs adatbázisokkal folytatott kommunikációra használjuk az ADO .NET alkalmazásakor.                                                                                                                                     |
| System.IO<br>System.IO.Compression<br>System.IO.Ports                                                     | Ezek a névterek olyan típusokat definiálnak, amelyeket fájl I/O-ra, adattömörítésre és portkezelésre használunk.                                                                                                                                   |
| System.Reflection<br>System.Reflection.Emit                                                               | Ezek a névterek olyan típusokat definiálnak, amelyek támogatják a futási időben történő típusmeghatározást, valamint a típusok dinamikus létrehozását.                                                                                             |
| System.Runtime.InteropServices                                                                            | Ez a névtér biztosítja a .NET-típusok számára azt a képességet, hogy a nem felügyelt kódokkal (pl. C-alapú DLL-ek és Com-kiszolgálók) kommunikáljanak és viszont.                                                                                  |
| System.Drawing<br>System.Windows.Forms                                                                    | Ezek a névterek olyan típusokat definiálnak, amelyek révén a .NET eredeti felhasználói felületének eszközkészletét (Windows-űrlapok) használó asztali alkalmazásokat hozhatunk létre.                                                              |
| System.Windows<br>System.Windows.Controls<br>System.Windows.Shapes                                        | A System.Windows névtér, számos egyéb névtér gyökere, amely a Windows Presentation Foundation (WPF) felhasználói felületi eszközkészletet reprezentálja.                                                                                           |
| System.Linq<br>System.Xml.Linq<br>System.Data.Linq                                                        | Ezek a névterek a LINQ API-ban végzett programozás során alkalmazott típusokat definiálják.                                                                                                                                                        |

| .NET-névtér                                           | Valós jelentése                                                                                                                                                     |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System.Web                                            | Ez az egyik olyan névtér a sok közül, amelyik lehetővé teszi az ASP .NET webalkalmazások létrehozását.                                                              |
| System.ServiceModel                                   | Ez az egyik olyan névtér a sok közül, amelyekkel a WCF API segítségével elosztott alkalmazásokat hozhatunk létre.                                                   |
| System.Workflow.Runtime<br>System.Workflow.Activities | Ez a két névtér olyan típusokat definiál, amelyekkel a WF API segítségével munkafolyamat-engedélyezett alkalmazásokat hozhatunk létre.                              |
| System.Threading                                      | Ez a névtér számos olyan típust definiál, amelyeket többszálú alkalmazások létrehozására használunk.                                                                |
| System.Security                                       | A Security a .NET-univerzum integrált nézőpontja. A biztonságos névterekben számos olyan típust találunk, amelyek engedélyekkel, kriptográfiával stb. foglalkoznak. |
| System.Xml                                            | Az XML-központú névterek számos olyan típust tartalmaznak, amelyeket az XML-ada-<br>tokkal való kommunikációra használunk.                                          |

1.3. táblázat: A .NET-névterek áttekintése

## A Microsoft-névterek szerepe

Az 1.3. táblázatot végigolvasva bizonyára feltűnt, hogy jó néhány beágyazott névtérnek (System.IO, System.Data stb.) a System a gyökérnévtére. Ahogy azonban kiderült, a .NET-alapostálykönyvtár definiálja a System névtéren kívüli legfontosabb gyökérnévterek nagy részét, amelyek közül a leghasznosabb a Microsoft.

Dióhéjban, a Microsoft-névtérbe ágyazott névterek (pl. Microsoft.CSharp, Microsoft.Ink, Microsoft.ManagementConsole és Microsoft.Win32) olyan típusokat tartalmaznak, amelyek egyedi, Windows operációs rendszerbeli szolgáltatásokkal való kommunikációra használatosak. Így ezeket a típusokat más .NET-engedélyezett operációs rendszerrel (pl. a Mac OS X-szel) nem használhatjuk. Könyvünk a Microsoft-gyökerű névterek részleteit azonban nem tárgyalja.

**Megjegyzés** A .NET Framework 3.5 SDK dokumentációjának használatát a 2. fejezet írja le, részletekkel szolgálva a névterekről, a típusokról és az alapsztálykönyvtárakban található tagokról.

---

## Programozott hozzáférés a névterekhez

A névtér tehát nem más, mint a kapcsolódó típusok logikai megértésének és elrendezésének a megfelelő módja. A `System` névtérre gondolva, azt feltételezhetjük, hogy a `System.Console` egy olyan `Console` nevű osztályt jelent, amely a `System` nevű névtérben található. A .NET-futtatórendszer szerint azonban ez nem így van. A futtatómodul csak egy `System.Console` nevű egységet lát.

A C#-ban a `using` kulcsszó leegyszerűsíti a névterekben definiált típusokra történő hivatkozás folyamatát. A működése a következő. Tegyük fel, hogy egy hagyományos asztali alkalmazást szeretnénk létrehozni. A főablak egy kiszolgálóoldali adatbázisból származó adatokra épülő oszlopdiagramot jelenít meg, valamint a cégünk logóját. Amíg kevéssé ismerjük még a névterekben lévő típusokat, íme néhány, amelyekre hivatkozhatunk a programunkban:

```
// Íme, az alkalmazás elkészítéséhez szükséges összes névtér.
using System;                // Általános alapsztálykönyvtár típusok.
using System.Drawing;        // Grafikus renderelési típusok.
using System.Windows.Forms;  // Windows Forms GUI céleszköz típusok.
using System.Data;           // Általános adatközpontú típusok.
using System.Data.SqlClient; // MS SQL Server adathozzáférés
                             // típusok.
```

Ha egyszer már meghatároztunk néhány névtérrel (és hivatkoztunk az őket definiáló szerelvényekre), szabadon létrehozhatjuk a bennük található típusok példányait. Ha például a `Bitmap` osztály (amelyet a `System.Drawing` névtér definiál) egy példányát szeretnénk létrehozni, írjuk a következőket:

```
// A fájl által használt névterek listázása explicit módon.
using System;
using System.Drawing;

class Program
{
    public void DisplayLogo()
    {
        // 20 * 20 pixeles bittérkép létrehozása.
        Bitmap companyLogo = new Bitmap(20, 20);
        ...
    }
}
```



Mivel a kódfájlunk a `System.Drawing`-ra hivatkozik, a fordító ennek a névtérnek a tagjaként értelmezi a `Bitmap` osztályt. Ha nem határoztuk meg a `System.Drawing` névtérrel, fordítási hibához jutunk. Teljesen meghatározott névvel azonban szabadon deklarálhatunk változókat is:

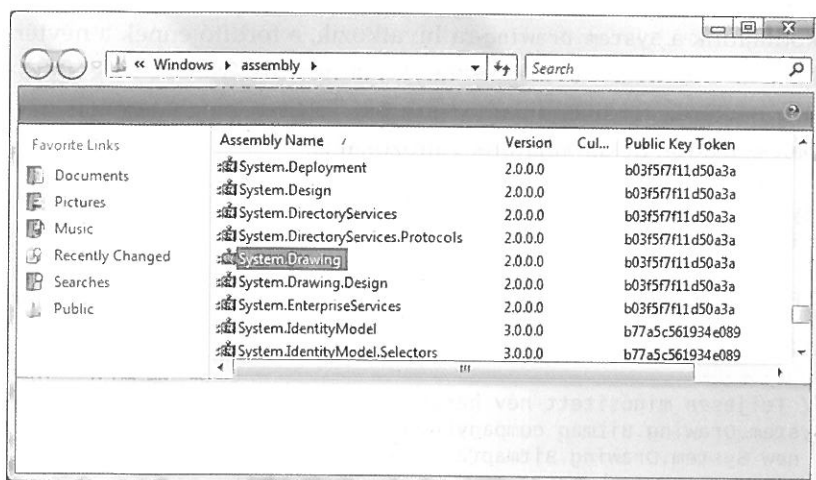
```
// A System.Drawing névtérrel nem listázzuk!  
using System;  
  
class Program  
{  
    public void DisplayLogo()  
    {  
        // Teljesen minősített név használata.  
        System.Drawing.Bitmap companyLogo =  
            new System.Drawing.Bitmap(20, 20);  
        ...  
    }  
}
```

Ha a típusdefiniálás teljesen minősített névvel történik, akkor tökéletesebb olvashatóságot biztosít, a C# `using` kulcsszava ezzel szemben lecsökkenti a billentyűleütések számát. A könyvben nem fogjuk a teljesen minősített neveket használni (kivéve, ha kétértelműséget kell feloldani), hanem a C# `using` kulcsszavának egyszerűsített szemléletét követjük.

