

ELSŐ FEJEZET

Üdvözljük a C# világában!

A fejezetből megtanulhatjuk, hogyan:

- használjuk a Visual Studio 2005 programozói környezetet,
- készítsünk C#-konzolalkalmazásokat,
- használjuk a névtereket,
- készítsünk C# Windows Forms alkalmazásokat.

A Microsoft Visual C# a Microsoft hatékony, komponens alapú programozási nyelve. Egyesek szerint a C#-nyelv szerepe a Microsoft .NET-keretrendszerben legalább annyira fontos, mint a C-nyelvé a UNIX fejlesztésében. Ha már járatosak vagyunk például C-, C++- vagy Java-programozásban, ismerős lesz a C# szintaxisa is, hiszen – azokhoz hasonlóan – a kódblokkokat kapcsos zárójellel határolja le. Ha másik nyelv használatához szoktunk, akkor sem okozhat gondot a C# szintaxisának elsajátítása. Csak arra kell ügyelnünk, hova tesszük a kapcsos zárójeleket és a pontosvesszőket. Remélem, hogy e könyv nagy segítségére lesz a Tisztelt Olvasónak.

Az I. részben a C# alapjaival foglalkozunk. Meglátjuk, hogyan deklarálhatunk változókat, és hogyan használhatjuk az operátorokat (többek között az + és a – jelet) értékek kiszámítására. Megtanulunk metódusokat írni, majd a metódusoknak paramétert átadni. Megismerjük az olyan kiválasztó utasításokat, mint például az *if*, valamint az *iteráló* utasításokat, mint például a *while*. Végül megismerjük a C# elegáns és felhasználóbarát hibakezelőjét. A II–VI. részben – a C# alapjainak elsajátítása után – bonyolultabb kérdésekkel foglalkozunk.

Ismerkedés a Visual Studio 2005 fejlesztői környezettel

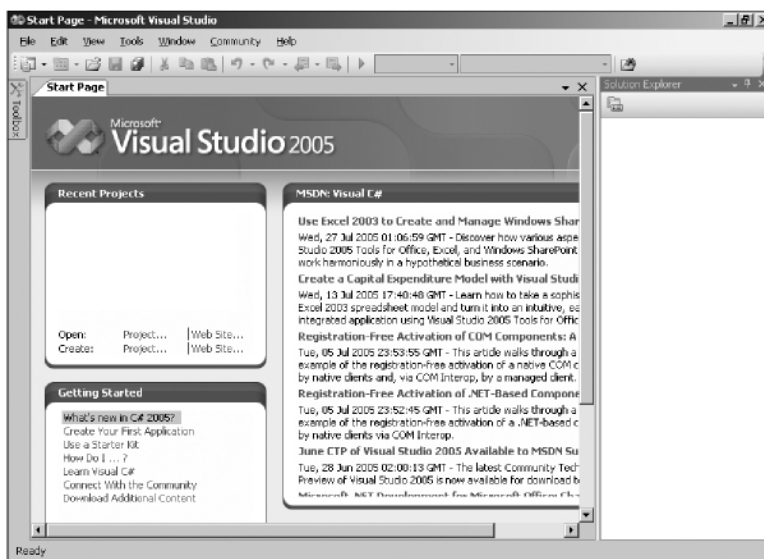
A Visual Studio 2005 fejlesztői környezetben minden szükséges eszközt és funkcionalitást megtalálunk, így akár nagyobb C#-projektek készítésére is alkalmas. Akár egyéb nyelvekben írt modulokat is gond nélkül használhatunk benne. Az első feladat során megtanulunk konzolalkalmazásokat írni a Visual Studio 2005 programozási környezetben.

Konzolalkalmazás létrehozása a Visual Studio 2005-ben

1. Indítsuk el a Microsoft Visual Studio 2005 programot.



Megjegyzés: A Visual Studio 2005 első futtatása során meg kell adnunk a fejlesztői környezet alapértelmezett beállításait. A Visual Studio 2005-ben beállíthatjuk a használni kívánt fejlesztői nyelvet. Az integrált fejlesztői környezet (IDE) párbeszéd- és eszközei az általunk választott nyelvhez igazodnak. A listából válasszuk a Visual C#-t, majd kattintsunk a Start Visual Studio (Visual Studio indítása) gombra. A Visual Studio 2005 integrált fejlesztői környezete hamarosan megjelenik.

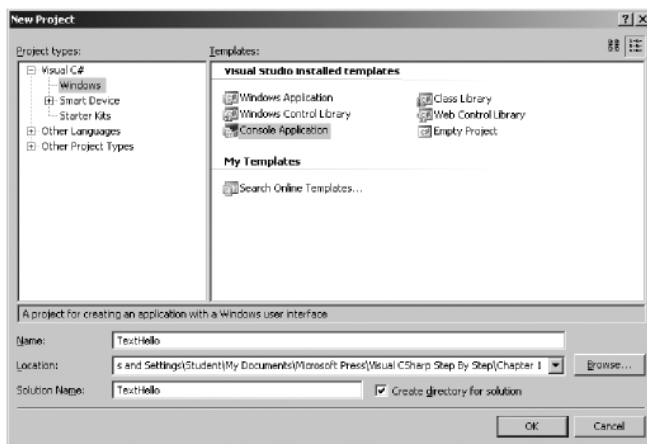


2. Kattintsunk a File/New/Project parancsra. A megnyíló New Project párbeszéd-
dobozban sablon segítségével meghatározhatjuk, milyen típusú alkalmazást szeretnénk készíteni. Többek között Windows-alkalmazások, osztálykönyvtárak és konzolalkalmazások közül választhatunk.



Megjegyzés: Az elérhető sablonok a Visual Studio 2005 verziójától függenek. Akár mi is definiálhatunk új projektsablonokat, ez azonban már túlmutat e könyv keretein.

3. Most a sablonok közül válasszuk a Console Application (konzolalkalmazás) ikont.
4. Elérési útnak a következőt adjuk meg: **C:\Documents and Settings\Nevem\My Documents\Microsoft Press\Visual CSharp Step by Step\Chapter 1** (a *Nevem* a Windows-ban használt felhasználónév). Az egyszerűség kedvéért a továbbiakban a „C:\Documents and Settings\Nevem\My Documents” mappára „\My Documents” könyvtárként fogunk hivatkozni.

