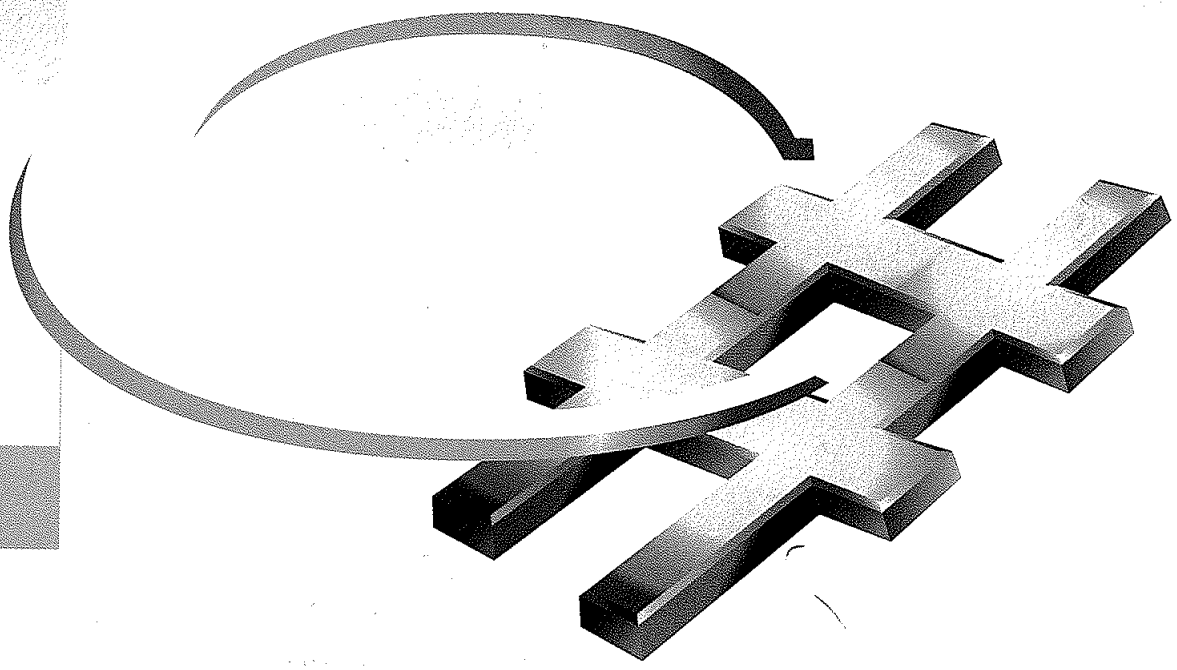
...önálló tanuláshoz...

Sipos Marianna

Programozás élesben

.net

KÉNYELMES, GYORS ÉS HATÉKONY
.NET ALKALMAZÁSFEJLESZTÉS,
MODERN FEJLESZTŐESZKÖZ,
KOMPONENSALAPÚ PROGRAMOZÁS,
EGYSZERŰEN FELDOLGOZHATÓ,
NAPRAKÉSZ ISMERETEK



Előszó

Elsősorban a családomnak szeretném megköszönni a könyv írásához nyújtott támogatást! A türelmet és megértést, a szoba és a számítógép átengedését, az első személyt, aki valóban kezdőként az én könyvemből ismerkedett a programozással.

A szakmai munkában az ELTE programozó matematikus és a BMF NIK műszaki informatikus hallgatói segítették kérdéseikkel, megjegyzéseikkel a kötet összeállítását. A BMF NIK oktatóinak véleményét is beépítettem a szövegbe. Ezúton szeretném megköszönni szakmai lektoromnak Albert Istvánnak alapos, a mondanivaló mélyébe tekintő javításait és javaslatait. Anyanyelvi lektorommal Temesy Klárával újra megküzdöttünk az egybeírás-különírás problémáival. Ez új szakkifejezések használatakor mindig kritikus kérdés. Tanácsai elősegítették, hogy a szöveg érthetőbb, áttekinthetőbb legyen.

A könyv találó címét Baga Zoltánnak köszönheti. Különlegesen szép borítóját Nagy Ágnes tervezte, extra kívánságaim maximálisan toleráns és ötletes beépítésével.

A könyv céljai közt a fejlesztés menetének és technikájának ismertetése is szerepel. Fejlesztőeszköz használatakor a legritkább esetben a kód olvasásával ismerkedünk meg a feladattal, vagy igazodunk el a részletekben, sőt a fejlesztés során is alig néhány sort kell a hosszú forráskódból begépelnünk. A könyv feladatainak forráskódja így nem szükséges a könyv használatához, mégis hozzáférhető a következő címen:

www.nik.hu/aii/oktatok/siposmarianna.html

Végül köszönöm a kedves olvasónak a megtiszteltetést, hogy e könyvet választotta ismeretei bővítéséhez. ☺

S.M.

Tartalomjegyzék

GYAKORLATOK

Bevezető

A számítógép merev, a gyártás után már nem változtatható elemeit nevezik hardvernek, míg a rugalmas, módosítható, továbbfejleszthető részek a szoftver elemek. A szabványok a szoftvert mint programok, adatok és dokumentációk együttesét definiálják. Az alkalmazások olyan szoftverek, melyek nem a számítógép működését biztosítják, hanem valamely cél érdekében annak felhasználását (alkalmazását) támogatják. Alkalmazás a gépen futó szövegszerkesztő, az internetes böngészést támogató szoftver, de az általa elért szerveren futó, valamilyen szolgáltatást nyújtó program is.

Mi az alkalmazásfejlesztés egy kényelmes, a gyors fejlesztést támogató fejlesztői környezet – a Visual Studio – segítségével megvalósított formáját választjuk abból a célból, hogy a ma elvárásainak megfelelni tudó alkalmazások és fejlesztésük alapfogalmaival megismerkedjünk.

A C# nyelv a C nyelvcsalád tagja

A Visual Studio lehetővé teszi alkalmazások különböző nyelvi környezetben történő fejlesztését, sőt azt is, hogy egy alkalmazáson belül az egyes elemeknél különböző nyelveket használjunk a fejlesztés során. A C# programozási nyelvet kifejezetten azzal a céllal hozták létre, hogy hatékonyan tudjon megfelelni a ma programozási elvárásainak. Amint az nevéből is kiderül, a C nyelvcsalád egy újabb tagjáról van szó.

Az első C nyelv 1972-ben készült el, Dennis Ritchie nevéhez kötődik. Az alacsonyszintű programozási nyelvet a Unix operációs rendszerhez készítették, s csak később terjedt el használata a többi operációs rendszeren. A nyelvet 1978-ban Brian Kernighan fejlesztette tovább az ismert (K&R) C nyelvvé.

1983-ban hozta létre Bjarne Stroustrup a C++ nyelvet, melyet eredetileg C osztályokkal néven neveztek. A cél az objektumorientált programozás támogatása volt a procedurális hagyományok megtartása mellett. A világ egyik legelterjedtebb programozási nyelve lett sokrétűségének köszönhetően. A C nyelvet 1983-ban, míg a C++-t 1998-ban standardizálták. Platformfüggetlenségét a két nyelv annak köszönhette, hogy szinte minden operációs rendszerhez készült fordítója. A standard forráskód módosítás nélkül fordítható bármely operációs rendszerre.[1] Ehhez azonban szükség van a fordítóra és a fordításhoz értő szakemberre.

Az 1995-ben James Gosling nevéhez fűződik a Java nyelv létrehozása C, C++ és Smalltalk alapokon. A Java tiszta objektumorientált nyelv. Elterjedését annak köszönhette, hogy az internet használatának általánossá válásával a platformfüggetlenség új szintjét kínálta. A különböző környezetekben a gépre telepített Java Virtuális Gép (JVM) a bájtkódra fordított fájlt automatikusan interpretálja (értelmezi). Vagyis nem kell szakember a távoli gépről letöltött kód futtatásához. Az interpretálás azonban gyakran lassúnak bizonyul.

A C nyelvcsalád legfiatalabb nyelve a C# (C sharp, ejtsd szí sárp)[2]. 2000-ben találkozhattunk a béta változattal, a hivatalos kiadás 2002 tavaszán került forgalomba. A C# csoport vezetője Anders Heilsberg, aki korábban a Borland Pascal és a Delphi nyelveket is készítette. Ő volt a Microsoft J++-t és az MFC-t készítő csoport vezetője is.

A C# egy tiszta objektumorientált programozási nyelv, melyet C++ és Java alapokon fejlesztettek ki. A .NET futtatókörnyezethez készíthetünk vele skálázható, biztonságos web alkalmazásokat. Támogatja a komponens alapú és a többrétegű alkalmazásfejlesztést.[3] „A C#-et használták már dinamikus webhelyek, fejlesztőeszközök, sőt fordítóprogramok készítéséhez is."[4]

---

[1] Wikipedia: Timeline of programming languages, http://wikipedia.org/Timeline_of_programming_languages

[2] Itthon a zenére utalva cisz-nek is szokták nevezni.