Ne feledjük azonban, hogy a `using` kulcsszó a típus teljesen minősített nevének a rövidített jelölése, és mindkét szemlélet pontosan ugyanazt a mögöttes CIL-t eredményezi (ugyanis a CIL-kód mindig a teljesen minősített neveket használja), illetőleg nincs hatással a teljesítményre vagy a szerelvény méretére.

## Hivatkozás külső szerelvényekre

Amellett, hogy a C# `using` kulcsszavával megadjuk a névtérrel, a hivatkozott típus aktuális CIL-definícióját tartalmazó szerelvény nevét is meg kell adnunk a C#-fordítónak. Amint láttuk, sok alapvető .NET-névtér az `mscorlib.dll`-ben található. A `System.Drawing.Bitmap` típust azonban a `System.Drawing.dll` nevű különálló szerelvényben találjuk. A .NET-keretrendszer szerelvényeinek túlnyomó többsége a globális szerelvénytárban (GAC) található. Ez a könyvtár egy Windows-gépen alapértelmezés szerint a `C:\Windows\Assembly` elérési útvonalon érhető el, ahogy az 1.6. ábra mutatja.



1.6. ábra: A globális szerelvénytár (GAC) alapszámítógéptárai

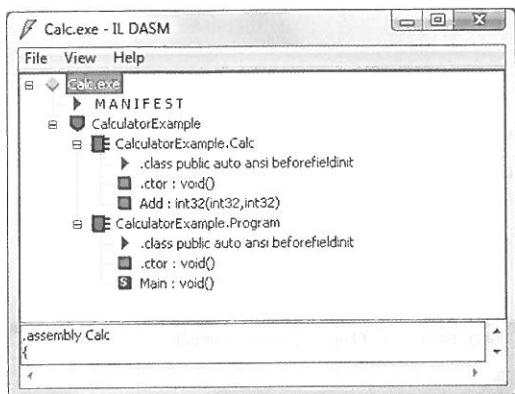
Attól függően, hogy milyen fejlesztőeszköz segítségével hozzuk létre a .NET-al-kalmazásokat, többféleképpen is közölhetjük a fordítóval, hogy mely szerelvényeket szeretnénk beemelni a fordítási ciklusba. Ennek a menetével a következő fejezetben ismerkedünk meg.

## Szerelvények feltárása az ildasm.exe segítségével

Nincs szükség arra, hogy a .NET-platform összes névterét ismerjük és kezeljük, hiszen egy névteret az tesz egyedivé, hogy szemantikailag kapcsolódó típusokat tartalmaz. Ezért, ha nincs szükségünk felhasználói felületre egy egyszerű konzolalkalmazáson kívül, el is felejtethetjük a `System.Windows.Forms`, `System.Windows` és a `System.Web` névtereket. Ha egy rajzoló alkalmazást hozunk létre, ahhoz az adatbázis-névtereknek valószínűleg nincs sok közük. Ahogy az előre gyártott kódkészletekkel általában lenni szokott, használat közben ismerjük meg őket.

A .NET Framework 3.5 SDK-val megjelent Intermediate Language Disassembler (`ildasm.exe`) lehetővé teszi bármilyen .NET-szerelvény betöltését és tartalmának vizsgálatát, köztük a hozzárendelt manifesztumot, a IL kódot és a típusmetaadatot. Az `ildasm.exe` betöltéséhez nyissuk meg a Visual Studio parancssort (Start > All Programs > Microsoft Visual Studio 2008 > Visual Studio Tools), írjuk be, `ildasm`, majd nyomjuk meg az Enter billentyűt.

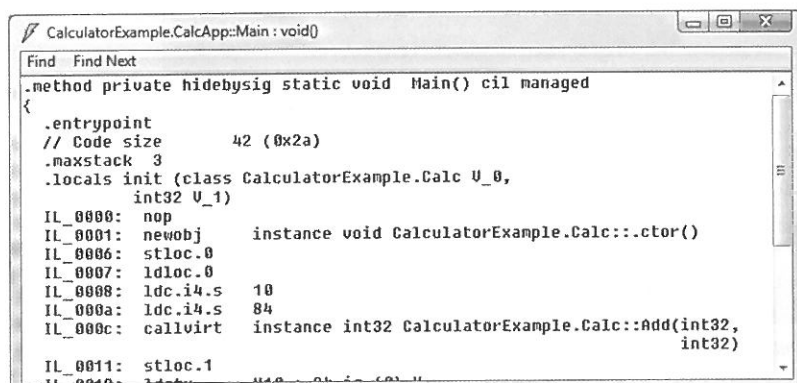
Ha már elindítottuk ezt az eszközt, csak nyissuk meg a File > Open menüpontot, és navigáljunk a vizsgálni kívánt szerelvényhez. Ennek szemléltetésére, nézzük meg a calc.exe szerelvényt, amelyet a fejezet korábbi részében bemutatott calc.csc fájl alapján generáltunk (lásd 1.7. ábra). Az ildasm.exe egy hasonló, fanézetű szerelvény szerkezetét mutatja.



1.7. ábra: Az ildasm.exe lehetővé teszi a .NET-szerelvény köztes nyelvi kódjának, manifesztumának és metaadatának megtekintését

## A köztes nyelvi kód megtekintése

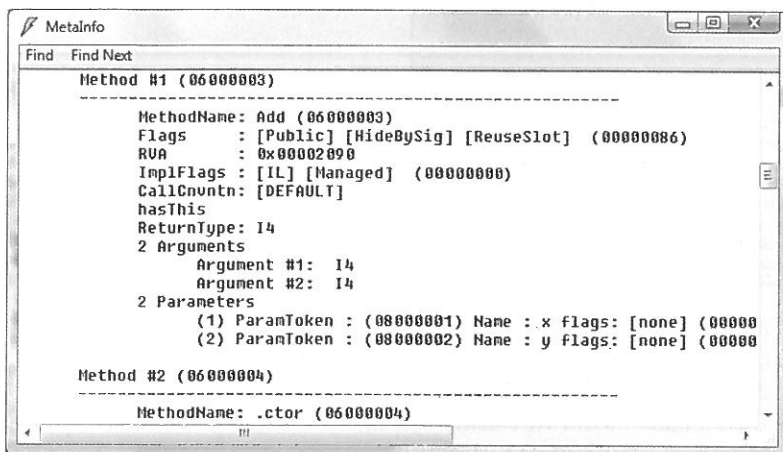
Amellett, hogy megmutatja az adott szerelvényben található névttereket, típusokat és tagokat, az ildasm.exe segítségével egy adott tag CIL-utasításait is megtekinthetjük. Ha például kétszer kattintunk a Program osztály Main() metódusára, egy új ablakban megjelenik a mögöttes köztes nyelvi kód (lásd 1.8. ábra).



1.8. ábra: A mögöttes köztes nyelvi kód megtekintése

## A típusmetaadat megtekintése

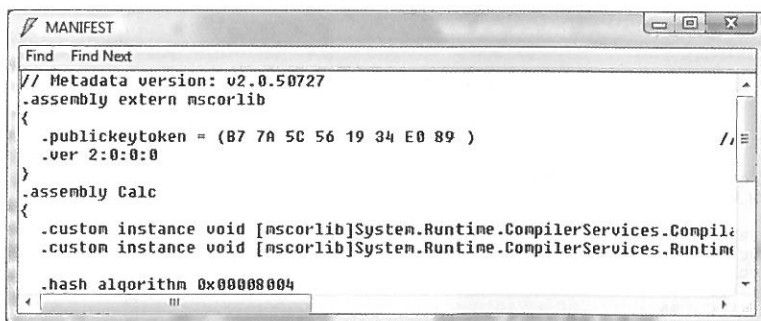
Az aktuálisan betöltött szerelvény típusmetaadatának megtekintéséhez, nyomjuk meg a Ctrl + M billentyűkombinációt. Az 1.9. ábrán a `calc.Add()` metódus metaadatát láthatjuk.