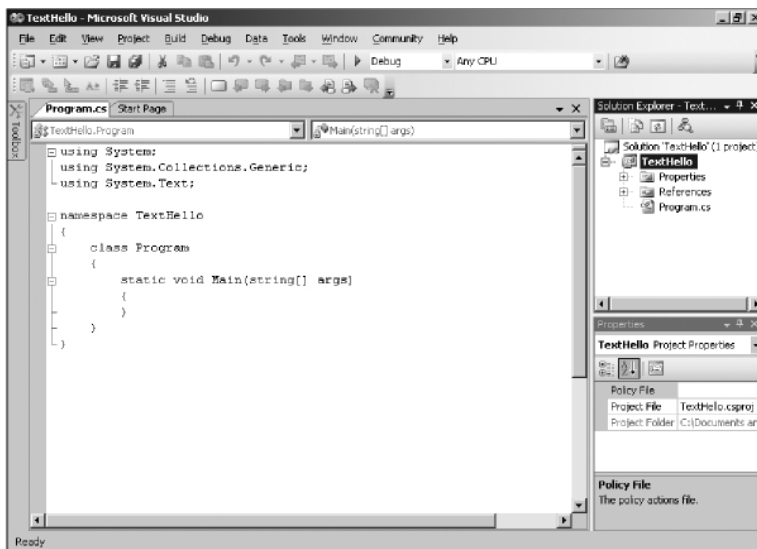


Megjegyzés: Ha a megadott könyvtár nem létezik, a Visual Studio 2005 létrehozza azt számunkra.



5. A Name mezőbe írjuk be: **TextHello**.
6. Pipáljuk ki a Create Directory for Solution jelölőnégyzetet, majd kattintsunk az OK gombra. Ekkor megjelenik az új projekt.

A képernyő tetején látható *menüsávval* elérhetjük a programozási környezet használatához szükséges funkciókat. Ahogy minden Windows-ban használt programban megszokhattuk, a menüt egérrel és billentyűzettel is elérhetjük. A menüsáv alatt található *eszköztár* gombjai megkönnyítik a leggyakrabban használt parancsok előhívását.



A képernyő legnagyobb részét a *Code and Text Editor ablak* foglalja el. Ebben láthatjuk a forrásfájl tartalmát. Többfájlos projekt esetén minden egyes forrásfájl saját *füllel* rendelkezik, melyen a forrásfájl neve látható. Ha valamelyik forrásfájlt meg szeretnénk nyitni a Code and Text Editor ablakban, kattintsunk az azonos nevű föltre. Más elemek mellett a *Solution Explorer* megjeleníti a projektben használt fájlok nevét. Ha a Code and Text Editor ablakban meg szeretnénk nyitni valamelyik fájlt, innen is megtehetjük: kattintsunk kétszer annak nevére.

Mielőtt hozzákezdénénk a kód írásához, a Solution Explorerben nézzük meg, milyen fájlokat hozott létre a Visual Studio 2005 a projekt részeként.

- **Solutions 'TextHello':** A legfelső szintű megoldásfájl, alkalmazásonként csak egy van belőle. Ha Windows Intézőben megnyitjuk a \My Documents\Visual CSharp Step by Step\Chapter 1\TextHello mappát, láthatjuk, hogy a fájl teljes neve TextHello.sln. Minden megoldásfájl hivatkozik legalább egy projektfájltra.
- **TextHello:** Ez a C#-projekt projektfájlja. Minden egyes projektfájl hivatkozásokat tartalmaz a forráskódot tartalmazó fájl(ok)ra és a projekthez kapcsolódó egyéb elemekre. Egy projekten belül valamennyi forráskódot ugyanazon a programozási nyelven kell írni. Windows Intézőben ez a fájl TextHello.csproj néven található a \My Documents\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello könyvtárban.
- **Properties:** A TextHello projekt egyik könyvtára. Megnyitása után találunk benne egy AssemblyInfo.cs nevű fájlt. Az AssemblyInfo.cs speciális fájl, mellyel beállíthatjuk a programunk *attribútumait*, többek között a szerző nevét, a létrehozás dátumát stb. Bizonyos attribútumokkal megszabhatjuk a program futásának módját, ezeket azonban nem tárgyaljuk a könyvben.
- **References:** Ebben a mappában található meg a programunk által használt, korábban lefordított kódokat. A kód fordításkor *szereplénné* alakul, és egyedi nevet kap. A hasznos kódkészleteket a fejlesztők szerelvényekbe csomagolják, így ezek más programozók számára is elérhetők. Alkalmazások fejlesztésekor gyakran használjuk majd a Microsoft Visual Studio 2005-ben található szerelvényeket.
- **Program.cs:** C#-forráskódot tartalmazó fájl, ezt látjuk a Code and Text Editor ablakban a projekt létrehozása után. Ebbe a fájlba írjuk a kódot. A Visual Studio 2005 automatikusan elhelyez néhány utasítást a fájlba, melyekről rövidesen szót ejtünk.

Az első program megírása

A Program.cs fájl definiál egy Program nevű osztályt, mely tartalmazza a *Main* nevű metódust. A metódusokat minden esetben osztályon belül kell definiálni. A *Main* metódus fontos, ez a program belépési pontja, csak statikus metódus lehet. [A metódusokkal bővebben a „Metódusok írása és hatáskör alkalmazása” című 3. fejezetben foglalkozunk. A statikus metódusukat a 7. fejezetben tárgyaljuk, melynek címe: „Osztályok és objektumok létrehozása, valamint kezelése”. A *Main* metódusról a 11. fejezetben („A paraméterek áttekintése”) lesz szó.]

Fontos: A C# kis- és nagybetűt megkülönböztető nyelv. A *Main* tehát nagy M-mel írandó!



A következő gyakorlatban a konzolon „Hello World” üzenetet megjelenítő kódot írunk, melyet aztán lefordítunk, és konzolalkalmazásként futtatunk. Végül megtanuljuk, hogyan oszthatjuk fel a kódelemeket névterekkel.