[3] További információk a www.microsoft.com keresőjébe beírva a Visual C# Language szöveget, vagy közvetlenül a: http://msdn.microsoft.com/library/en-us/cscon/html/vcoriCStartPage.asp lapon.

[4] Bradley L. Jones: C# mesteri szinten, Kiskapu Kiadó, 2004. 4. old.

A C# platformfüggetlenségét a .NET futtatókörnyezet biztosítja. A JIT (just in time) technika ötvözi az automatikus helyben futtatást a lefordított kód gyors végrehajthatóságával.

A .NET

A .NET egy operációs rendszer szintű futtatókörnyezet. David S. Plattot idézve: „Előregyártott infrastruktúra az internetes alkalmazásokban jelentkező problémák megoldására”[1]. Ma még külön telepítjük az operációs rendszer fölé, de a jövőben része lesz telepítőcsomagjuknak.

A .NET megkönnyíti különböző nyelveken írt komponensek együttműködését, a CLR (Common Language Runtime) és a CTS (Common Type System) segítségével (6. fejezet). A felügyelt kód számára biztosítja az operációs rendszer szintű automatikus szemétgyűjtést, és rendelkezésünkre bocsátja a .NET osztálykönyvtárat, melynek felhasználásával gyorsan és hatékonyan fejleszthetünk alkalmazásokat a .NET futtatókörnyezethez. A .NET-ről bővebben olvashatunk a 6-8. fejezetekben. Mélyebb információkat ’A .NET Framework és programozása’[2] című kötetben találhat az olvasó.

A C# alkalmazások platformfüggetlensége a .NET platformfüggetlenségétől függ. A különböző Microsoft platformokon maga a Microsoft biztosítja a platformfüggetlenséget, de már létezik a Microsofttól független .NET megvalósítás is. A Mono egy nyílt forráskódú .NET kompatibilis ECMA standard C# fordítóval, CLR-el. Fut Linux, UNIX, Mac OS X és Windows operációs rendszeren is.

A Visual Studio fejlesztőeszköz

A könyv a Visual Studio fejlesztői környezet szolgáltatásait használja az alkalmazások fejlesztéséhez. Az eszköz egy többablakos, a fejlesztés kényelmét szolgáló környezet. A Visual Studió ablakai dokkolhatók, vagyis odaragaszthatók a keret egyes részihez úgy, hogy a címsort megfogva az egérrel vonszoljuk az ablakot. Vonszolás közben egy téglalap keret jelzi az ablak elengedett helyzetű helyét a felületen. Így tehát különböző megjelenésű lehet az eszköz. Mi itt ragaszkodunk a telepítés utáni változathoz.

---

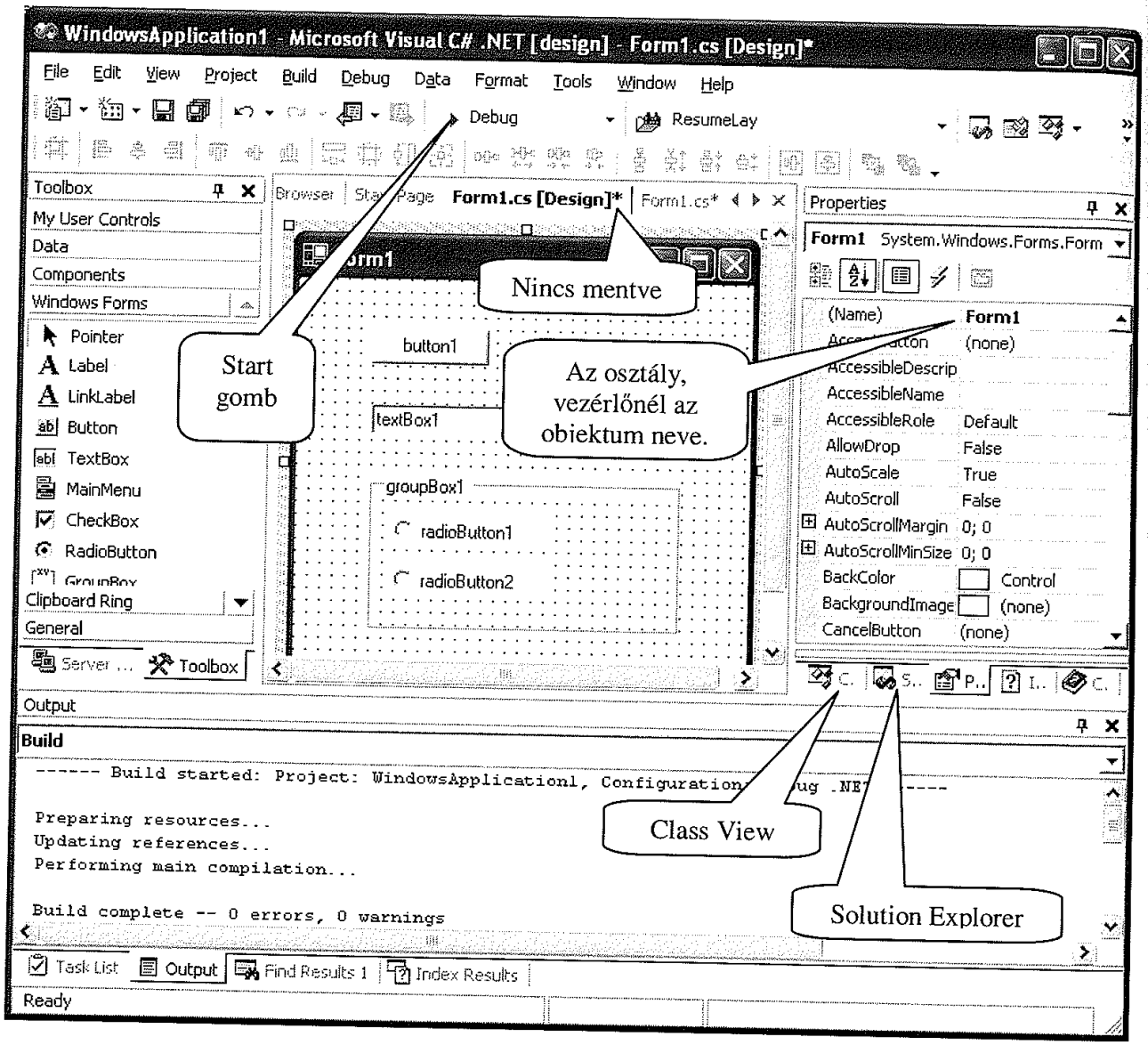
[1] David S. Platt: Bemutatkozik a Microsoft .NET, Szak Kiadó Kft. Bicske, 2001. 23. old.

[2] Albert István, Balássy György, Charaf Hassan, Erdélyi Tibor, Horváth Ádám, Levendovszky Tihamér, Péteri Szilárd, Rajacsics Tamás: A .NET Framework és programozása, Szak kiadó, 2004.

A Visual Studio a Solution adatai közt elmenti beállításainkat is, így ha egyszer kényelmes felületet alakítottunk ki magunknak, a következőben ezt a felületet látjuk indításkor.

A New Project /Visual C# project / Windows Application választás után az 1. ábra által mutatott eszközt látjuk.

Az ábra közepét a Form1 osztály felhasználói felületének szerkesztését támogató [Design] nézet foglalja el. Ide tudunk a baloldali Toolbox ablakból vezérlőket vonszolni, melyek mögött automatikusan létrejönnek a szövegüknek (Text tulajdonság) megfelelő – az osztályuk nevéből és egy sorszámból összefűzött – objektumok. Ezeket az objektumokat javasolt 'beszédes' nevűre átnevezni.