1.9. ábra: Típusmetaadat megtekintése az `ildasm.exe` segítségével

## Szerelvény metaadatának (manifesztumának) megtekintése

Végezetül, ha szeretnénk megtekinteni egy szerelvény manifesztumának tartalmát, kattintsunk kétszer a MANIFEST ikonra (lásd 1.10. ábra).



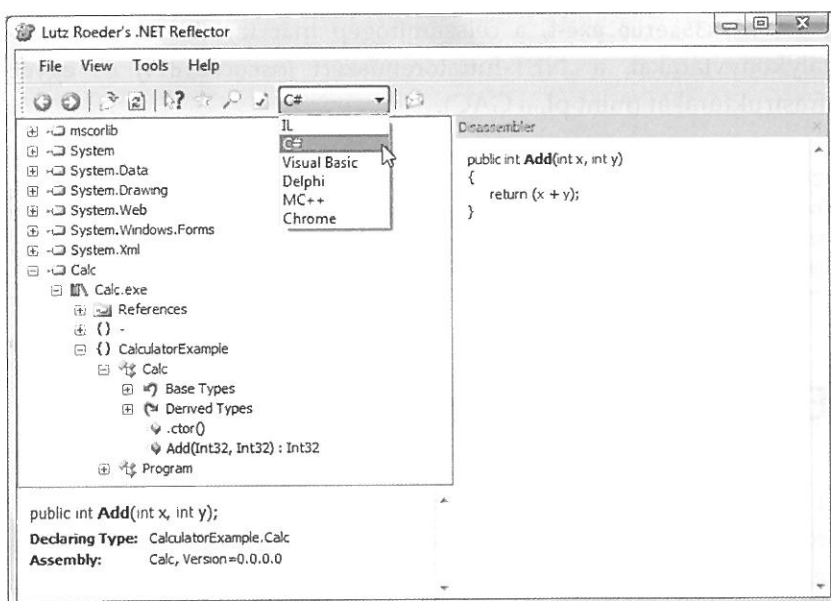
1.10. ábra: Önléíró adatok megtekintése az `ildasm.exe` segítségével

Természetesen az `ildasm.exe` többre is képes, mint amit itt bemutatunk, ezeket a megfelelő részeknél további funkciókkal is szemléltetjük.

## Szerelvények feltárása a Lutz Roeder's Reflector segítségével

Az ildasm.exe segítségével végzett kutatás a .NET-binárisban nagyon gyakori feladat, az egyetlen bökkenője, hogy csak a mögöttes köztes nyelvi kódot tekinthetjük meg, ahelyett, hogy egy szerelvény megvalósításába nézhetnénk bele az általunk kiválasztott nyelv segítségével. Szerencsére sok .NET-objektumböngésző, köztük a népszerű Reflector is letölthető.

Ezt az ingyenes eszközt a <http://www.aisto.com/roeder/dotnet> címről lehet letölteni. Kicsomagolás után futtathatjuk is az eszközt, és megadhatunk neki bármilyen szerelvényt, amelyet szeretnénk megnyitni a File > Open menüpont alatt. Az 1.11. ábra a calc.exe alkalmazást mutatja be újra.



1.11. ábra: A Reflector nagyon népszerű objektumböngésző eszköz

A reflector.exe egy Disassembler ablakot (a szóköz billentyű lenyomására nyílik meg) is támogat, valamint egy legördülő listát, amelyben a választott nyelv mögöttes alapkódbázisát tekinthetjük meg (benne természetesen a köztes nyelvi kódot).

Az eszköz további funkcióit könyvünk nem részletezi. A különféle elméletek szemléltetésére mind az ildasm.exe, mind a reflector.exe a segítségünkre lesz a továbbiakban.

## A .NET-futtatókörnyezet telepítése

Bizonyára nem meglepő, hogy a .NET-szerelvények csak olyan gépeken futtathatók, amelyekre telepítve van a .NET-keretrendszer. Annak, aki .NET-szoftvereket épít, ez nem jelenthet problémát, hiszen az ingyenesen elérhető .NET-keretrendszer 3.5 SDK verziójának (ugyanúgy, mint a kereskedelemben kapható .NET fejlesztői környezetek, mint például a Visual Studio 2008) telepítésével megfelelően konfiguráljuk a gépünket.

Ha azonban olyan gépre telepítünk szerelvényt, amelyen nincs .NET, a szerelvény nem fog tudni lefutni. Ezért a Microsoft kiadott egy `dotnetfx3-setup.exe` nevű telepítőcsomagot, amely ingyenesen letölthető és telepíthető a .NET-szoftverekkel együtt. Ez a telepítőprogram a Microsoft .NET-letöltő területéről (<http://msdn.microsoft.com/netframework>) elérhető. Ha telepítettük a `dotNetFx35setup.exe`-t, a célszámítógép már tartalmazza a .NET-alaposztálykönyvtárakat, a .NET-futtatórendszert (`mscorlib.dll`) és egyéb .NET-infrastruktúrákat (mint pl. a GAC).

---

**Megjegyzés** A Vista operációs rendszerbe előre konfigurálták a szükséges .NET-futtatókörnyezet infrastruktúráját. Ha azonban Windows XP-re vagy Windows Server 2003-ra telepítjük az alkalmazást, meg kell bizonyosodnunk arról, hogy telepítve van-e a célszámítógépre a .NET futási környezet.

---

## A .NET platformfüggetlen természete

A fejezet lezárásaként, röviden szólnunk kell a .NET platformfüggetlen természetéről. A .NET-szerelvények nem Microsoft operációs rendszereken is fejleszthetők és futtathatók (Mac OS X, számos Linux-disztribúció, Solaris, hogy csak egy párat említsünk). Ahhoz, hogy ezt megértsük, meg kell ismerünk a .NET-univerzum még egy hárombetűs rövidítését: a CLI-t (közös nyelvi infrastruktúra).

Amikor a Microsoft kiadta a C# programozási nyelvet és a .NET-platfomot, megalkottak egy formális dokumentumkészletet is, amely a C# és a köztes nyelv, a .NET-szerelvényformátum, az alapvető .NET-névterek és a feltételezett .NET-futtatómotor (ismertebb nevén, virtuális végrehajtó rendszer vagy VES) működésének szintaxisát és szemantikáját írja le.

Ezeket a dokumentumokat az ECMA-hoz is benyújtották (ahol jóváhagyták őket) hivatalos nemzetközi szabványként (<http://www.ecma-international.org>). Lényeges előírások:

- ECMA-334: The C# Language Specification
- ECMA-335: The Common Language Infrastructure (CLI)

Ezek a dokumentumok azért fontosak, mert lehetővé teszik a .NET-platform disztribúcióinak létrehozását külső gyártók számára az összes operációs rendszerre és/vagy processzorra. A két előírás közül talán az ECMA-335 a tartalmasabb, így azt több részre bontották, ahogy azt az 1.4. táblázatban láthatjuk.

| Az ECMA-335 részei    | Vaiós jelentése                                                                                                        |
|-----------------------|------------------------------------------------------------------------------------------------------------------------|
| I. rész: Architektúra | A CLI általános architektúráját írja le, benne a CTS és a CLS szabályait, valamint a .NET-futtatómotor működését.      |
| II. rész: Metaadat    | A .NET-metaadat részleteit írja le.                                                                                    |
| III. rész: CIL        | A CIL-kód szintaxisát és szemantikáját írja le.                                                                        |
| IV. rész: Könyvtárak  | Magas szintű áttekintést ad a .NET-disztribúció által kötelezően támogatandó minimális és teljes osztálykönyvtárakról. |
| V. rész: Mellékletek  | Olyan részletek gyűjteménye, mint az osztálykönyvtár-tervezési útmutató és a CIL-fordító megvalósítása.                |

#### 1.4. táblázat: A CLI részei

A IV. rész (Könyvtárak) csak *minimális* névtérkészletet definiál, ezek egy CLI-disztribúció alapvető szolgáltatásai (gyűjtemények, konzol I/O, fájl I/O, szálkezelés, reflexió, hálózati hozzáférés, alapvető biztonsági intézkedések, XML-kezelés stb.). A CLI *nem* definiál olyan névtereket, amelyek megkönnyítik a webes fejlesztést (ASP .NET), az adatbázis-hozzáférést (ADO .NET) vagy a grafikus felhasználói felületen (GUI) az alkalmazásfejlesztést (Windows Forms/Windows Presentation Foundation).