Kódírás IntelliSense-technológiával

1. A Program.cs fájlt megjelenítő Code and Text Editor ablakban vigyük a kurzort a *Main* metódust követő kapcsos zárójel után, majd írjuk be a következőt: **Console**. Ahogy beírjuk a *Console* szó C betűjét, IntelliSense-lista jelenik meg. A listában minden olyan C#-kulcsszó és -adattípus megjelenik, amely ebben a környezetben használható. Folytathatjuk a beírást, vagy kiválaszthatjuk a listából a Console elemet. Vagy ha a beírásban már a *Con*-ig eljutottunk, az IntelliSense-lista felajánlja a *Console* szót, melyet a Tab, az Enter vagy a szóköz billentyűk megnyomásával elfogadhatunk.

A *Main* a következőket kell hogy tartalmazza:

```
static void Main(string[] args)
{
    Console
}
```

Megjegyzés: A *Console* egy beépített osztály. Segítségével üzeneteket jeleníthetünk meg a képernyőn, és adatokat kérhetünk be a billentyűzetről.



2. Írjunk pontot közvetlenül a *Console* szó után. Ekkor még egy IntelliSense-lista jelenik meg, mely a *Console* osztály metódusait, tulajdonságait és mezőit tartalmazza.
3. A listából válasszuk ki a *WriteLine* sort, majd üssünk Entert. Vagy folytassuk a gépelést, amíg a *WriteLine* meg nem jelenik, majd akkor üssünk Entert.

Az IntelliSense-lista bezárul, a *WriteLine* metódus pedig automatikusan megjelenik a forrásfájlban. A *Main* most így néz ki:

```
static void Main(string[] args)
{
    Console.WriteLine
}
```

4. Nyissunk zárójelet. Ekkor újabb IntelliSense-javaslat jelenik meg. Ebben a *WriteLine* metódus paraméterei láthatók. A *WriteLine* valójában *túlterhelt metódus*, azaz a *Console* osztályon belül több *WriteLine* nevű metódus is található. A *WriteLine* metódus különböző verziói más és más típusú adatot adnak eredményül. (A túlterhelt metódusokról részletesen a 3. fejezetben szólunk.) A *Main* most így néz ki:

```
static void Main(string[] args)
{
    Console.WriteLine(
}
```

A javaslatok listáján a fel- és lefelé mutató nyilakra kattintva végigböngészheti a *WriteLine* túlterhelt verzióit.

5. Zárjuk be a zárójelet, majd írjunk pontosvesszőt. A *Main* most így néz ki:

```
static void Main(string[] args)
{
    Console.WriteLine();
}
```

6. Zárójelek közé írjuk be a következőt: „*Hello World*”. A *Main* most így néz ki:










```
static void Main(string[] args)
{
    Console.WriteLine("Hello world");
}
```



Tipp: Szokjunk hozzá, hogy azonnal beírjuk a karakterek párját is (például (után) vagy { után }), *mielőtt* még a köztes tartalmat begépelnénk. Könnyen megfélekedzhetünk a záró karakterről, ha begépelésüket a végére hagyjuk.

IntelliSense ikonok

Az IntelliSense megjeleníti az osztály minden tagjának a nevét. Minden tag nevéből balra egy ikon mutatja annak típusát. Az ikonok és azok típusát a következő táblázat foglalja össze:

Ikron	Jelentés
	C# kulcsszó
	metódus (a 3. fejezetben tárgyaljuk)
	tulajdonság (a 14. fejezetben tárgyaljuk)
	osztály (a 7. fejezetben tárgyaljuk)
	struktúra (a 9. fejezetben tárgyaljuk)
	felsorolás (a 9. fejezetben tárgyaljuk)
	interfész (a 12. fejezetben tárgyaljuk)
	delegate (a 16. fejezetben tárgyaljuk)
	namespace

Megjegyzés: A forráskódban gyakran találkozunk majd két törtvonal után folyószöveggel. Ezek kommentek. A fordítóprogram figyelmen kívül hagyja a kommenteket, de a fejlesztők számára rendkívül hasznosak, hiszen leírják, hogy mit is csinál a program. Például:



```
Console.ReadLine(); // wait for the user to press the Enter key
```

A fordítóprogram a két törtvonal után minden szöveget figyelmen kívül hagy egészen a sor végéig. Akár többsoros kommenteket is beszúrhatunk a kódba a /* jel után. A fordító mindent figyelmen kívül hagy a következő */ jelig, így akár többsornyi kommentet is beszúrhatunk. Érdemes a kódot megfelelően dokumentálni kommentek beszúrásával.

A konzolalkalmazás fordítása és futtatása

1. A Build menüben kattintsunk a Build Solution parancsra. Ennek hatására a Visual Studio lefordítja a C#-kódot, és készít egy futtatható programot. A Code and Text Editor ablak alatt megjelenik a kimeneti ablak.

Tipp: Ha a kimeneti ablak nem jelenne meg, a megjelenítéshez kattintsunk a View menü Output parancsára.



A kimeneti ablakban a következőhöz hasonló üzenetek jelennek meg, melynek segítségével nyomon követhetjük a program fordítását és az esetleges hibák részleteit. Jelen esetben a programnak hibák és figyelmeztetések nélkül kell lefordulnia:

Üdvözljük a C# világában!

```
----- Build started: Project: TextHello, Configuration: Debug Any CPU -----  
Csc.exe /config /nowarn:"1701;1702" /errorreport: prompt /warn:4 ...
```

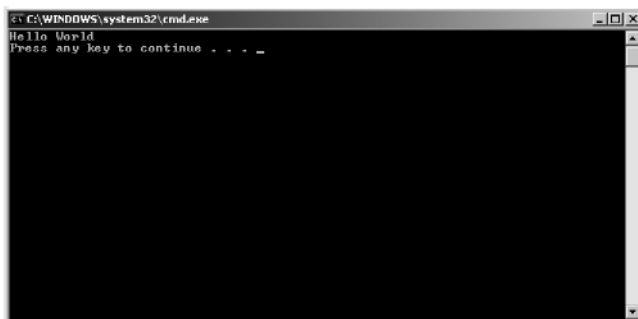
```
Compile complete -- 0 errors, 0 warnings
```

```
TextHello -> C:\Documents and Settings\John\My Documents\Microsoft Press\...  
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
```



Megjegyzés: Ha utolsó mentés óta módosítottunk a fájlban, a fájl neve után csillag karakter jelenik meg a Code and Text Editor ablak feletti fülben. Nem szükséges fordítás előtt külön elmenteni a fájlt, hiszen a Build Solution parancs automatikusan megteszi ezt helyettünk.