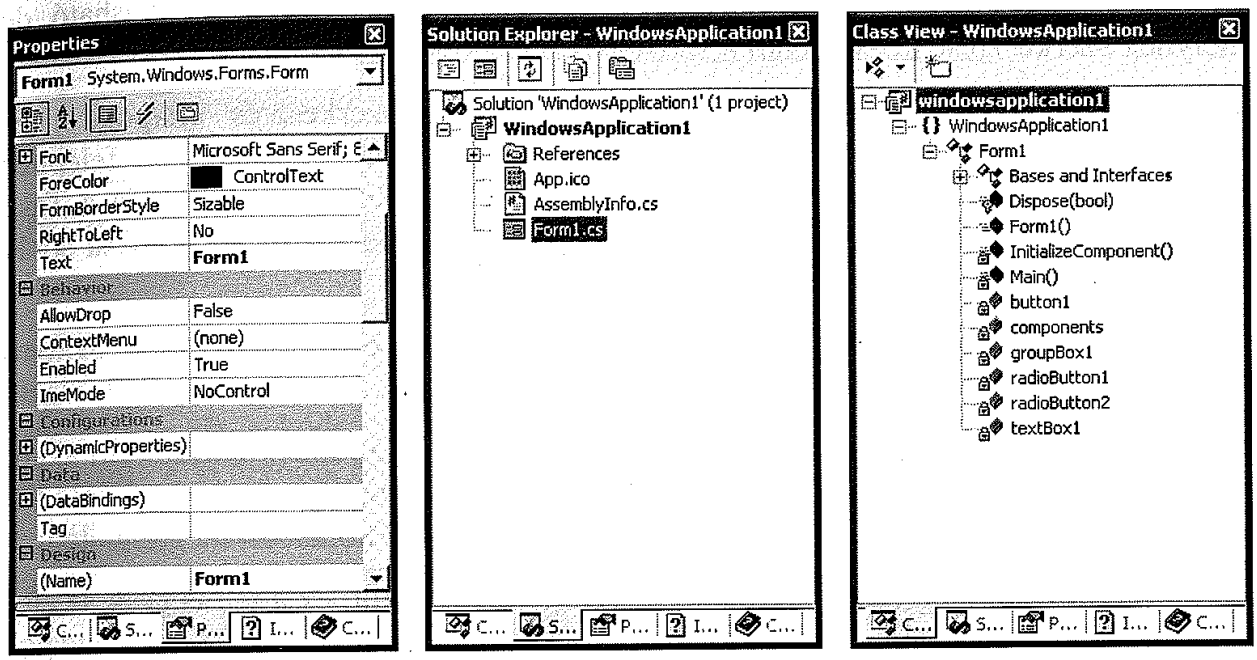


Bevezetés 1. ábra

Az átnevezéshez célszerűen az ábra jobb oldalán látható Properties ablak Name tulajdonságát válasszuk! A propertiest lehet kategóriánként és ABC sorrendben is keresni, az ablak megfelelő felső eszközgombjának választásával.

A jobb oldalon még további fülek teszik lehetővé a projekt adatainak különböző szempontú feldolgozását. A kis hely miatt az ábrán S betűvel jelzett fül – Solution Explorer – választására a Solution, azon belül a projekt fájljaival dolgozhatunk. A C betűvel jelzett fül a Class View rövidítése. A Class View mint nevéből is kiderül, az alkalmazásban létrehozott osztályokról ad információt a grafikus UML jelöléseket használva (2. ábra).

Az eddig említett ablakokat a két oldalon, ha nincsenek nyitva, a View menüből érhetjük el.



Bevezetés 2. ábra a Properties, a Solution Explorer és a Class View ablak

A jobboldali ablak további fülei a súgó ablakokat adják, melyek a Help menü Contents, Index vagy Search menüpontjával nyithatók ki. A súgó szövege középen jelenik meg. A súgó, mivel a fejlesztőeszköz több nyelvű, tehát több programozási nyelven közöl információkat. Az ablak tetején látható egy tölcsér formájú ikon mely nyelvi szűrőként működik. Ha kiválasztjuk benne a C# nyelvet, akkor áttekinthetőbb információkhoz jutunk.

Ha a [Design] nézet Form1 ablaka (1. ábra) fölött jobbegérrel kattintunk a gyorsmenüből kiválaszthatjuk a View Code menüpontot, mely a Form1 osztály forráskódját mutatja. A forrásfájlok neve cs kiterjesztésű, utalva a C# nyelvre. A két nézet között az ablak tetején található fülekkel váltogathatunk.

Ha a kód az utolsó mentés óta módosult, a füleknél a forrásfájl neve után egy * jelzi. Az ábrán is ezt láthatjuk. Ha csak egy fájlt akarunk menteni, használhatjuk az egy

floppyval jelölt ikont, az összes fájlmentéséhez a több floppyst. A fejlesztés során ritkán szoktunk menteni, mert a Visual Studio fordítás előtt menti a forrásfájlokat. Ilyenkor eltűnik a * a fájlok füleiről.

A forráskód bal szélén kis kockákban +, – jeleket látunk, amelyek lehetővé teszik a forráskód áttekinthetőségének növelését. A C++ nyelv header fájljai többek között lehetővé teszik, hogy egy osztályról átfogó képet kapjunk, lássuk, milyen szolgáltatásokat nyújt anélkül, hogy elmélyednénk a függvények megvalósításában, a kód részleteiben. A C# esetén ezt a funkciót a szövegszerkesztő vállalja magára azzal, hogy a függvényeink kódja becsukható vagy kinyitható.

A szövegszerkesztőben ezen kívül régiókat is létrehozhatunk, mellyel nagyobb egységek ily módon történő kezelését tesszük lehetővé. Ilyen régió a kódgenerátor által létrehozott kód (Windows Form Designer generated code), mely olvasható, de melynek szerkesztése közvetlenül a szövegszerkesztőből nem ajánlott.

Az 1. ábra alján látható Output ablak a fordítás szerkesztés állapotáról és eredményéről nyújt információkat. Most épp azt írta ki, hogy a felépítés befejeződött. 0 hiba, 0 figyelmeztetés. A hibák és figyelmeztetések listája a Task List fülben található. A nyomkövetés szolgáltatásairól a feladatok megoldása során találunk információt.

Az Output ablak combo boxában a Build helyett a Debug elemet választjuk, akkor a futtatás lefolyásáról kapunk információkat az ablakban.

A Find Results fül az utolsó keresés eredményét mutatja, az Index Results fül akkor aktivizálódik, ha a súgóban az adott címszóhoz több bejegyzés is kapcsolódik. Akkor itt tudjuk kiválasztani, melyiket kívánjuk a középső ablakban megjeleníteni.

A fejlesztőeszköz még láthatóan rengeteg szolgáltatást nyújt, ez a leírás csak az első pillanatban történő eligazodást szolgálja. A további részletekkel a feladatok megoldása során találkozhatunk.

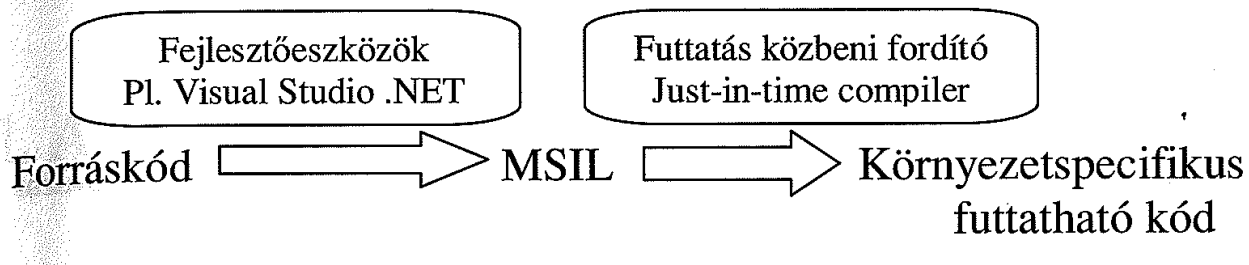
Fordítás és futtatás

Technikailag nagyon egyszerűen, a Start gomb (1. ábra) választásával a Visual Studio menti, fordítja és futtatja az alkalmazást. Ha a fordításkor hibát talál a fordító, figyelmeztető ablak jelenik meg, és a folytatás elutasítása esetén a Task List ablakban a hibára duplán kattintva a hibás kódsorra ugrik a szerkesztőben.

Alapértelmezésben kétféle módot kínál a fordításra: a Debug és a Release módot, de további fordítási beállítások is megadhatók. A Debug beállítást használjuk a fejlesztés során, mert a nyomkövetéshez szükséges információkat is szolgáltatja. Release módban a kész alkalmazást szokás fordítani, ha már nincs szükségünk a hibák elemzéséhez a nyomkövetésre.

A C# fordításakor nem közvetlenül natív kódot kapunk, hanem egy köztes nyelvű kódot **MSIL** (Microsoft Intermmediate Language). Ez a kód bármely .NET **CLR** (Common Language Runtime) környezettel rendelkező gépen futtatható. Vagyis, ha a gépünkre nincs telepítve a .NET CLR, akkor hiába exe kiterjesztésű az állomány, nem futtatható a gépen.

A köztes kódú fájl tehát közvetlenül nem futtatható, le kell fordítani. A **JIT** (Just In Time) compiler feladata, hogy köztes kódról natív kódra fordítsa a fájlt futtatása közben. A JIT a .NET CLR által szállított fordító, mely a felhasználó gépén fut, a felhasználó CPU architektúrájához fordít natív kódra. Összegezve: a fordítás két lépésből áll. A fejlesztő köztes kódra (MSIL) fordít, melyet a felhasználó gépén futó JIT fordít le natív kódra. E két lépésnek köszönhető a platformfüggetlenség.



A JIT, figyelembe véve, hogy bizonyos függvények egyáltalán nem lesznek meghívva, csak azokat a kódrészleteket fordítja le, melyeket a futás során meghívunk. Ha a kódrészletet nem először hívjuk, már nem kell újrafordítani, a tárolt natív kód hívódik meg. Ezzel csökken a fordításra szánt idő.