Jó hír azonban, hogy a .NET-disztribúciók fő irányvonala az ASP .NET, az ADO .NET és a Windows Forms Microsoft-kompatibilis ekvivalenseivel bővíti ki a CLI-könyvtárakat, hogy teljes körű, termékszerű fejlesztői platformokat kínáljon.

A CLI-nek a mai napig két fő megvalósítása létezik (a Microsoft Windows-specifikus kínálatán kívül). Habár könyvünk a Microsoft .NET-disztribúcióival létrehozott .NET-alkalmazásokra összpontosít, az 1.5. táblázatban a Mono és a Portable .NET-projektekről adunk tájékoztatást.

| Disztribúció                                                          | Valós jelentése                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="http://www.mono-project.com">http://www.mono-project.com</a> | A Mono projekt egy nyílt forráskódú CLI-disztribúció, amely különböző Linux-disztribúciókhoz (pl. SuSE, Fedora stb.), valamint Win32-höz és Mac OS X-hez készült.                                                                                        |
| <a href="http://www.dotgnu.org">http://www.dotgnu.org</a>             | A Portable .NET egy másik nyílt forráskódú CLI-disztribúció, amely számos operációs rendszeren fut. A Portable .NET a lehető legtöbb operációs rendszert célozza meg (Win32, AIX, BeOS, Mac OS X, Solaris, az összes fontosabb Linux-disztribúció stb.). |

**1.5. táblázat:** Nyílt forráskódú .NET-disztribúciók

Mind a Mono, mind a Portable .NET tartalmaz egy ECMA-kompatibilis C#-fordítót, .NET-futtatómotort, forráskódpéldákat, dokumentációkat, valamint számos olyan fejlesztőeszközt, amelyek működésük szempontjából egyenértékűek a Microsoft .NET keretrendszer 3.5 SDK verziójában találhatók. Továbbá, a Mono-ban és a Portable .NET-ben is megtalálható egy VB .NET-, egyJava- és egy C-fordítók.

**Megjegyzés** A Monót használó platformközi .NET-alkalmazások lefedése a II. kötetben helyet kapó B függelékben található.

## Összefoglalás

A fejezet célja az volt, hogy a könyv többi részének megértéséhez szükséges fogalmakat magyarázza. A .NET előtti technológiák korlátainak és bonyolultságának bemutatásával kezdtük, majd annak az áttekintésével folytattuk, hogyan próbálja meg a .NET és a C# leegyszerűsíteni a jelenlegi helyzetet.



A .NET lényege alapvetően egy futtatómotor (`mscorlib.dll`) és egy alapsztálykönyvtár (`mscorlib.dll`). A közös nyelvi futtatórendszer (CLR) képes hosztolni bármilyen .NET-binárist (szerelvényt), amely a felügyelt forráskód szabályainak megfelel. Ahogy azt láthattuk, a szerelvények CIL-utasításokat tartalmaznak (a típusmetaadattal és a szerelvény manifesztumával együtt), amelyek egy JIT-fordító segítségével platformfüggő utasításokká alakulnak. Ezután megvizsgáltuk a közös nyelvi specifikáció (CLS) és az egységes típusrendszer (CTS) szerepét.

Majd az `ildasm.exe` és a `reflector.exe` objektumböngésző segédprogramok bemutatása következett, valamint annak ismertetése, hogyan konfiguráljunk egy számítógépet a `dotnetfx3setup.exe` segítségével, hogy .NET-alkalmazásokat hosztoljon. A C# és a .NET-platform platformfüggetlen természetének rövid leírásával zárul ez a rész (ezt a leírást a B függelékben részletezzük).



## M Á S O D I K F E J E Z E T

# C#-alkalmazások fordítása

Egy C#-programozó több különböző eszköz közül választhat, ha .NET-alkalmazások készítésével foglalkozik. Ebben a fejezetben bemutatjuk a .NET-fejlesztés különféle lehetőségeit, többek között a Visual Studio 2008-at is. Először azonban megvizsgáljuk a C# parancssoros fordítót, a `csc.exe`-t, valamint a legegyszerűbb szövegszerkesztőt, a Jegyzettömböt, amely a Microsoft Windows operációs rendszer része.

Az „IDE-mentes” fordítás megismerése után néhány egyszerűbb szerkesztőt fogunk megismerni (mint például TextPad és Jegyzettömb++), amelyek lehetővé teszik C#-forráskódfájlok írását, valamint kissé kifinomultabban működnek együtt a fordítóval.

Noha a `csc.exe` és az egyik szövegszerkesztő ismerete elegendő lenne könyvünk tanulmányozásához, mégis érdemes megismerkedni a sokoldalú integrált fejlesztőkörnyezetekkel (IDE) is. Így bemutatjuk a SharpDevelop elnevezésű, ingyenes, nyílt forráskódú .NET-integrált fejlesztői környezetet is. A SharpDevelop felveszi a versenyt sok kereskedelmi forgalomban lévő .NET fejlesztői környezettel. Ezután a (szintén ingyenes) Visual C# 2008 integrált fejlesztői környezet rövid bemutatása következik, majd a Visual Studio 2008 legfontosabb szolgáltatásait nézzük végig. Végül áttekintést adunk több kiegészítő .NET-fejlesztőeszközzel (amelyek közül sok megint csak nyílt forráskódú) és beszerzési lehetőségeikről.

---

**Megjegyzés** A fejezetben több olyan C# programozási konstrukcióval szó lesz, amelyet itt formálisan nem vizsgálunk meg. A C# nyelv tanulmányozását formálisan a 3. fejezet kezdi meg.

---

## A .NET 3.5 keretrendszer fejlesztőkészletének szerepe

Gyakori tévhit a .NET-fejlesztéssel kapcsolatban, hogy meg kell venni a Visual Studio egy példányát C#-alkalmazások fejlesztéséhez. Ezzel szemben az az igazság, hogy bármilyen .NET-program létrehozható az ingyenesen letölt-

hető .NET Framework 3.5 szoftverfejlesztő készlet (SDK) segítségével. Ebben a szoftverfejlesztő készletben megtalálható több felügyelt fordító, parancssori segédprogram, fehér könyv, példaprogram, a .NET-osztálykönyvtárak és egy teljes dokumentációs rendszer is.

---

**Megjegyzés** A .NET Framework 3.5 SDK (dotNetFx35setup.exe) letölthető a .NET webhelyéről (<http://msdn.microsoft.com/netframework>).

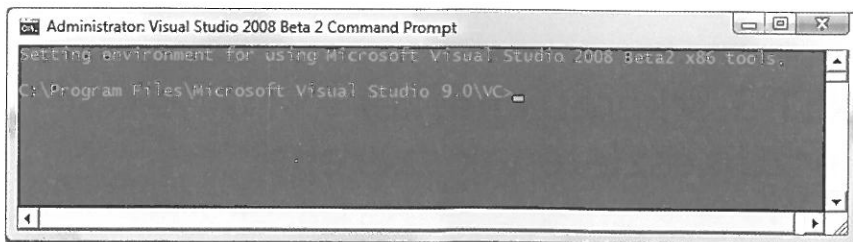
---

Ha viszont a Visual Studio 2008 vagy a Visual C# 2008 Express mellett döntünk, nem kell manuálisan telepíteni a .NET Framework 3.5 SDK fejlesztő-készletet. Bármelyik termék telepítésekor ugyanis az SDK automatikusan települ, így minden a rendelkezésünkre áll a csomagban, amire csak szükségünk lehet. Ugyanakkor ha *nem* Microsoft integrált fejlesztői környezetet szeretnénk használni a könyvbeli gyakorlatokban, ne felejtjük el a szoftverfejlesztő készletet telepíteni.