2. A Debug menüben kattintsunk a Start Without Debugging menüpontra. Ekkor megjelenik a parancssori ablak, és a program elindul. Ahogy a következő képen látható, a program kiírja, hogy „Hello World”, majd a felhasználóra vár, hogy nyomjon meg egy billentyűt.



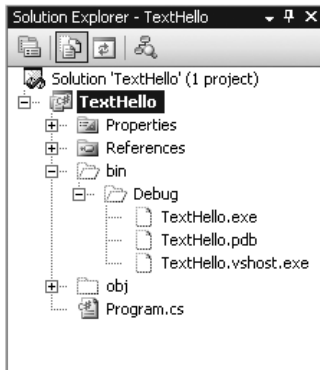
3. Győződjünk meg róla, hogy a parancssori ablak van előtérben, majd üssünk Entert. A parancssori ablak bezárul, és visszatérünk a Visual Studio 2005 programozói környezetbe.



Megjegyzés: Ha a Debug menü Start Debugging parancsával indítjuk el az alkalmazást, az ablak az üzenet kiírása után azonnal bezárul, anélkül, hogy bármelyik gombot megnyomnánk.

4. A Solution Explorerben kattintsunk a TextHello projektre (ne a megoldásra), majd válasszuk a Show All Files gombot. A C#-forrásfájl felett megjelenik két bejegyzés bin és obj néven. Ez a két bejegyzés a projekt mappáján belül (\MyDocuments\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello) két ilyen nevű mappára utal. Ezek a könyvtárak a program fordításakor jönnek létre, és az alkalmazás végrehajtható verzióját, valamint egyéb fájlokat tartalmaznak.
5. A Solution Explorerben kattintsunk a bin bejegyzés mellett található + jelre. Ekkor egy Debug nevű könyvtár is megjelenik.

6. A Solution Explorerben kattintsunk a Debug bejegyzés mellett látható + jelre. Három bejegyzés fog megjelenni: TextHello.exe, TextHello.pdb, valamint TextHello.vshost.exe. A TextHello.exe a lefordított program. Ez indul el, ha a Debug menü Start Without Debugging parancsára kattintunk. A másik két fájlt a Visual Studio akkor használja, ha hibakereső üzemmódban indítjuk el a programot (azaz ha a Debug menü Start Debugging parancsára kattintunk).



Parancssori fordítás

A forrásfájlokat a csc parancssori C#-fordítóval parancssorban is lefordíthatjuk. A környezet beállításához azonban először végére kell hajtani a következő lépéseket:

1. Indítsuk el a Visual Studio 2005 Command Prompt programot. Ekkor egy parancssori ablak jelenik meg. A *PATH*, a *LIB*, valamint az *INCLUDE* környezeti változók ilyenkor úgy vannak beállítva, hogy tartalmazzák a .NET-keretrendszer könyvtárait és segédeszközeit.

Tipp: Hagyományos parancssori környezetben is beállíthatjuk ezeket a környezeti változókat: futtassuk a C:\Program Files\Microsoft Visual Studio 8\VC mappában található vcarsall.bat parancsfájlt.



2. A Visual Studio 2005 Command Prompt ablakában írjuk be a következő parancsot, hogy a \MyDocuments\Microsoft Press\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello könyvárba lépünk:

```
cd \Documents and Settings\YourName\My Documents\Microsoft
Press\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello
```

3. Itt adjuk ki a következő parancsot:

```
csc /out:TextHello.exe Program.cs
```

Ezzel a paranccsal létrehozunk a TextHello.exe végrehajtható fájlt a C#-forrásfájlból. Ha kihagyjuk a */out* parancssori paraméteret, a végrehajtható fájl a forrásfájl nevét fogja felvenni, azaz Program.exe lesz.

4. A következő paranccsal futtathatjuk a programot:

```
TextHello
```

A program az előzőekben látott módon fog futni, azzal a kivétellel, hogy nem jelenik meg az „A folytatáshoz nyomjon meg egy gombot” üzenet.

Névterek használata

Az előbbi példában végtelenül egyszerű programot írtunk. A kis programokból azonban szempillantások alatt nagyobb programot varázsolhatunk. Ahogy programunk egyre nagyobb lesz, két problémával is szembesülhetünk: először is minél több kód van a programban, annál nehezebb megérteni és kézben tartani. Másodszor pedig a hosszabb program általában több névvel is jár: több változó, több metódus és több osztály. A nevek számának növekedésével egyre valószínűbb, hogy fordítási hiba fog fellépni a programban használt nevek ütközése miatt (különösen, ha külső gyártótól származó könyvtárakat használunk).

Korábban a programozók úgy próbálták elkerülni a nevek összeütközését, hogy a használt nevek elé valamilyen minősítőt vagy minősítőket tettek. Ez nem túl szerencsés megoldás, hiszen nem bővíthető. A nevek hossza egyre csak nő, és a programozás helyett egyre csak a neveket gépeljük és bogarásszuk ki a korábbi sorokból.

A névterekkel megoldható ez a probléma: az azonosítóknak, mint például az osztályoknak, névvel ellátott tárolót hozunk létre. Két azonos nevű osztály nem fog összeütközni, hacsak nem egy névtéren belül vannak. A *TextHello* nevű névtéren belül az alábbi utasítással hozhatunk létre egy *Greeting* nevű osztályt:

```
namespace TextHello
{
    class Greeting
    {
        ...
    }
}
```

A *Greeting* osztályra a *TextHello.Greeting* kifejezéssel hivatkozhatunk a programjainkban. Ha egy másik programban is található *Greeting* osztály, de az másik névtérben van, a programunk gond nélkül fog futni, hisz a *TextHello.Greeting* osztályt használja. Ha a másik *Greeting* osztályt szeretnénk használni, meg kell határozni, hogy a másik névtér osztályát fogjuk alkalmazni.