Az MSIL fájlokat tehát futásidőben fordítjuk. Emiatt a program kezdetben lassabban fut, mint a hagyományos alkalmazások. Ez a platformfüggetlenség és a felügyelt környezet ára. Lehetőség van azonban olyan beállításra, hogy telepítéskor fordítsuk le a kódot (install-time code generation)[1]. Ehhez a Native Image Generator (Ngen.exe) áll a rendelkezésünkre, mely az adott környezethez fordítja a fájlt.

Megjegyzés:

Csak akkor hívódik meg a native image fájl, ha a fordításkor beállított paraméterek és környezet megegyezik a híváskor megadottal. (pl. .NET Framework verzió) Egyébként a szokásos JIT fordító lép életbe.

[1] MSDN Library for Visual Studio .NET 2003, JIT compilation, Compiling MSIL to Native Code.

XML dokumentáció generálása az alkalmazáshoz

A Visual Studio .NET C# forráskód esetén támogatja a kód és a megjegyzéssorok felhasználásával XML dokumentáció generálását. Ezt a következő beállítással érhetjük el:

Solution View

Projekt név kiválasztva

View menüpont

Property Pages

Configuration Properties

Build

XML Documentation File: Fájlnév

Alkalmaz, OK

A dokumentációs fájlt célszerű a projekttel (assemblyvel) azonos néven nevezni (xml kiterjesztés) és a futtatható állománnyal azonos könyvtárban elhelyezni.

1. Az osztály és az objektum

Az első programok bonyolult számítások gyors elvégzését célozták meg. A programozás fejlődésével egyre bonyolultabb igények jelentkeztek. A hardver és a programozás továbbfejlesztésével ezek az elvárások megoldódtak, de az új lehetőségek újabb és újabb elvárásokat teremtettek. E folyamat kapcsán alakult ki a hagyományos programozásból az áttekinthetőség érdekében megszorításokat hozó strukturált programozás. A még összetettebb programok írása érdekében pedig, az objektumorientált programozás (OOP). Az OOP lényege nagyon leegyszerűsítve, hogy az összetartozó adatokat és a rajtuk végzett műveleteket egy egységben tárolja. Ezzel lehetővé téve, hogy egyszerre egy kis egység – az osztály – létrehozásának problémáira koncentráljunk, vagy az osztályok közötti kapcsolatok segítségével egy magasabb absztrakciós szinten vizsgáljuk a program működését. Az OOP tette lehetővé többek között a grafikus felhasználói felület, az eseményvezérelt programozás valamint a komponensalapú programozás kialakulását.

A következőkben az OOP alapfogalmaival ismerkedünk meg.

1.1. Osztály, objektum fogalma

Az osztály (class) **egy típus megvalósítása.**

Az osztály leírása tartalmazza az osztály adattagjait, és a rajtuk végezhető műveleteket, a tagfüggvényeket. Az osztály leírása megadja az adattagok

láthatóságát, típusát és nevét, a tagfüggvények láthatóságát, visszatérési típusát nevét és kódját.

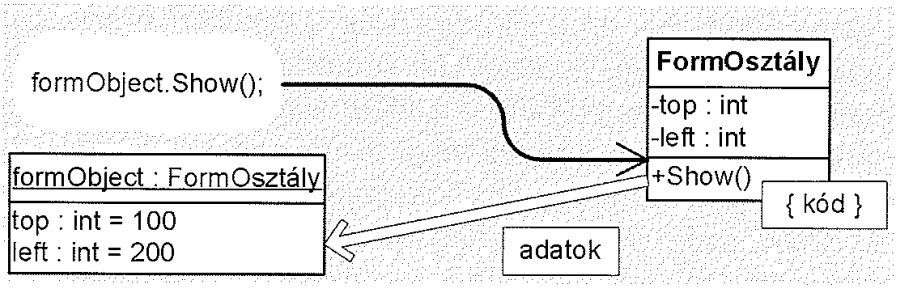
```
class Osztály
{
 private int adatTag;
 public int TagFüggvény(paraméterek)
        { kód }
}
```

Az osztály típusú változók az objektumok. Az objektumokat szokás az osztály példányainak, előfordulásának (**instance**) is nevezni.

Az objektum az osztály leírásában megadott típusú és nevű adatokat tárolja.

Az osztály a leírás, műveleteket végezni az objektumon lehet. A kapcsolatot illusztrálhatjuk a következő hétköznapi példával: Az osztály a sütemény receptje, az objektum maga a süti. A recept meghatározza a sütemény különböző tulajdonságait, a hozzávalókat és a műveleteket, amit elvégzünk rajtuk. A süteményt a recept segítségével készítjük el, ahogy az objektumot az osztály segítségével. A gyúrást, szaggatást, sütést a sütin (a rendelkezésünkre álló hozzávalókon, az objektum adatain) végezzük. A sütemény olyan lesz, amilyen a recept, (csak egy kicsit jobban ízlik)[1].

A műveleteket tagfüggvényekkel vagy tagoperátorokkal, objektumorientált szóhasználattal élve metódusokkal valósítjuk meg. Az osztály és objektum együttműködését mutatja a következő ábra:



1.1. ábra Osztály és objektumának együttműködése

A formObject FormOsztály típusú. Az formObject bal fölső sarkának adatait tároló tagváltozók (top, left) típusát és nevét a FormOsztály határozza meg. A formObject tárolja a bal fölső sarok értékeit. A kódban (balra fent) a formObjectre hívjuk meg a megjelenítését biztosító Show függvényt, mely a kódot a típusának megfelelő FormOsztályból veszi. A megjelenítéshez a Show metódusnak tudnia kell hova

[1] Guy Eddon, Henry Eddon: Inside Distributed COM, Microsoft Press, 1998, Redmond, Washington, 12. old.

rajzolja ki az ablakot. Az adatokat az azokat tároló, őt meghívó objektumból olvassa ki.

Az objektum:

- **Információt tárol** (tagváltozók, **member variables,** adattagok, **data members**).
- **Feladatokat hajt végre** (tagfüggvények, metódusok, **member functions**).

Az objektum adataihoz és metódusaihoz való hozzáférés egységesen a '.' operátorral történik.

```
objektum.adattag;
objektum.TagFüggvény();
```

Az objektum felelős feladatai elvégzéséért. Ezt a felelősséget az osztályának leírása biztosítja. Csak olyan adatokhoz, metódusokhoz férhetünk hozzá, melyek nincsenek előlünk elrejtve, csak olyan módosítást végezhetünk az adatokon, mely megengedett. Pl. egy osztályzatokat tároló objektum értéke csak 1 és 5 közötti egész szám lehet. Tehát biztosítja pl. az ellenőrzött adatbevitelt. Hagyományos változók esetén nem volt lehetőségünk ilyen ellenőrzések beiktatására.

Objektumok létrehozása jellemzően a következő módon történik:

```
MyClass myObject = new MyClass();
```

A MyClass() metódus a MyClass osztály konstruktora. Feladata az osztály objektumainak létrehozása, inicializálása. **A konstruktor** az osztály tagfüggvénye, **neve megegyezik az osztály nevével**, visszatérési értéke nincs.

Az objektum felszámolását egy másik speciális tagfüggvény a **destruktor** végzi. A destruktor **neve megegyezik az osztály nevével, csak előtte egy tilde '~' jel van** ~MyClass() visszatérési értéke és paramétere sincs. A destruktort felügyelt kód esetén az automatikus szemétgyűjtő hívja.

Konstruktor és destruktor használatára a 2. gyakorlat 3. feladatában látunk példát.

Ha kifejezetten nem tiltjuk, egy osztályból akárhány objektumot is készíthetünk. Minden objektum létrejöttekor lefoglalódik az osztályleírásban megadott adattagoknak megfelelő hely. Az egyes objektumok jellemzően különböző értékű adattagokat tartalmaznak. Az adattagokra történő hivatkozás az egyes objektumok adott mezőjére történő hivatkozást jelent. A tagfüggvények kódja az osztályleírásban található. Tagfüggvényhívás esetén tehát az osztály tagfüggvénye dolgozik az objektum által tárolt adatokon.

Megjegyzés:

Megállapodás: az osztályok és a tagfüggvények (tulajdonságok) nevét nagy, míg az objektumok, változók nevét kisbetűvel írjuk.

1.2. Adattagok

Osztály adattagja lehet a nyelv által biztosított típusok, az osztálykönyvtár által szolgáltatott osztályok és a fejlesztők által készített bármely típus példánya. A C# változókról részletesebben a 'A változók' szakaszban lesz szó.