## A Visual Studio 2008 parancssor

Amikor a .NET Framework 3.5 SDK-t, a Visual Studio 2008-at vagy a Visual C# 2008 Expresszt telepítjük, egy sor olyan új könyvtár jön létre a helyi merevlemezen, amelyek mindegyike különböző .NET-fejlesztőeszközöket tartalmaz. Sok ilyen eszköz parancssorból kezelhető, így ha ezeket a segédprogramokat Windows Command Prompt ablakból szeretnénk használni, ehhez az útvonalakat regisztrálni kell az operációs rendszerben.

Bár manuálisan is módosíthatjuk a PATH változót, jelentős időt takaríthatunk meg azzal, ha egyszerűen a Start > Programok > Visual Studio 2008 > Visual Studio Tools mappából elérhető Visual Studio 2008 parancssort használjuk (lásd 2.1. ábra).



2.1. ábra: A Visual Studio 2008 parancssor

Ennek a parancssornak az az előnye, hogy előre úgy konfigurálták, hogy minden .NET-fejlesztőeszközhöz hozzáférést biztosítson. Ha már telepítettünk .NET-fejlesztőkörnyezetet, gépeljük be a következő parancsot, és nyomjuk meg az Enter billentyűt:

```
csc -?
```

Ekkor a C# parancssoros fordító parancssori argumentumainak listája jelenik meg (ahol a csc jelentése C#-fordító).

## C#-alkalmazások fordítása csc.exe használatával

Még ha nem is fogunk nagyszabású alkalmazást fordítani a C# parancssoros fordítóval, nagyon fontos, hogy tisztában legyünk a kódok kézi fordításának alapjaival. Miért is hasznos ezzel a folyamattal megismerkedni?

- Esetleg nincs Visual Studio 2008 programunk.
- Előfordulhat, hogy valamiért nem használhatunk kódgeneráló eszközöket vagy integrált fejlesztői környezeteket.
- Olyan automatikus fordítóeszközt szeretnénk alkalmazni, mint például az MSBuild vagy az NAnt, amelyekhez szükség van a használt eszközök parancssori opcióinak ismeretére.
- Egyszerűen szeretnénk elmélyíteni C#-ismereteinket. Ha grafikus IDE-k segítségével fordítjuk az alkalmazásokat, lényegében a csc.exe-t utasítjuk a C#-fájljainak manipulálására. Ekkor tisztában kell lennünk azal, hogy mi is zajlik a háttérben.

A másik előnye a csc.exe csupasz használatának, hogy így sokkal nagyobb biztonsággal kezelhetjük majd a .NET Framework 3.5 SDK egyéb parancssori eszközeit is. Ahogy azt a későbbiekben látni fogjuk, rengeteg fontos segédprogram csupán parancssorból érhető el.

A .NET-alkalmazások IDE-mentes fordításának illusztrálására a C# parancssoros fordító és a Jegyzetömb segítségével létrehozunk egy TestApp.exe nevű egyszerű futtatható alkalmazást.

Először is szükség lesz egy forráskódra. Nyissuk meg a Jegyzettömböt (a Start > Programok > Kellékek menüpont használatával), és írjuk be a következő triviális C#-forráskódot:

```
// Egyszerű C#-alkalmazás.
using System;

class TestApp
{
    static void Main()
    {
        Console.WriteLine("Testing! 1, 2, 3");
    }
}
```

Mentsük a fájlt megfelelő helyre (pl. C:\CscExample) TestApp.cs néven. A következőkben a C#-fordító kulcsfontosságú parancsaival ismerkedünk meg.

---

**Megjegyzés** A konvenció szerint minden C#-kódfájl kiterjesztése \*.cs. A fájl nevének nem kell semmilyen kapcsolatban lennie az általa definiált típussal (vagy típusokkal).

---

## Bemeneti és kimeneti célok megadása

Az első kérdés az, hogy hogyan adjuk meg a létrehozandó szerelvény nevét és típusát (pl. myShell.exe nevű parancssori alkalmazás, MathLib.dll nevű kódkönyvtár vagy Hal08.exe Windows Forms-alkalmazás stb.). Az egyes lehetőségeket egy, a csc.exe-nek parancssori paraméterként átadott speciális kapcsoló jelöli (lásd 2.1. táblázat).

| Opció           | Valós jelentése                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /out            | A létrehozandó szerelvény nevének megadására szolgál. Alapértelmezés szerint a szerelvény neve megegyezik az eredeti *.cs fájl nevével.                                |
| /target:exe     | Futtatható parancssori alkalmazást hoz létre. Ez az alapértelmezett szerelvénykimenet-típus, ebből következően az ilyen típusú alkalmazások létrehozásakor kihagyható. |
| /target:library | Egyfájlos *.dll szerelvényt előállító opció.                                                                                                                           |
| /target:module  | Modult állít elő. A modulok többfájlos szerelvények részei (ezeket részletesen a 15. fejezet mutatja be).                                                              |

| Opció          | Valós jelentése                                                                                                                                                                             |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /target:winexe | Bár a /target:exe opció segítségével szabadon hozhatunk létre grafikus felhasználói felületes alkalmazásokat, a /target:winexe megakadályozza, hogy a konzolablak a háttérben jelenjen meg. |

### 2.1. táblázat: A C#-fordító kimeneti opciói

**Megjegyzés** A parancssoros fordítónak (éppúgy, mint a legtöbb parancssoros eszköznek) küldött opciók előtagja vagy kötőjel (-?), vagy perjel (/?) lehet.

Ahhoz, hogy a TestApp.cs fájlt TestApp.exe nevű parancssori alkalmazássá fordítsuk, módosítsuk a forráskódfájlt tartalmazó könyvtárat:

```
cd C:\CscExample
```

Ezután írjuk be a következő parancsot (ügyelve arra, hogy a parancssori kapcsolóknak a bemeneti fájlok neve előtt kell szerepelniük, nem pedig mögöttük):

```
csc /target:exe TestApp.cs
```

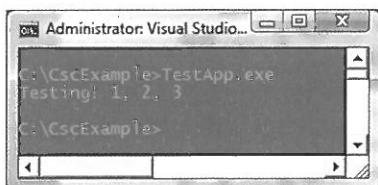
Itt nem adtunk meg explicit módon egy /out kapcsolót, így a végrehajtható fájl neve TestApp.exe lesz, minthogy a bemeneti fájl neve TestApp. Fontos az is, hogy a C#-fordító legtöbb kapcsolója rövidített formátumban is használható, például /t a /target helyett (az összes rövidítés megtekinthető, ha a parancssorba beírjuk a csc -? kifejezést):

```
csc /t:exe TestApp.cs
```

Továbbá, mivel a /t:exe a C#-fordító által használt alapértelmezett kimenet, a TestApp.cs egyszerűen a következő sor begépelésével is fordítható:

```
csc TestApp.cs
```

A TestApp.exe ekkor parancssorból futtatható, ahogy a 2.2. ábrán látható.



2.2. ábra: A TestApp.exe működés közben

## Hivatkozás külső szerelvényekre

A következőkben olyan alkalmazás fordítását vizsgáljuk meg, amely különálló .NET-szerelvényben definiált típusokat használ. Arról, hogy a C#-fordító honnan érti meg a `System.Console` típusra történő hivatkozást, az 1. fejezetben már szó volt: tudniillik a fordítási folyamat során a fordító *automatikusan hivatkozik* a `microsoft.dll`-re. (Ha valamiért ki akarjuk kapcsolni ezt a lehetőséget, akkor használhatjuk a `csc.exe /nostdlib` opcióját.)