Ajánlott minden osztályt névtérben definiálni. A Visual Studio 2005 programozói környezet úgy tesz eleget ennek a követelménynek, hogy a projekt nevét állítja be legfelső szintű névtérnek. A .NET-keretrendszer szoftverfejlesztési készlet (SDK) szintén eleget tesz ennek a követelménynek: a .NET-keretrendszer minden osztálya névtéren belül létezik. A *Console* osztály például a *System* névtéren belül található. Ezek szerint tehát az osztály teljes neve *System.Console*.

Persze ha minden alkalommal meg kéne adni az osztály teljes nevét, semmivel sem lennének előrébb, mintha az osztályt egyszerűen *SystemConsole*-nak hívnánk. Szerencsére a *using* direktívával megoldható a probléma. Ha visszatérünk a *TextHello* programhoz a Visual Studio 2005-ben, és alaposabban megnézzük a *Program.cs* fájlt a Code and Text Editor ablakban, a következőket láthatjuk:

```
using System;
using System.Collections.Generic;
using System.Text;
```

A *using* utasítással meghatározhatunk egy névteret, így ezután nem kell névtérrel együtt megadni az ahhoz a névtérhez tartozó objektumokat. A fenti három névtér osztályait olyan gyakran használják, hogy a Visual Studio 2005 automatikusan hozzáadja minden új projekthez ezeket a *using* utasításokat. A forrásfájl elejére további *using* direktívákat szűrhatunk be.

A következő feladatok a névterek további használatát mutatják be.

Teljes nevek

1. A Code and Text Editor ablakban tegyük megjegyzés jelek közé a *Program.cs* elején található *using* direktívát.

```
//using System;
```

2. A Build menüben kattintsunk a Build Solution parancsra. A fordítás sikertelen, az Output ablakban pedig a következő hibaüzenet jelenik meg kétszer (valahányszor a *Console* osztályra hivatkozunk):

```
The name 'Console' does not exist in the current context.
```

3. Az Output ablakban kattintsunk kétszer a hibaüzenetre. A hibát okozó azonosítót a program kijelöli a *Program.cs* forrásfájlban.



Tipp: Az első hiba befolyásolhatja az azt követő diagnosztikai üzenetek megbízhatóságát. Ha a fordítás során több diagnosztikai üzenet is megjelenik, először csak az elsőt javítsuk ki, a többit hagyjuk figyelmen kívül, majd fordítsuk újra a programot. Ez a stratégia akkor a leghasznosabb, ha rövid forrásfájlokkal, iteratív módon dolgozunk, és gyakran lefordítjuk a programot.

4. A Code and Text Editor ablakban írjuk át a *Main* metódus Console osztályait a teljes nevükre, azaz *System.Console*-ra.

A *Main* a következőket kell hogy tartalmazza:

```
static void Main(string[] args)
{
    System.Console.WriteLine("Hello world");
}
```



Megjegyzés: A *System* gépelése közben figyeljük meg, ahogy az IntelliSense a listában megjeleníti a *System* névtér összes elemét.

5. A Build menüben kattintsunk a Build Solution parancsra. Ezúttal sikeresen lefordul a program. Ha mégsem így történne, győződjünk meg róla, hogy a *Main* pontosan úgy néz ki, ahogy az előbbi kódban, majd fordítsuk újra a programot.
6. Hogy meggyőződjünk arról, hogy az alkalmazás még mindig működik, futtassuk le a Debug menü Start Without Debugging parancsával.

A Solution Explorerben kattintsunk a References bejegyzés mellett látható + jelre. Ekkor megjelenik a Solution Explorer által használt szerelvények listája. Szerelvénynek más programozók által írt kódot nevezünk (ilyen a Microsoft .NET-keretrendszer is). Bizonyos esetekben a névtérben használt osztályok a névtérrel azonos nevű szerelvényben található (például a *System* esetében), azonban ez nem mindig van így: néhány szerelvény akár több névtérrel is tartalmazhat. Ha névtérrel használunk, minden esetben hivatkoznunk kell a névtér osztályait tartalmazó szerelvényekre, különben a program nem fordul le (vagy nem indul el).

Windows Forms alkalmazás készítése

Az előző példában alapszintű parancssori alkalmazást írtunk a Visual Studio 2005-ben. Azonban semmi sem akadályoz meg minket abban, hogy grafikus Windows-alkalmazást írjunk a Visual Studio 2005 programozói környezetben. A grafikus tervező segítségével interaktív módon tervezhetjük meg Windows-alkalmazásunk felhasználói felületét. A Visual Studio 2005 ezután előállítja a felhasználói felületünkhöz tartozó programutasításokat.

Ebből következik, hogy a Visual Studio 2005 környezetben kétféleképpen tervezhetjük programjainkat: tervezési nézetben, valamint kódnézetben. A Code and Text Editor ablakban (amelyben a programutasításokat láthatjuk), valamint a Design View ablakban (amelyben a felhasználói felületet tervezhetjük meg). A kettő között bármikor átválthatunk.

A következő gyakorlatokban megtanuljuk, hogyan készíthetünk Windows-programokat Visual Studio 2005-ben. Az első programunk egyetlen egyszerű űrlapból fog állni. Az űrlapon elhelyezünk egy szövegmezőt és egy gombot. Ha a szövegmezőbe beírjuk a nevünket, és a gombra kattintunk, személyes üdvözlő üzenet jelenik meg. Első lépésként a grafikus tervező segítségével elhelyezzük a szükséges vezérlőelemeket a megjelenő űrlapra, azaz megtervezzük a felhasználói felületet. Ezután megfigyeljük, hogy ennek hatására milyen kódot hozott létre a Visual Studio. Ezt követően a grafikus tervező használatával megváltoztatjuk a vezérlőelemek tulajdonságait, és átméretezzük az űrlapot. Végül megírjuk a gombnyomásra futtatandó kódot, és elindítjuk első Windows-programunkat.