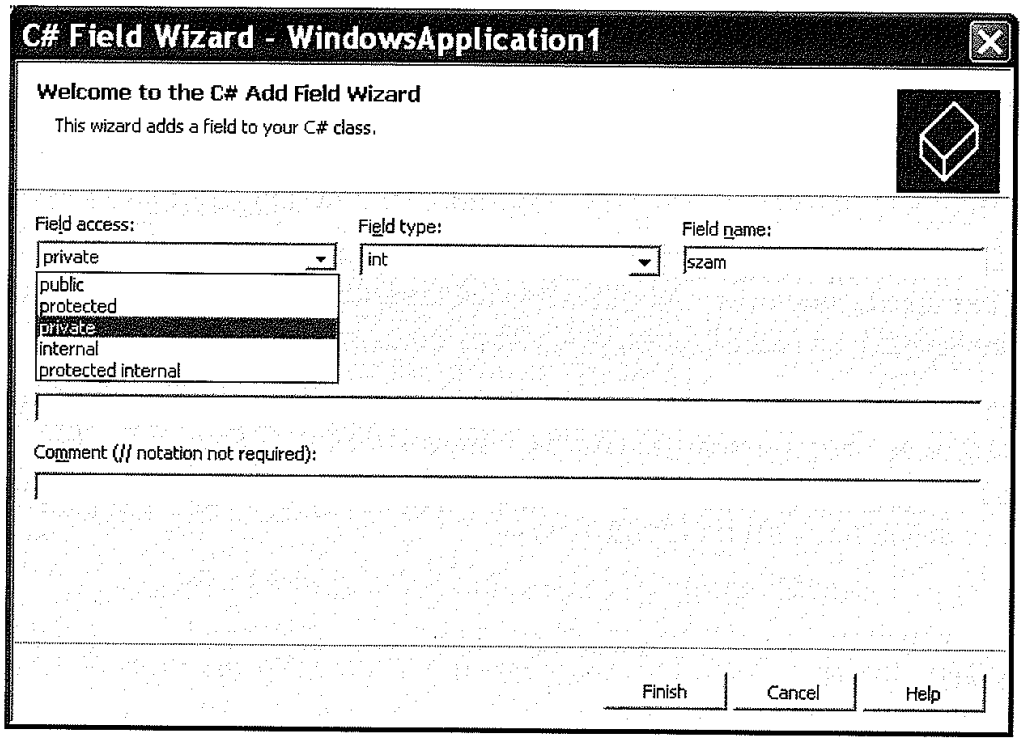
Új adattagot a definíció forráskódba történő beírásával vagy a következő módon hozhatunk létre:

Class View

Az osztály neve jobbegér

Add

Add Field



1.2. ábra Egy int típusú 'szam' nevű private adattag létrehozása

Az adattagok kétféle módosítóval (Field modifiers) láthatók el.

- **Static**: Osztályszintű vagy statikus adattagot hoz létre. **Statikus adattag** értéke nem az objektumban tárolódik, **így létrehozható és lekérdezhető**

akár egyetlen objektum létrehozása nélkül is. Osztályszintű adattagból osztályonként egy példány van. Az osztály neve és a '.' operátor segítségével hivatkozhatunk rá.

Tipikusan statikus tagja az osztálynak az aktuálisan létrehozott példányok számát mutató adattag. Ennek értéke ugyanis nem az objektumra, hanem az osztályra jellemző.

- **Constant**: A kódban a **const** kulcsszó áll előtte. **Constans adattag értéke nem változtatható.** Kötelező megadni a kezdőértékét.

Statikus adat a jelszóbekérő ablak az őt meghívó alkalmazás Main metódusában. Először az 8. gyakorlaton találkozunk vele.

1.3. A tagfüggvények

A **függvény** utasítások logikailag összefüggő csoportja, mely önálló névvel és visszatérési értékkel rendelkezik. A függvénynek a praméterlistájában átadhatunk adatokat, és a visszatérési értékben vagy a paraméterlistában visszakaphatjuk a végrehajtás eredményét. A függvényt a nevével és paraméterei megadásával hívjuk meg. A függvény visszatérési értéke lehet **void** is, ez azt jelenti, nincs visszatérési érték. Ekkor a **return** kiírása nem szükséges.

```
visszatérési érték típusa FüggvényNeve( paraméterlista )
{
 utasítások;
 return visszatérési érték;
}
```

A függvény **szignatúrája** (**signature**) alatt a függvény nevét és paramétereinek listáját értjük. A paraméterlista a paraméterek sorrendjét és típusát jelenti. A C# nyelv megengedi a függvények **átdefiniálását**, vagy **túlterhelését** (**overloading**). Vagyis létrehozhatunk több azonos nevű függvényt is, ha paraméterlistájuk eltér. Az egyes függvényeket a szignatúrájuk különbözteti meg egymástól. A fordító számára egyértelműnek kell lennie egy függvényhíváskor, hogy az átdefiniált függvények melyikét hívja aktuálisan.

Pl. lehet **Mozgat()** függvényünk, ha megállapodtunk, hogy ez a függvény az objektumot 1 egységgel mozgatja el x irányban. Írhatunk **Mozgat(int x)** függvényt is, amit akkor hívunk, ha függvényünknek meg akarjuk adni, hány egységgel mozduljon el az objektum x irányba. Írhatunk **Mozgat(int x, int y)** függvényt, ha azt akarjuk, hogy függőlegesen is megadhassuk az elmozdulást. De nem írhatunk Mozgat(int y) függvényt, ha már van Mozgat(int x), mert e kettőnél híváskor Mozgat(5); nem lehetne eldönteni, hogy x vagy y irányban szeretnénk 5 egységgel elmozdítani objektumunkat. A két függvény szignatúrája megegyezik, azonos nevűek és egy int típusú paraméterük van.

A **tagfüggvény**ek az osztályhoz tartozó, annak leírásában megadott függvények. Az objektumorientált gondolkodásmód épp abban tér el a strukturálttól, hogy míg strukturált programozás esetén az adatokat megadhatjuk egy egységben, de a rajtuk végzett műveletek általában a program különböző helyein szétszórva helyezkednek el, addig az objektumorientált programozás egyetlen helyen, az osztályban adja meg az adatokon kívül az azokon végezhető műveleteket is[1].

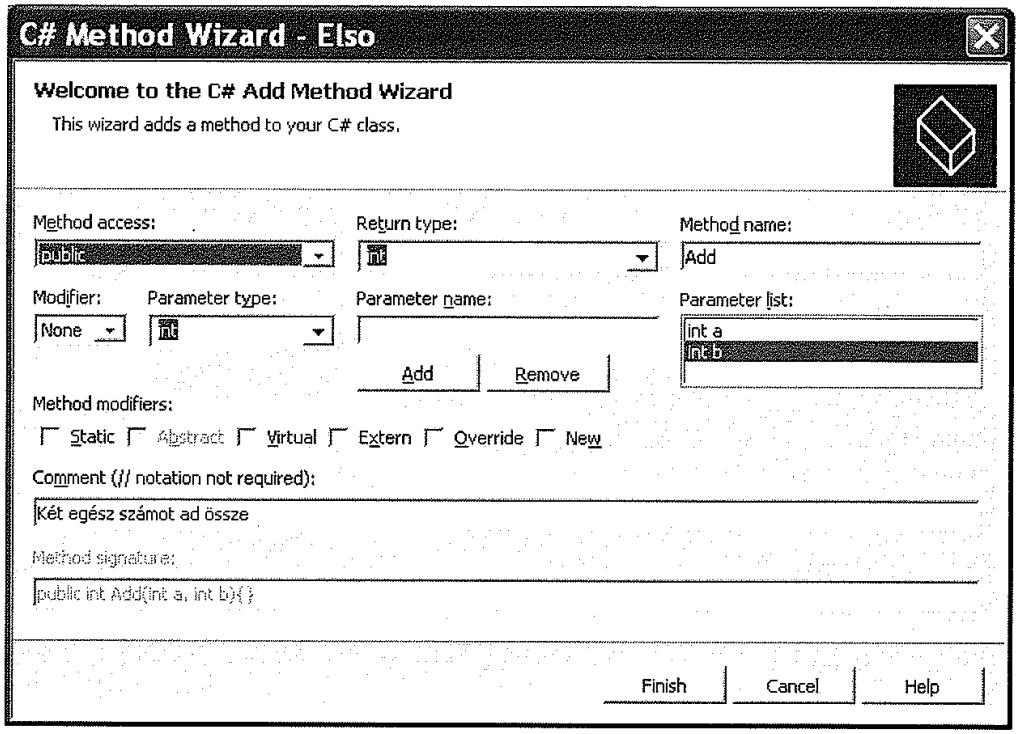
Új tagfüggvény létrehozása:

Class View

Az osztály neve jobbegér

Add

Add Method



1.3. ábra Két egészet összeadó Add függvény létrehozása

↳ **Static**: A függvényeknél is bejelölhetjük a Static módosítót (kódban static). Ekkor **osztályszintű tagfüggvény**t hoztunk létre, ami azt jelenti, hogy a függvény **objektum létrehozása nélkül is meghívható.** (Szintaxis: MyClass.MyStaticFuntion.) **Statikus metódusok csak statikus**

[1] Vég Csaba: Alkalmazásfejlesztés a Unified Modeling Language szabványos jelöléseivel, Logos 2000 Kiadó, 1999, 5. old.

adattagokon dolgozhatnak. Ilyen a Form osztályok Main metódusa vagy az Application osztály Run metódusa, amit a Mainben hívunk meg.