Módosítsuk a `TestApp` alkalmazást úgy, hogy Windows Forms-üzenetdobozt jelenítsen meg. Nyissuk meg a `TestApp.cs` fájlt, és a következőképpen módosítsuk:

```
using System;

// Adjuk hozzá ezt!
using System.Windows.Forms;

class TestApp
{
    static void Main()
    {
        Console.WriteLine("Testing! 1, 2, 3");

        // Adjuk hozzá ezt!
        MessageBox.Show("Hello...");
    }
}
```

Ekkor a `System.Windows.Forms` névteret importáljuk a C# `using` kulcsszó segítségével (ezt az 1. fejezetben mutattuk be). Ahogy láttuk, amikor explicit módon felsoroljuk az adott \*.cs fájlban használt névtereket, nem kell a típusok teljesen meghatározott nevét alkalmaznunk (ez igencsak fárasztó munka lenne).

A parancssorban közölni kell a `csc.exe`-vel, hogy mely szerelvény tartalmazza a használt névtereket. Mivel a `System.Windows.Forms.MessageBox` osztályt használjuk, a `/reference` kapcsolóval (rövidítve: `/r`) meg kell adnunk a `System.Windows.Forms.dll` szerelvényt:

```
csc /r:System.Windows.Forms.dll TestApp.cs
```

Ha így futtatjuk az alkalmazást, a konzolos kimeneten kívül a 2.3. ábrán láthatóaknak is meg kell jelenniük.





2.3. ábra: Az első Windows Forms-alkalmazásunk

## Hivatkozás több külső szerelvényre

Ehhez kapcsolódóan érdekes felvetés, hogy mi történik, ha több külső szerelvényre kell hivatkoznunk a `csc.exe` használata során? Egyszerűen csak soroljuk fel az összes szerelvényt, pontosvesszővel elválasztva. Ugyan a mostani példánkhoz nem kell több külső szerelvényt megadni, azért bemutatunk egy mintát:

```
csc /r:System.Windows.Forms.dll;System.Drawing.dll *.cs
```

## Több forrásfájl fordítása

A `TestApp.exe` aktuális verziója egyetlen `*.cs` forráskódfájl segítségével jött létre. Bár teljes mértékben lehetséges, hogy minden .NET-típust egyetlen `*.cs` fájlban definiáljunk, a legtöbb projekt több `*.cs` fájl segítségével teszi rugalmasabbá a kódbázist. Tegyük fel, hogy létrehoztunk egy másik osztályt is, amely a `HelloMsg.cs` fájlban található:

```
// A HelloMessage osztály
using System;
using System.Windows.Forms;

class HelloMessage
{
    public void Speak()
    {
        MessageBox.Show("Hello...");
    }
}
```

Ezek után módosítsuk az eredeti `TestApp` osztályt, és használjuk az új osztálytípust, a korábbi Windows-űrlapos logikát pedig tegyük megjegyzésbe:

```
using System;

// Erre nincs már szükség:
// using System.Windows.Forms;

class TestApp
{
    static void Main()
    {
        Console.WriteLine("Testing! 1, 2, 3");

        // Már ez sem szükséges:
        // MessageBox.Show("Hello...");

        // Használjuk a HelloMessage osztályt!
        HelloMessage h = new HelloMessage();
        h.Speak();
    }
}
```

A C#-fájlokat lefordíthatjuk úgy, hogy egyszerűen explicit módon felsoroljuk az összes bemeneti fájlt:

```
csc /r:System.Windows.Forms.dll TestApp.cs HelloMsg.cs
```

Alternatív megoldásként a C#-fordító támogatja a (\*) helyettesítő karakter használatát, így a csc.exe a projektkönyvtárban található összes \*.cs fájlt beleveszi az aktuális felépítésbe:

```
csc /r:System.Windows.Forms.dll *.cs
```

Ha ismét lefuttatjuk a programot, a kimenet változatlan marad. Az egyetlen eltérés a két alkalmazás között az, hogy a program logikát több fájlban osztottuk szét.

## A C#-reakciófájlok használata

Ha parancssorban kellene összetett C#-alkalmazást írunk, igen jelentős számú bemeneti opciót kellene megadnunk ahhoz, hogy a fordító kezelni tudja a forráskódunkat. A gépelési teher csökkentése érdekében a C#-fordító támogatja a *reakciófájlokat*.

A C#-reakciófájl tartalmazza az aktuális fordításkor alkalmazandó instrukciókat. Konvenció szerint az ilyen fájlok kiterjesztése \*.rsp (response, reakció). Tegyük fel, hogy létrehoztunk egy TestApp.rsp nevű reakciófájlt, amely a következő opciókat tartalmazza (mint látható, a megjegyzéseket a # karakter jelzi):

```
# Ez a reakciófájl a 2. fejezet
# TestApp.exe példájához.
# Külső szerelvény referenciák.
/r:System.Windows.Forms.dll

# Kimenet és a fordítandó fájlok (helyettesítő karakterrel).
/target:exe /out:TestApp.exe *.cs
```

Tételezzük fel, hogy ezt a fájlt ugyanaz a könyvtár tárolja, mint a fordítandó C#-forráskódfájlokat, ekkor a teljes alkalmazás a következőképpen építhető fel (figyeljük meg a @ szimbólum használatát):

```
csc @TestApp.rsp
```

Ha szükséges, bemenetként több \*.rsp fájlt is megadhatunk (pl. csc @FirstFile.rsp @SecondFile.rsp @ThirdFile.rsp). Ehhez a megközelítéshez tudnunk kell, hogy a fordító az opciókat olyan sorrendben dolgozza fel, amilyenben megkapja. Éppen ezért egy későbbi \*.rsp fájl parancssori argumentumai felülírhatják egy korábbi reakciófájl opcióit.

Arra is ügyeljünk, hogy a parancssorban a reakciófájl előtt explicit módon megadott kapcsolókat felülírja az adott \*.rsp fájl. Így ha a következőt íránk:

```
csc /out:MyCoolApp.exe @TestApp.rsp,
```

a szerelvény neve még mindig TestApp.exe lenne (MyCoolApp.exe helyett), mivel a /out:TestApp.exe kapcsolót a TestApp.rsp reakciófájl tartalmazza. Ugyanakkor ha a kapcsolókat a reakciófájl után soroljuk fel, a kapcsoló felülírja a reakciófájl beállításait.

---

**Megjegyzés** A /reference kapcsoló hatása kumulatív. Függetlenül attól, hogy hol határozzuk meg a külső szerelvényeket (reakciófájl előtt, után vagy benne), a végeredmény az összes referenciaszerelvény összesítése.

---

Végül még azt kell tudnunk a reakciófájlokról, hogy a C#-fordítóhoz tartozik egy alapértelmezett reakciófájl (`csc.rsp`), amely ugyanabban a könyvtárban található, mint maga a `csc.exe` (amely alapértelmezés szerint a `C:\Windows\Microsoft.NET\Framework\v3.5` mappába települ). Ha a Jegyzetomb segítségével megnyitnánk ezt a fájlt, azt tapasztalnánk, hogy sok .NET-szerelvény van már a `/r:` kapcsoló segítségével megadva, többek között különböző könyvtárak webes fejlesztéshez, LINQ-hez, adateléréshez, valamint egyéb központi könyvtárak (az `mscorlib.dll` mellett).

Ha a `csc.exe` segítségével készítjük el C#-programjainkat, erre a reakciófájltra automatikusan történik hivatkozás akkor is, ha nem adunk meg egyedi `*.rsp` fájlt. Mivel létezik alapértelmezett reakciófájl, a jelenlegi alkalmazást a következő utasításkészlet segítségével sikeresen fordíthatjuk (mivel a `system.windows.forms.dll`-re történik hivatkozás a `csc.rsp` fájlban belül):

```
csc /out:TestApp.exe *.cs
```

Ha szeretnénk kikapcsolni a `csc.rsp` automatikus beolvasását, megadhatjuk a `/noconfig` opciót:

```
csc @TestApp.rsp /noconfig
```

---

**Megjegyzés** Ha olyan szerelvényekre hivatkozunk (az `/r` opció segítségével), amelyeket valójában nem használunk, akkor azt a fordító nem veszi figyelembe. Ezért nem kell a „kód-duzzasztól” tartani.