Windows projekt létrehozása Visual Studio 2005-ben

1. Kattintsunk a File/New/Project parancsra.
2. A megjelenő New Project párbeszédobozban válasszuk ki a Visual C#-t.
3. Ezután a sablonok közül válasszuk a Windows Application (Windows-alkalmazás) ikont.
4. Adjuk meg elérési útnak a következőt: **\MyDocuments\Microsoft Press\Visual CSharp Step by Step\Chapter 1.**
5. A Name mezőbe írjuk be: **WinFormHello.**
6. Pipáljuk ki a **Create new Solution** jelölőnégyzetet. Ezzel Windows-alkalmazásunk tárolására létrehozunk egy új megoldást. A másik választási lehetőség az **Add to Solution** jelölőnégyzet, mellyel projektünk a TextHello megoldásához kerülne.
7. Kattintsunk az OK gombra. A Visual Studio 2005 bezárja nyitva lévő alkalmazásunkat (ha szükséges, felajánlja a mentés lehetőségét), majd létrehoz egy üres Windows-űrlapot a Design View ablakban.

A következő feladatban megtanulunk vezérlőelemeket helyezni Windows-űrlapunkra a grafikus tervezővel, majd megfigyeljük, hogy ennek hatására milyen C#-kódot hozott létre a Visual Studio.

A felhasználói felület megtervezése

1. Tervezési nézetben kattintsunk az űrlap mellett baloldalt megjelenő Toolbox földre. Ekkor megjelenik az eszköztár, mely részben eltakarja az űrlapot. Az eszköztáron találjuk meg az űrlapra helyezhető vezérlőelemeket és egyéb komponenseket.
2. Az eszköztáron kattintsunk a Common Controls mellett látható + jelre, hogy a Windows Forms alkalmazások által leggyakrabban használt vezérlőelemeket megjelenítsük.
3. Válasszuk a Label-t (címké), majd kattintsunk az űrlap látható részére. Ennek hatására *címke*-vezérlőelem jelenik meg az űrlapon, és az eszköztár eltűnik.



Tip: Ha nem szeretnénk, hogy az eszköztár eltűnjön, de azt sem, hogy emiatt ne lássuk az űrlap egy részét, kattintsunk az Auto Hide gombra az eszköztár fejlécének jobb oldalán (a gomb egy gombostűre hasonlít). Az eszköztár ilyenkor a Visual Studio 2005 bal oldalán jelenik meg, ezért a tervezési nézet mérete is lecsökken, hogy baloldalt elérjen az eszköztár. (Alacsony felbontású képernyő esetén nagy helyet foglalhat el az eszköztár.) Ha ismét az Auto Hide gombra kattintunk, az eszköztár eltűnik.

4. Az űrlapra elhelyezett *címke*-vezérlőelem valószínűleg nem a megfelelő helyre került. Ezt azonban könnyen kiküszöbölhetjük: az űrlapon elhelyezett vezérlőelemeket az egérrel egyszerűen áthelyezhetjük. Az egérrel helyezzük a *címke*-vezérlőelemet az űrlap bal felső sarkába. (A vezérlőelem pontos elhelyezése nem befolyásolja az alkalmazás működését.)
5. Kattintsunk a View menü Tulajdonságok menüpontjára. A képernyő jobb oldalán ekkor megjelenik a tulajdonságablak. A tulajdonságablakban a projekt elemeinek tulajdonságait állíthatjuk be. Környezetfüggő, azaz az éppen kijelölt elem tulajdonságait mutatja. Ha az űrlapon bárhova kattintunk, láthatjuk, hogy a tulajdonságablakban magának az űrlapnak a tulajdonságai jelennek meg. A *címke*-vezérlőelemre kattintva pedig a címke tulajdonságai láthatók az ablakban.
6. Most kattintsunk a *címke*-vezérlőelemre. Keressük meg a *Text* tulajdonságot az ablakban, majd változtassuk meg **label1**-ről a következőre: **Írja be a nevét**, és üssünk Entert. Az űrlapon a címkében megjelenő szöveg az általunk beírta módosul.



Tip: Alapértelmezésként a tulajdonságok kategóriákba csoportosítva jelennek meg. Ha inkább abc-sorrendben szeretnénk megjeleníteni őket, kattintsunk a tulajdonságok felett található Alphabetical gombra.

7. Jelenítsük meg a Toolbox-ot. Válasszuk a TextBox (szövegmező) gombot, majd kattintsunk az űrlapra. Az űrlapon megjelenik egy *TextBox* vezérlőelem. Vigyük a *TextBox* vezérlőelemet közvetlenül a *címke*-vezérlőelem alá.

Tipp: A vezérlőelemek mozgatásakor az igazítást rácpontok segítik: azonnal láthatjuk, hogy a mozgatott vezérlőelem igazodik-e az űrlapon elhelyezett többi vezérlőelemhez. Így gyors vizuális eligazítást kapunk a vezérlőelemek szabályos elrendezéséhez az űrlapon.



8. Jelöljük ki a *TextBox* vezérlőelemet, majd a tulajdonságablakban keressük meg a *Text* tulajdonságot. Írjuk be a következőt: **ide**. A szövegmezőben azonnal megjelenik az *ide* szó.
9. Keressük meg a (*Name*) tulajdonságot is a tulajdonságablakban. A Visual Studio 2005 automatikusan elnevezi a vezérlőelemeket és az űrlapokat. Az alapértelmezett név – bár jó kiindulási pont – nem mindig találó. Változtassuk meg a *TextBox* vezérlőelem nevét **userName**-re.

Megjegyzés: A „Változók, operátorok és kifejezések használata” című 2. fejezetben bővebben tárgyaljuk a vezérlőelemek és változók elnevezési rendszerét.