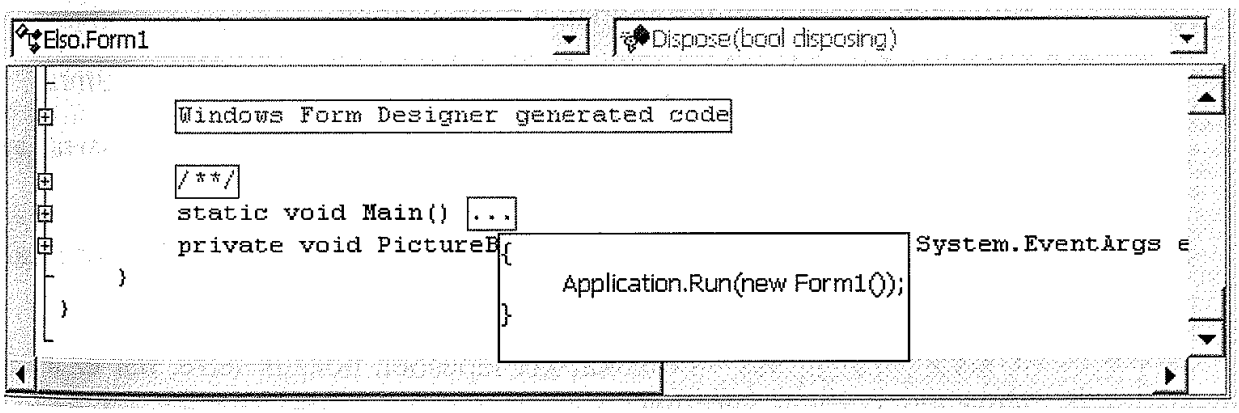
A tiszta objektumorientált nyelvekben (C#, Java) – a hagyományos objektumorientált programozásban globálisan megvalósított függvények statikusak. Nézzük meg bármelyik osztályt a súgóban! Az $\mathcal{S}$ ikonnal jelölt tagok mind statikusak és meglehetősen gyakoriak.

Gyakran használt statikus függvény a MessageBox osztály Show metódusa, melyet úgy használhatunk üzenetek megjelenítésére, hogy nem veszünk fel az osztályból objektumot. MessageBox.Show(„Statikus!"); De ilyen a Thread osztály Sleep metódusa is, melyet a program várakoztatására használunk (4. gyakorlat) vagy a Process osztály Start metódusa, mely programok indítására használatos (6. gyakorlat).

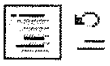
Az események kezelése során is új, eseménykezelő tagfüggvényeket hozunk létre.

A metódusok kódja a kódszerkesztőben a gépelés során automatikusan megjelenő súgó (**IntelliSense**) segítségével írható. A szó gépelését elkezdve, a legördülő listából kiválasztva a megfelelő elemet a következő operátor ('(', ';', '.') leütésével a név a kívánt formában kiíródik. Ennek a technikának köszönhetőek a gyakran this kulcsszóval kezdődő utasítások. A függvények, tulajdonságok neve a Ctrl + szóköz egyidejű lenyomásával is kiegészíthető, ez fölöslegessé teszi a this használatát. Az IntelliSense szolgáltatás a kód begépelése során piros aláhúzással jelzi a hibás kódrészletet. A hibát követően nem működik a szolgáltatás.

A függvények neve előtt a kódszerkesztő biztosít egy +/- ikont, melynek választásakor kinyílik / bezáródik a függvény kódja. Ha a függvényeket bezárjuk, az osztály leírását láthatjuk, míg kinyitva a megvalósítást is. Ha bezárt függvény neve mellett a ... ikonra mutatunk a kurzorral, a függvény kódja egy kinyíló ablakban olvasható. Megjegyzések esetén a /**/ szöveget tartalmazó doboz teszi ugyanezt.



1.4. ábra A szövegszerkesztő az összecsukott Main kódjával

Az eszközsor  ikonjaival a kijelölt sorok megjegyzéssé alakíthatók, illetve a megjegyzésjelet kivehetjük előlük.

1.4. A változók

Amikor eldöntjük egy adattag típusát, akkor döntünk az adat által felvehető értékekről és a rajta elvégezhető műveletekről. A közös típusrendszer a .NET környezetben a **CTS** (Common Type System) a CLR (Common Language Runtime) része. A CTS biztosítja a különböző nyelveken írt komponensek közti adatkommunikáció lehetőségét. A változók két kategóriába sorolhatók:

- Érték típusok (Value Types)

  Közvetlenül tartalmazzák az adatokat.

  A rajtuk végzett műveletek nem módosítanak más változókat.

  A minden érték típusnak van alapértelmezett kezdőértéke.

  (Súgó: types, default values)

  Pl. **int** az egészek tárolására, **float** a valós számok eléréséhez.

  (Súgó: types, built-in, Built-in Types Table)

  A következő két kódsor ugyanazt az eredményt adja, egy int típusú egész számára foglal helyet és 0 kezdőértékkel inicializálja.

```
int i = new int();
int i = 0;
```

- Hivatkozás típusok (Reference Types)

  Egy hivatkozást tartalmaznak egy adatterületre. Az adatok a hivatkozott területen tárolódnak a memóriában.

  Két hivatkozás típusú változó hivatkozhat ugyanarra az objektumra.

  Az egyiken elvégzett művelet módosítja a többi, ugyanarra az objektumra hivatkozó változó értékét.

  Hivatkozás típusú pl. a string a szövegek tárolására, de ilyenek az összes osztály típusú változók és a tömbök is.

  A hivatkozás típusú változókat két lépésben hozzuk létre, bár ez a két lépés egy sorba is kerülhet.

```
System.Windows.Forms.Button button; // A referencia változó deklarációja. Még nincs meg az objektum amire hivatkozik.
```

`button = new System.Windows.Forms.Button();` // A new operátor létrehozza az objektumot és visszaad egy referenciát, amit a hivatkozás típusú változóban tárolni tudunk.

A C# nyelv tartalmaz egy speciális értéket – a **null**-t. Egy referencia értéke null, ha nem hivatkozik semmire. Ha fel akarjuk szabadítani a hivatkozott területet, egyszerűen állítsuk a változó értékét null-ra! A memória felszabadítását az **automatikus szemétgyűjtő** végzi majd el.

Megjegyzés:

Az érték típusok nem lehetnek null értékűek.

Az automatikus szemétgyűjtést a 2. gyakorlat 3. feladatában demonstráljuk.

Egyszerű **tömb**öt hozhatunk létre a **[ ] operátor** segítségével. A tömb olyan értékek gyűjteménye, melyekre indexeléssel hivatkozhatunk.

Pl. egy számokból álló tömböt a következőképpen hozhatunk létre:

```
int[]  nums = new int[] {5,-1,12,8,5}
int[]  numbers = new int[5];
```

Mindkét esetben egy int típusú 5 elemű tömb keletkezett, csak a nums tömb elemekkel is fel van töltve. A tömb elemeire sorszámukkal hivatkozhatunk. A sorszámozás 0-val kezdődik. A következő utasítás a numbers tömb első (nulladik) elemének 8 értéket ad.

```
numbers[0] = nums[0]+3;
```

Ha a tömb nem létező elemére hivatkozunk pl. -1. vagy jelen példánál 5. vagy 12. – akkor (out of range) kivétel keletkezik.

A **string** típus egy gyakran használt speciális típus, így szánjunk néhány szót a megismerésére! A string típusnév valójában egy rövidebb elérés a **System.String** osztályhoz. Bár a sztringeken mindenféle módosító műveleteket végezhetünk, valójában egy sztring értéke nem módosítható. A módosító függvények, melyek látszólag a sztringet változtatják, valójában egy új példányt hoznak létre.

A [ ] operátor a sztringet tömbként kezeli. Az első karakter indexe 0.

```
string s = "Karaktersor";
char ch = s[2];                  // 'r'
s[0] = 'k';          // Hiba! Nem módosítható a sztring értéke!
```

A **Length** tulajdonság megadja a sztring hosszát. A sztringek a + operátorral összefűzhetőek (konkatenálhatók), a **ToUpper** és **ToLower** metódussal nagy– és

kisbetűssé alakíthatók. Az str() metódussal számok sztringgé alakíthatók. A sztringek összehasonlítása (== egyenlő, != nem egyenlő) során nem az általuk hivatkozott címet hasonlítjuk össze, hanem karaktereiket. A többi összehasonlító operátor nincs megvalósítva a String osztályban.

Megjegyzés:

Módosítható sztring osztály a **StringBuffer**.

Minden osztály tartalmaz egy **ToString** metódust, melynek feladata az osztály leírása. Alapértelmezett megvalósítása a típus nevével tér vissza, de ezt felülírhatjuk például úgy, hogy az adattagok értékeit is kiíratjuk vele.