---

Nyilvánvaló, hogy a C# parancssoros fordító több más opciót is biztosít az előálló .NET-szerelvény generálásának a szabályozására. A `csc.exe` működéséről bővebb információ online a <http://msdn.microsoft.com> oldalon a „Working with the C# 2.0 Command Line Compiler” című cikkben olvasható. Noha ez a cikk a C# 2.0 fordító opcióit taglalja, a C# 2008 fordító is ugyanazt a szolgáltatáskészletet támogatja.

---

**Forráskód** A `CscExample` alkalmazás megtalálható a 2. fejezet alkönyvtárában.

---

## .NET-alkalmazás készítése TextPad használatával

Bár a Jegyzetomb megfelelő egyszerű .NET-programok létrehozására, a fejlesztői munka terén azonban nem nyújt semmit. Az lenne az ideális, ha a \*.cs fájlokat olyan szerkesztővel tudnánk megalkotni, amely (legalább) kulcsszószínezést és a C#-fordítóval való integráltságot biztosítja. Szerencsére létezik ilyen eszköz: a TextPad.

A TextPad olyan szerkesztő, amely sok programozási nyelven, többek között a C#-on támogatja a kódolást. Legfőbb előnye az, hogy nagyon egyszerűen használható, és csak éppen annyi plusz van benne, amely kényelmesebbé teszi a kódolást, de nem zavarja össze a tanulási folyamatot.

A TextPad legújabb verziója (a könyv írásakor 5.0.3) letölthető a <http://www.textpad.com> oldalról. Amint telepítettük, teljes szolgáltatású TextPad verziónk lesz; viszont ez az eszköz nem ingyenes, ugyanis be kell szereznünk hozzá egy egyfelhasználós licencet (kb. 30 USD-ért). Erre az alkalmazás minden futtatásakor „barátságos emlékeztető” figyelmeztet.

### A C#-kulcsszószínezés bekapcsolása

A TextPad alapállapotában nem ismeri a C#-kulcsszavakat, és nem tud együttműködni a csc.exe fordítóval sem. Ehhez a \*.cs fájlkiterjesztést kell regisztrálni az eszközben. Indítsuk el a TextPadet, és a New Document Wizard segítségével hajtsuk végre a következő feladatokat:

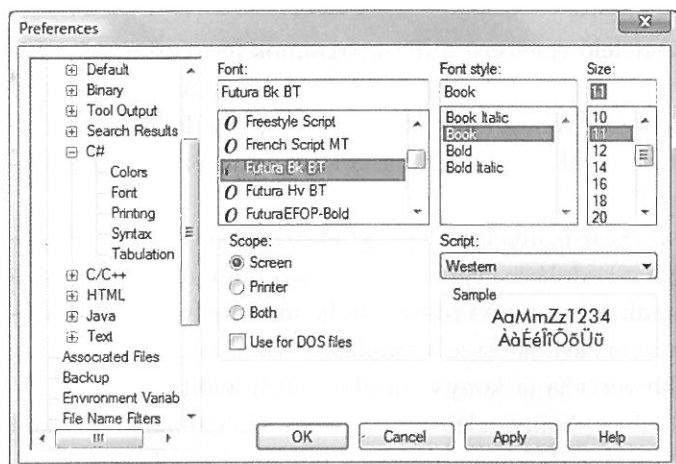
1. Válasszuk a Configure > New Document Class menüpontot.
2. Írjuk be a Document Class name szövegmezőbe a C# nevet.
3. A következő lépésben írjuk be a \*.cs kifejezést a Class members szövegmezőbe.
4. Végül engedélyezzük a szintaxiskiemelést, válasszuk a csharp.syn elemet a legördülő listamezőben, majd zárjuk be a varázslót.

---

**Megjegyzés** A TextPad korábbi verzióiban nem volt benne a C#-szintaxisfájl (csharp.syn), ám ez letölthető a TextPad webhelyéről. Innen egyébként sok különböző nyelvhez tölthető le szintaxisfájl.

---

Ezután a Configure > Preferences menüpontból elérhető Document Classes csomópont segítségével kicsit átalakíthatjuk a TextPad C#-os megjelenését és működését.



2.4. ábra: A TextPad C#-beállításainak megadása

## A \*.cs fájlzsűrő konfigurálása

A következő konfigurációs részlet a szűrő létrehozása lesz az Open és a Save párbeszédboxokban megjelenő C#-forráskódfájlokhoz.

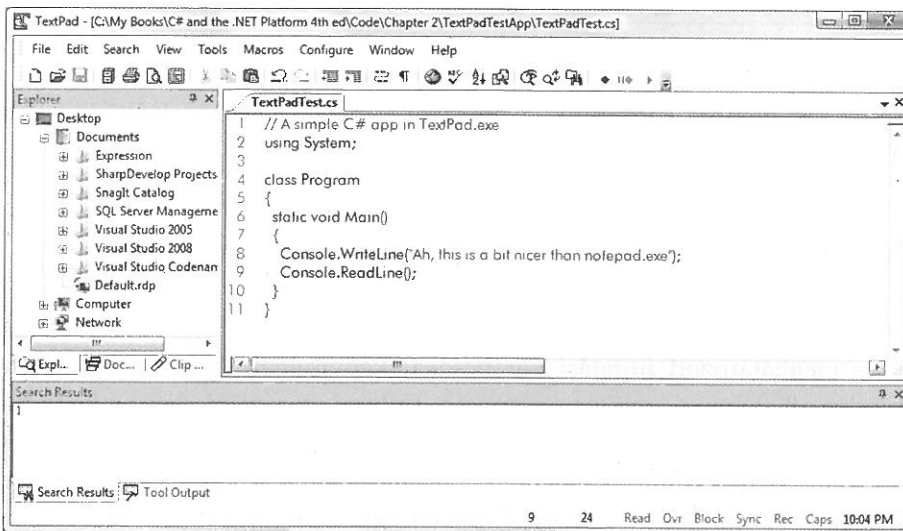
1. Válasszuk a Configure > Preferences menüpontot, majd a fa nézetben a File Name Filters elemet.
2. Kattintsunk a New gombra, majd írjuk be a C#-karaktereket a Description mezőbe, és a \*.cs kifejezést a wildcard szövegmezőbe.
3. A Move Up gomb segítségével helyezzük az új szűrőnket a lista élére, majd kattintsunk az OK gombra.

Hozzunk létre egy új fájlt (a File > New) segítségével, majd textpadtest.cs néven mentjük egy megfelelő helyre (pl. C:\TextPadTestApp). Ezután írunk be egy triviális osztálydefiníciót (lásd 2.5. ábra).

## A csc.exe integrálása

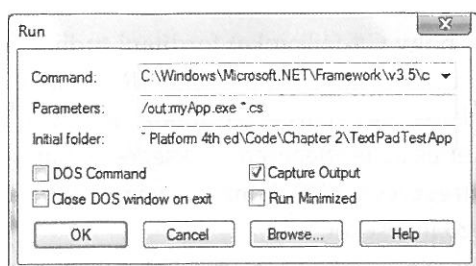
Az utolsó konfigurációs részlet ahhoz, hogy C#-fájljainkat fordítani tudjuk, a csc.exe betöltése lesz a TextPad programból. Elsőként a Tools > Run menüpontot kell használni. Ebben megjelenik egy párbeszéddoboz, amelyben megadhatjuk a futtatandó eszköz nevét és az esetlegesen szükséges parancssori kapcsolókat. Ahhoz, hogy TextPadTest.cs-t .NET-konzolos végrehajtható fájlá fordítsuk, végezzük el a következő lépéseket:

1. Írjuk be a csc.exe teljes elérési útvonalát a command szövegmezőbe (pl. C:\Windows\Microsoft.NET\Framework\v3.5\csc.exe).
2. Írjuk be a megadni kívánt parancssori opciókat a Parameters szövegmezőbe (pl. /out: myApp.exe \*.cs). Ne feledjük, hogy az egyszerűség kedvéért megadhatunk egyedi reakciófájl is (pl. @myInput.rsp).
3. Adjuk meg a bemeneti fájlokat tartalmazó könyvtárat az Initial Folder szövegmezőben (pl. C:\TextPadTestApp).
4. Ha azt szeretnénk, hogy a TextPad a fordító kimenetét közvetlenül (és nem egy különálló parancsablakban) kapja el, jelöljük be a capture output jelölőnégyzetet.