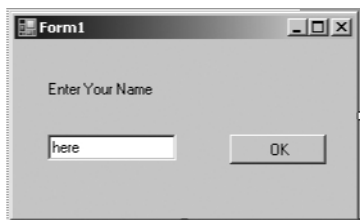


10. Jelenítsük meg a Toolboxot, kattintsunk Button (gomb) gombra, majd kattintsunk ismét az űrlapra. Húzzuk a *gomb*-vezérlőelemet a *TextBox* vezérlőelem mellé úgy, hogy egy vonalban legyen a két vezérlőelem.
11. A tulajdonságablakban változtassuk meg a *gomb*-vezérlőelem *Text* tulajdonságát **OK**-ra. A (*Name*) tulajdonság legyen **ok**. A gomb felirata megváltozik.
12. Kattintsunk a Form1 űrlapra a Designer View ablakban. Figyeljük meg, hogy az űrlap alján, jobb oldalán, valamint a jobb alsó sarokban méretezőpontok (kis négyzetek) láthatók.
13. Vigyük az egérmutatót a méretezőpont felé. Az egér mutatója kétirányú nyíl alakját veszi föl.
14. Az űrlap átméretezéséhez tartsuk nyomva a bal egérgombot, és húzzuk arrébb a mutatót. Engedjük fel az egérgombot, ha a vezérlőelemek közti térköz nagyjából egyenlő.

Tipp: A vezérlőelemek nagy részét is átméretezhetjük hasonlóképpen: kattintsunk az átméretezni kívánt vezérlőelemre, majd az elem sarkaiban megjelenő méretezőpontokkal válasszuk ki a kívánt méretet. Az űrlapoknak csak egy méretezőpontja van, míg a vezérlőelemeknek négy (minden sarokban egy). Az űrlapon ugyanis a jobb alsó sarokban lévő méretezőponton felül felesleges lenne további méretezőpontok elhelyezése. Néhány vezérlőelem, így a *cimke*-vezérlőelem mérete is automatikusan igazodik a tartalmához, méretezéssel azon nem tudunk változtatni.



Az űrlapunknak az alábbi ábrán láthatóhoz kell hasonlítani:



15. A Solution Explorerben kattintsunk jobb gombbal a Form1.cs fájlra, majd válasszuk a View Code (kód megjelenítése) parancsot. A Code and Text Editor ablakban megjelenik a Form1.cs forrásfájl.

A Design View/Code and Text Editor ablak felett most két Form1.cs nevű fül látható. Bármikor visszatérhetünk a Design View ablakba, ha a [Design] utótaggal ellátott fülre kattintunk.

A Form1.cs fájlban a Visual Studio által automatikusan generált kód egy részletét láthatjuk. Nézzük meg alaposabban az alábbi elemeket:

- **Using direktívák** A Visual Studio 2005 több *using* direktívát is beszúrt a forrásfájl elejére (többet, mint az előző példában). Például:

```
using System.Windows.Forms;
```

Ezekre a további névterekre a grafikus alkalmazások fejlesztésekor használt osztályok és vezérlőelemek miatt van szükség. (Többek között a *TextBox*, a *Label*, valamint a *Button* osztályok miatt.)

- **A névtér** A legfelső szintű névtérnek a Visual Studio a projekt nevét állította be:

```
namespace winFormHello
{
    ...
}
```

- **Egy osztály** A *WinForm Hello* névtéren belül a Visual Studio létrehozott egy *Form1* nevű osztályt.

```
namespace winFormHello
{
    partial class Form1 ...
    {
        ...
    }
}
```

Megjegyzés: Egyelőre hagyjuk figyelmen kívül a partial kulcsszót. Hamarosan részletesebben foglalkozunk vele.



Ez az osztály a tervezési nézetben létrehozott űrlap megvalósítása. (Az osztályokkal bővebben a 7. fejezetben foglalkozunk.)

Ebben az osztályban van egy rövid kód, amit konstruktornak hívnak, ez hívja meg az `InitializeComponent` metódust. Ezenkívül szinte semmi más nincs benne. (Konstruktor alatt az osztály nevével megegyező különleges metódust értjük, amely az űrlap létrehozásakor végrehajtásra kerül. Ide szűrhatjuk az űrlap inicializálására használt kódot. A konstruktorokkal a 7. fejezetben is foglalkozunk.) Azonban a Visual Studio 2005 bővészkedik: elrejt egy pár dolgot, amit mindjárt meg is mutatok.

Windows Forms alkalmazásokban a Visual Studio 2005 akár nagyméretű kódot is generálhat. Ezekkel a kódokkal hozza létre és jeleníti meg az űrlapot az alkalmazás indításakor, valamint ezzel állítja elő és teszi helyükre a vezérlőelemeket is. Ez a kód azonban megváltozik, amikor vezérlőelemeket helyezünk az űrlapra, és megváltoztatjuk azok tulajdonságait. Ezt a kódot nem szükséges módosítani (sőt, ha meg is változtatunk benne valamit, azt a Visual Studio valószínűleg visszaírja, amint tervezőnézetben változtatunk programunkon). Ezért a Visual Studio elrejtje ezeket az utasításokat.

A rejtett kód megjelenítéséhez térjünk vissza a Solution Explorerbe, és kattintsunk a Show All Files (minden fájl megjelenítése) gombra. Megjelenik a bin és az obj könyvtár, ahogy a konzolalkalmazás fejlesztésekor is láttuk. Azonban észrevehetjük, hogy a `Form1.cs` mellett megjelent egy + jel. Ha erre az + jelre kattintunk, két fájlt láthatunk: a `Form1.Designer.cs`-t és a `Form1.resx`-t.