? A System.String osztályról bővebb információt a .NET Framework SDK-ban találhatunk.

1.5. A láthatóságok

Az osztályokat, adattagokat és a metódusokat egyaránt négy+1 féle láthatósággal (**visibility**), hozzáférési szinttel (**accessibility**) érhetjük el. A nevek előtt UML (Unified Modeling Language) jelölésük látható.

– **Privát** (private)

\# **Védett** (protected)

Belső(internal)

\+ **Nyilvános** (public) A nevek önmagukért beszélnek.

Van még **protected internal** hozzáférési szint is (+1).

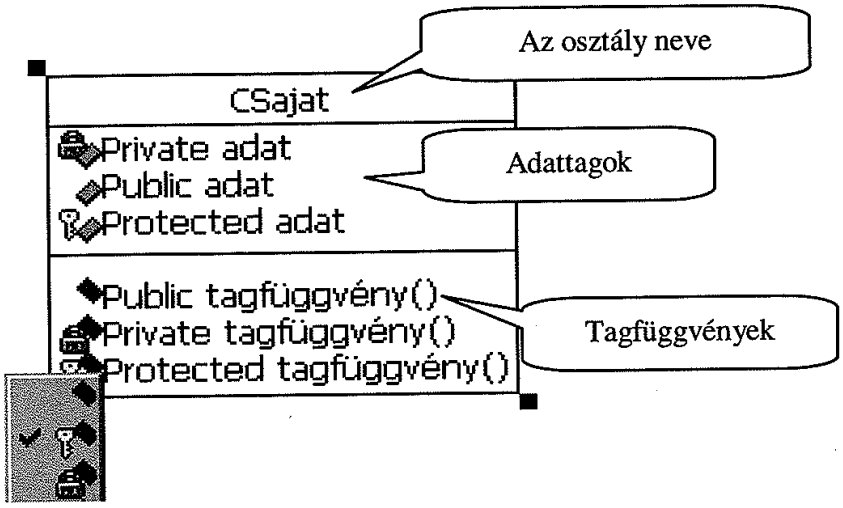
A privát tagot csak maga az osztály láthatja, ezek a tagok csak az osztály belsejéből (metódusainak blokkjából) érhetők el. Ha a kód más részében hivatkozunk rájuk, a fordító hibaüzenetet küld.

A védett láthatóságot az öröklés során (5.1.4. szakasz), a belsőt az assembly fogalmánál (7.2.1. szakasz) tárgyalja a könyv.

A nyilvános tagokat mindenhonnan elérjük, az osztályok belsejéből is és kívülről is hivatkozhatunk rájuk. Az adattagokat lehet ugyan, de az objektumorientált elvek betartása esetén nem illik nyilvánosra deklarálni.

Az **objektumorientált gondolkodásmód szerint alapértelmezésben az osztály minden tagja private.** Ez azt jelenti, ha nem írunk a kódba láthatóságot, akkor az private láthatóságot jelent. Ha a kód írója meg akarja engedni más osztályok vagy függvények számára a hozzáférést, akkor azt ki kell írnia. Ha a fejlesztőeszköz

segítségével hozzuk létre az osztály tagjait, akkor a hozzáférés alapértelmezett beállítása public, amit a combo listájából módosíthatunk.



1.5. ábra Láthatóságok jelölése

1.6. A tulajdonság fogalma

Az **egységbezárás** (**encapsulation**) az objektumorientált programozás egyik alapelve. Az elnevezés onnét származik, hogy míg a procedurális programozásban külön változókban tároltuk az adatokat, s külön függvények, eljárások dolgoztak ezekkel az adatokkal, addig az objektumorientált programozásban **egy egységben történik az adatok és a hozzájuk tartozó metódusok kezelése**.

Az **egységbezárás** lehetővé teszi, hogy az adattagokat csak tagfüggvényeken keresztül érhessük el. Ennek megvalósítását a **tulajdonság** (**property**) fogalmának bevezetésével támogathatjuk. Egy tulajdonságot olvasni (getting) vagy írni (setting) vagy mindkettőt lehet. A tulajdonság írása a vele azonos nevű adattag set metóduson keresztül megvalósuló beállítását, míg olvasása a get metóduson keresztüli értékvisszaadását jelenti. A következő kódrészlet egy Name tulajdonság megvalósítását mutatja:

```
private string name; // A tároló adattag

// Írható-olvasható tulajdonság:
public string Name
{
   get
   {
      return name;
   }
   set
   {
      name = value;
   }
}
```

A tulajdonság (nagybetűvel) a hívó kódban úgy viselkedik, mint egy tagváltozó, de értékadáskor a set metódusa hívódik, míg olvasáskor a get metóduson keresztül jutunk az adathoz.

A tulajdonság segítségével gyakran egy adattag egy részét, vagy több elrejtett adattagban tárolt adatot kezelünk.

Megjegyzés:

A készen kapott felhasználói interfész osztályaink (pl. Form, vezérlők) tulajdonságainak kezdőértékét kényelmesen állíthatjuk be a Visual Studio Properties ablakában.

Milyen előnyökkel jár a tulajdonság használata?

- Lehetővé válik **csak olvasható** (csak írható) **adat** létrehozása, ha nem írunk set (get) metódust.
- Lehetővé válik az **ellenőrzött adatbevitel**. A set metódus kódjában ellenőrzi a beírni kívánt értéket.
- Lehetővé válik az adatok praktikus tárolása mellett azok **kényelmes írása, olvasása**. A get és set metódusok kódjában átalakíthatjuk az adatokat.
- Lehetővé válik **az adattárolás módjának megváltoztatása** anélkül, hogy az osztály felhasználójának módosítani kellene a kódját. (Pl. az ezredfordulón szükségessé vált a kétjegyű évszám tárolásról a négyjegyűre történő átváltás. Y2K) Propertyvel történő megvalósítás esetén elegendő csak a tároló adattag és a get / set függvények kódjának megváltoztatása. Az osztályt használó kódot csak újra kell fordítani, nem kell újraírni, hisz az csak a public propertyvel dolgozhat, a private adattagot nem érheti el.
- Lehetővé válik összefüggő adatok több különböző adattagban történő tárolása és egységes kiolvasása. Pl. egy nevet a későbbi kényelmesebb feldolgozás érdekében lehet két mezőben, családi név és utónévként tárolni, de beolvasáskor, kiíráskor ez egyetlen adatként érzékelhető.
- Lehetővé válik ugyanarról az adathalmazról különböző módon információ kérése.
- Általunk írt vezérlő felhasználása esetén a tulajdonság megjelenik a Properties ablakban, kezdőértéke a fejlesztés során kényelmesen állítható be.

A tulajdonság használata jól megfigyelhető **DateTime** típus tulajdonságainak tanulmányozásával. A DateTime típus az – encapsulation elvnek megfelelően – számunkra ismeretlen módon tárolja az adott dátum, idő adatot. Viszont tulajdonságain keresztül kényelmesen olvasható az adat.

? A súgóban (index) a DateTime felsorolásban a properties kategóriát választva láthatjuk, hogy az év a **Year**, a hónap a **Month**, a nap a **Day**, az óra a **Hour**, a perc a **Minute**, a másodperc a **Second** tulajdonság segítségével kapható meg, de a **DayOfWeek** segítségével azt is megtudhatjuk, hogy a hét hányadik napjára esik az adott dátum (0 érték a vasárnap). A **Now** tulajdonság a gép aktuális dátumát és időpontját adja vissza egy DateTime típusú változóban.

A tulajdonság létrehozását a Visual Studio környezet támogatja.

Class View: Osztály kiválasztása **jobbegér gomb**

Add

Add Property

A párbeszédablakban megadhatjuk a kívánt értékeket. A tulajdonsághoz a tároló adattagot külön kell létrehoznunk!

Megjegyzés:

A DateTime osztály Now tulajdonsága az aktuális időt adja vissza. Ez egy statikus, vagyis osztályszintű tulajdonság, mert a feladatban szereplő időt tároló objektumoktól független értéket a gépidőt adja vissza. Értéke nem módosítható, tehát csak get metódussal rendelkezik. Lásd súgó!

Tulajdonság felhasználása már a legegyszerűbb programban is elkerülhetetlen, készítésére a 8. gyakorlat 8.1.5. szakaszában és a 9. gyakorlaton többször is láthatunk példát.