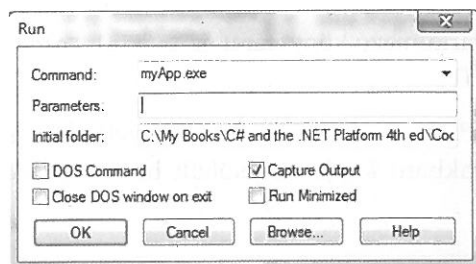
2.5. ábra: A TextPad működés közben

A 2.6. ábra néhány lehetséges fordítási beállítást ábrázol.



2.6. ábra: Egyedi Run parancs megadása

Ekkor a programunkat futtathatjuk úgy, hogy kétszer rákattintunk a végrehajtható fájlra a Windows Intézőben, vagy ismét használjuk a Tools > Run menüpontot, és aktuális parancsként megadjuk a myApp.exe fájlt (lásd a 2.7. ábrát).



2.7. ábra: A TextPad utasítása a myApp.exe futtatására

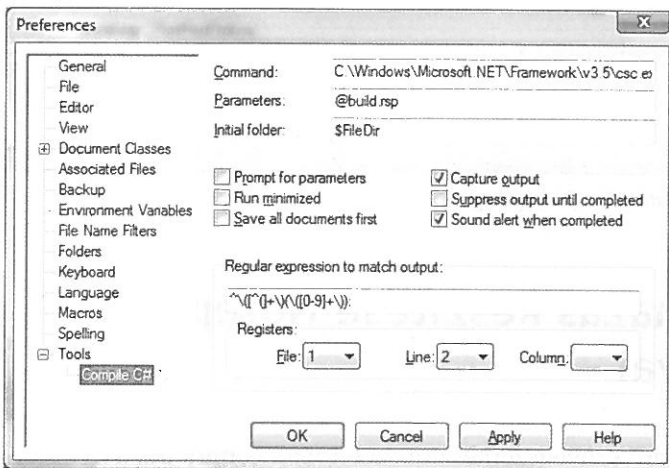
Ha az OK gombra kattintunk, a program kimenetének meg kell jelennie a kimeneti ablakban.

## Futtatási parancsok társítása menüelemekhez

A TextPad lehetőséget biztosít egyedi menüpontok létrehozására is, amelyek előre meghatározott futtatási parancsokat képviselnek. Így ahelyett, hogy minden alkalommal, amikor a parancsot futtatni szeretnénk, kézzel konfigurálnánk az eszközt, a paramétereket és az indulási mappát, lényegében csak egyszer tesszük ezt meg, és mentjük a beállításokat a későbbi felhasználásra. Hozzunk létre „C#-kód fordítása” néven egy olyan egyedi elemet a Tools menüben, amely az aktuális könyvtár minden C#-fájlját lefordítja a reakciófájl használatával.



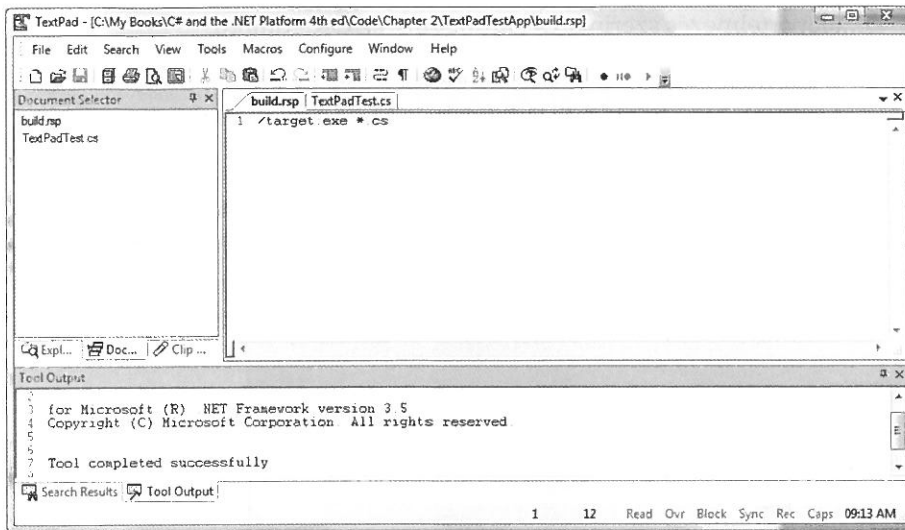
1. Válasszuk a **Configure > Preferences** menüpontot, majd a fa nézetben válasszuk a **Tools** elemet.
2. Az **Add** gomb segítségével válasszuk ki a **Program** elemet, majd a megjelenő párbeszédpanelben adjuk meg ismét a **csc.exe** teljes elérési útvonalát (ismét **C:\Windows\Microsoft.NET\Framework\v3.5\**, az alapértelmezés szerint), és kattintsunk az **OK** gombra.
3. Adhatunk az új menüelemünknek egy sokkal beszédesebb nevet (**C#-fordítás**) úgy, hogy a listadobozban kijelöljük az eszköz nevét (szerkesztésre).
4. Végül válasszuk ki a saját eszközünk nevét (**C#-fordítás**) a **Tools** csomópontban, és a **Parameters** mező egyetlen értékeként írjuk be a **@build.rsp** kifejezést; az **Initial Folder** mezőben a **\$FileDir** token utasítja a TextPadet, hogy nézze meg az aktív fájlt tartalmazó mappát (lásd 2.8. ábra).



2.8. ábra: Tools menüpont létrehozása

Így az egyedi **Tools** menüpontunk segítségével már minden **C#**-fájlt le tudunk fordítani az aktuális könyvtárban, feltéve, hogy ennek a könyvtárnak létezik egy **build.rsp** nevű **C#**-reakciófájlja. A 2.9. ábra azt mutatja, hogy a **Document Selector** panel segítségével megnézhetjük a TextPadben aktuálisan megnyitott fájlokat. A **Tool Output** ablak mutatja az egyedi **Tool** menüelemünk futtatásának kimenetét.

Ennyi információ a TextPadről elegendő ahhoz, hogy továbbléphessünk. Összetett szerkesztőről van tehát szó, amely számos további beépülő szolgáltatást támogat (helyesírás-ellenőrző, kódformázó, eszköztár, automatikus kiegészítés stb.). További részletek a <http://www.textpad.com/add-ons> oldalon olvashatók.



2.9. ábra: A C#-fordító futtatása TextPad segítségével

## .NET-alkalmazás készítése Notepad++ használatával

Végül még egy egyszerű szövegszerkesztőt szeretnénk bemutatni, a nyílt forráskódú (és ingyenesen letölthető) Notepad++-alkalmazást. Ezt az eszközt a <http://notepad-plus.sourceforge.net> oldalról lehet beszerezni, és a TextPad-hez hasonlóan több nyelven lehetővé teszi kódok írását, bekapcsolható vele a C#-fordító (a Run menü keresztül), és számos beépülő modul telepíthető hozzá. Ezen túlmenően a Notepad++ néhány egyéb érdekességet is rejt magában, például:

- a C#-kulcsszavak beépített támogatása;
- a *kódblokk egységbe zárása*, amely lehetővé teszi kódszegmenscsoportok kibontását és összecsukását a szerkesztőben (hasonlóan, mint a Visual Studio 2008/C# 2008 Express alkalmazásokban);