A `Form1.Designer.cs` tartalmának megjelenítéséhez kattintsunk kétszer a fájl nevére. Itt láthatjuk a `Form1` osztály maradék kódját. `C#`-nyelvben lehetőségünk van az osztály kódját több forrásfájlba is felosztani, ha az osztály minden részét partial kulcsszóval jelöljük. Ebben a fájlban találhatunk egy Windows Form Designer generated code címkéjű részt. Ha az + jellel kibontjuk a részt, megláthatjuk, milyen kódot generált a Visual Studio 2005, míg mi a Design View ablakban terveztük az űrlapot. A fájl tartalma a következőkből épül fel:

- **InitializeComponent metódus** Ez a metódus a `Form1.cs` fájlban is megjelenik. A metódus utasításai beállítják a tervezési nézetben elhelyezett vezérlőelemek tulajdonságait. (A metódusokról részletesen a 3. fejezetben szólnunk.) A tulajdonságablakban beállítottak a metódus alábbi utasításaiban jelenik meg:

```
...
private void InitializeComponent()
{
    this.label1 = new System.Windows.Forms.Label();
    this.userName = new System.Windows.Forms.TextBox();
    this.ok = new System.Windows.Forms.Button();
    ...
    this.label1.Text = "Enter your name";
    ...
    this.userName.Text = "here";
    ...
    this.ok.Text = "OK";
    ...
}
...
```

- **Három mező** A Form1 osztályban a Visual Studio 2005 létrehozott három mezőt. Ezek a fájl vége felé találhatók:

```
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox userName;
private System.Windows.Forms.Button ok;
...
```

Ez a három mező a tervezési nézetben elhelyezett három vezérlőelemet valósítja meg. (A mezőket részletesen a 7. fejezetben tárgyaljuk.)

Úgy gondolom, fontos még egyszer hangsúlyozni, hogy bár érdekes lehet átnézni a fájl tartalmát, sohase szerkesszük azt kézzel. A Visual Studio 2005 automatikusan frissíti a fájl tartalmát, ha bármit változtatunk a tervezési nézetben. Az általunk írt kódokat pedig a Form1.cs fájlba kell tenni.

Egyesekben felmerülhet a kérdés, hogy hol a Main metódus, és hogyan jelenik meg az alkalmazás futtatásakor az űrlap. Emlékezzünk rá, hogy a Main metódus azt határozza meg, honnan indul a program futtatása. Most vegyük észre, hogy a Solution Explorerben van még egy forrásfájl. Ennek neve Program.cs. Ha kétszer kattintunk a fájlra, a következő kód jelenik meg a Code and Text Editor ablakban:

```
namespace winFormHello
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
```

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
}
```

A kód nagy részét figyelmen kívül hagyhatjuk. Azonban vessünk egy pillantást a következő kulcsfontosságú utasításra:

```
Application.Run(new Form1());
```

Ez az utasítás létrehozza az űrlapot, megjeleníti, majd átadja neki az irányítást.

A következő feladatban megtanuljuk, hogyan rendelhetünk az OK gombhoz kódot.

A kód megírása az OK gombhoz

1. Kattintsunk a Code and Text Editor ablak felett látható Form1.cs[Design] fülre. Ezzel megnyitjuk a Form1-et tervezési nézetben.
2. Helyezzük az egérmutatót az OK gomb felé, majd kattintsunk kétszer. A Code and Text Editor ablakban megjelenik a Form1.cs forrásfájl. A Visual Studio a Form1 osztályhoz hozzáadott egy `ok_Click` nevű metódust. (Ezenkívül a Form1.Designer.cs fájlban található `InitializeComponent` metódushoz is automatikusan hozzáadott egy utasítást, amely meghívja az `ok_Click` metódust, ha az OK gombra kattintunk. Ezt metódusreferencia-típus segítségével éri el. A metódusreferencia-típusokkal a „Metódusreferenciák és események” című 16. fejezetben foglalkozunk bővebben.)
3. Írjuk be az alábbi `MessageBox` utasítást az `ok_Click` metódusba. A teljes metódusnak valahogy így kell kinéznie:

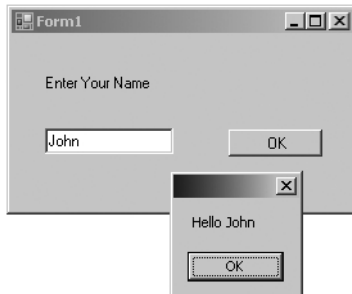
```
private void ok_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Hello " + userName.Text);
}
```

Győződjünk meg róla, hogy helyesen gépeltük-e be a fenti kódot, különös tekintettel a sor végén található pontosvesszőre.

Ezzel készen is állunk első Windows-programunk futtatására.

A Windows-program futtatása

1. A Debug menüben kattintsunk a Start Without Debugging menüpontra. A Visual Studio elmenti munkánkat, majd lefordítja és futtatja programunk. Megjelenik a Windows-űrlap.
2. Írjuk be a nevünket, majd kattintsunk az OK gombra. A megjelenő üzenetdoboz név szerint üdvözlö bennünket.



3. Az üzenetdobozban kattintsunk az OK gombra. Ennek hatására az üzenetdoboz bezárul.
 4. A Form1 ablakban kattintsunk a Bezárás gombra (az ablak jobb felső sarkában látható X). A Form1 ablak bezárul.
- **Ha szeretnénk továbblépni a következő fejezetbe,** hagyjuk bekapcsolva a Visual Studio 2005 programozói környezetet, és lapozzunk a 2. fejezethez.
 - **Ha ki szeretnénk lépni a Visual Studio 2005 fejlesztői környezetből,** kattintsunk a File menüre, majd az Exit menüpontra. Ha megjelenik a Save párbeszédpanel, munkánk elmentéséhez kattintsunk a Yes gombra.

1. fejezet – Gyorsreferencia

Ehhez	Tegyük ezt	Billentyű-kombináció
Konzolalkalmazás létrehozása	A New Project párbeszédpanel megnyitásához kattintsunk a File/New/Project parancsra. Válasszuk a Visual C# projekt típust. A sablonok közül válasszuk a Console Application (konzolalkalmazás) ikont. A Location (elérési út) sorban adjuk meg a projekt könyvtárát. Adjunk nevet a projektnek. Kattintsunk az OK gombra.	
Windows-alkalmazás létrehozása	A New Project párbeszédpanel megnyitásához kattintsunk a File/New/Project parancsra. Válasszuk a Visual C# projekt típust. A sablonok közül válasszuk a Windows Application (Windows-alkalmazás) ikont. Az elérési út sorban adjuk meg a projekt könyvtárát. Adjunk nevet a projektnek. Kattintsunk az OK gombra.	
Alkalmazás lefordítása	A Build menüben kattintsunk a Build Solution parancsra.	F6
Az alkalmazás futtatása	A Debug menüben kattintsunk a Start Without Debugging menüpontra.	Ctrl+F5