1.7. A névterek

A névterek az osztályok valamilyen logikai összefüggés szerinti csoportját jelentik. Minél nagyobb egy program, annál hasznosabbak a névterek, hogy kifejezzék a program egyes részeinek logikai elkülönülését. A névterek egymásba ágyazhatók. Például a .NET keretrendszer biztosítja számunkra a:

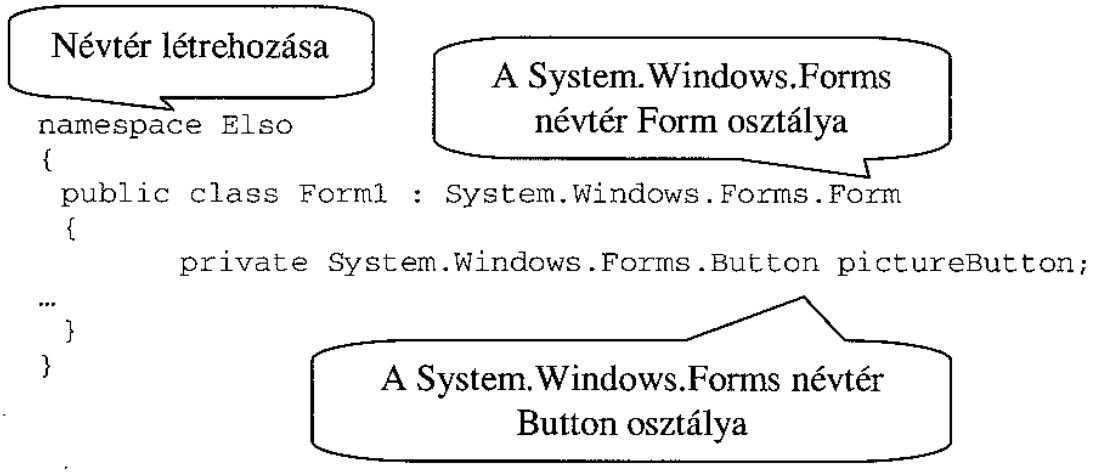
- **System** névteret, mely tartalmazza a .NET környezetben általánosan használt osztályokat. A System több kisebb névtérre bontható, melyek nevét a '.' elválasztó karakterrel adhatjuk meg. A System névtér tartalmazza közvetlenül a DateTime típust és a String osztályt.

- **System.Net** névtér biztosítja a hálózat programozásához szükséges protokollok pl. a **TCP/IP** (Transmission Control Protocol/Internet Protocol) magas szintű elérését.
- **System.Windows.Forms** névtér Windows alapú alkalmazások számára lehetővé teszi a Windows operációs rendszer által biztosított gazdag felhasználói interfész lehetőségek kényelmes elérését. Itt találjuk a formokon kívül a párbeszédablakok és a vezérlők, valamint egyéb komponensek osztályainak leírását.
- **System.IO** névteret Input/Output műveletek (Írás / Olvasás) elvégzésére az operációs rendszer fájlrendszerével kapcsolatos műveletek elvégzését támogatja. Pl. a File és Directory osztályai lehetővé teszik az alkalmazás számára fájlok és könyvtárak létrehozását, törlését, módosítását.
- **System.Data** névtér szolgáltatja az adatbázis elérések egységes felületét. Több névtérre oszlik. A System.XML névtérrel együtt támogatja osztott adatbázis-kezelő alkalmazások fejlesztését és az adatok kezelését connected és disconnected rendszerekben.

Különböző névterekben használhatunk egymással megegyező elnevezéseket is, hisz névterük megkülönbözteti őket egymástól.

Névteret a — **namespace** névtér neve — deklarációval hozhatunk létre.

A névtér osztályaira a — névtér neve.osztály neve — szintaktikával hivatkozhatunk.



A **using** névtér neve; direktíva használata lehetővé teszi hogy a névtér osztályai el ne kelljen kiírnunk a névtér nevét. Az előző kód a direktíva használatával a következőre módosulhat:

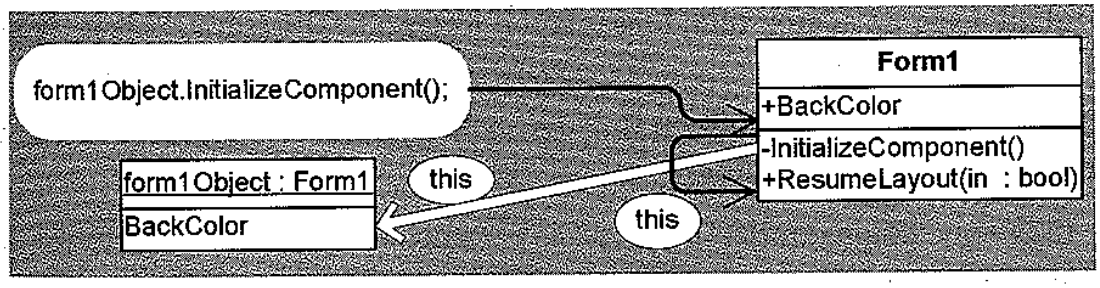
```
using System.Windows.Forms;

namespace Elso
{
 public class Form1 : Form
 {
        private Button pictureButton;
 }
}
```

1.8. A this paraméter

Az osztály csak az adott típus leírását tartalmazza, az osztályban nem történik helyfoglalás az adatok számára. A helyfoglalás az objektum definiálásakor történik meg. Ez természetes is, hisz egy adott osztályból több különböző értékű objektumot hozhatunk létre. Honnét tudják az osztály leírásában szereplő függvények, hogy most éppen melyik objektumon dolgoznak?

A metódushívások (ha nem statikusak) **objektumból indulnak**. Az lehet,' hogy egy metódust egy másik metódus belsejéből hívunk meg, de a hívás itt is objektumból indult, mi csak továbbadtuk. A metódusok a kódot az osztály leírásából veszik, és van egy **láthatatlan paraméterük: a this**. A this az aktuális objektumra hivatkozik. A függvény végrehajtásakor az adatokat a this által hivatkozott a címről veszi, a this által hivatkozott adatterületen dolgozik. Ha a függvény belsejéből metódust hívunk, a this továbbadódik. A következő kódrészletben a Form1 osztály InitializeComponent metódusa paraméter nélkülinek látszik, de kódjában mégis tudunk hivatkozni a this-re. A this a látszat ellenére a ResumeLayout metódusnak is továbbadódik.

```
private void InitializeComponent()
{
 this.BackColor = System.Drawing.Color.LightBlue;
 this.ResumeLayout(false);
}
```



1.6. ábra A this szerepe a függvényhívásban

C# kódban a thist és a névterek hivatkozását explicit módon ki szokás írni, talár forráskód szerkesztő szöveg kiegészítő támogatása miatt. Vagyis ann köszönhetően, hogy a this. után a szövegszerkesztő felkínálja az adott oszt metódusait, tagváltozóit és tulajdonságait (**IntelliSense**). Ezek többnyire hoss nevűek, de néhány kezdőbetű leütése után a név kiegészül, a kis- és nagybet helyesen kiíródnak Enterre, vagy még gyorsabb, ha kódban a nevet követő operáto ';' '.' '(') karaktert ütjük le. Tehát a this kiírásával gyorsabban írható hibátlan kó mint elhagyásakor. A fenti kódban mindkét this elhagyható lenne, és mivel a usi System.Drawing; a fájl elején szerepel ez is, elhagyható lenne a Color elől. Bá kód a kiírásokkal gyorsabban írható, de nehezebben olvasható, különösen egymás ágyazott függvényhívások esetén. A könyvben ritkán szerepelnek fölöslegesen ki tagok.

A kódgenerátor mindig kiírja a névterek hivatkozását és a this referenciát.

Megjegyzés:

A this vagy a névtér hivatkozás elhagyása esetén a kezdőbetűk ut a Ctrl + szóköz billentyűk segítségével hívhatjuk elő a legördü súgót (**IntelliSense**).

1.9. Teszt

1. A névtér a logikailag összetartozó osztályok csoportját jelenti.
2. Statikus metódusnak nincs this paramétere, mert nem objektumból indul hívása.
3. A 'using névtér neve;' után már ne írjuk ki a kódban a névtér nevét, mert az névtér dupla hivatkozását jelenti, és hibaüzenethez vezet.
4. Statikus metódusnak nincs visszatérési értéke.
5. Az osztály tagjai alapértelmezésben publikusak, ha el akarjuk őket rejteni, eléji kell írnunk pl. a private kulcsszót.
6. A tulajdonság az adatok tulajdonságlapon történő beállítására szolgál. A kódb értéke nem változtatható.
7. A változókat kötelező inicializálni, ha nem akarunk kezdőértéket adni, akk null értéket kell beállítanunk.
8. A függvény szignatúráját a neve, a paraméterlista és a visszatérési értél határozza meg.
9. A tulajdonságok használata lehetővé teszi a háttéradatok különböző módo történő felhasználóbarát kiolvasását.

10. A tulajdonság egyik előnye a public adattaggal szemben, hogy értékadáskor a set metódusban ellenőrizni lehet a bevitt adatok helyességét.
11. Két különböző láthatóságú függvénynek lehet azonos a szignatúrája.
12. A string típusú változó referencia típusú, így csak a szövegek helyének címét tartalmazza.
13. int a=5; int b=a; a=7; esetén b értéke 5 marad, mert az int érték típusú változó.
14. string a="alama", string b=a; a="körte"; esetén b értéke „körte" lesz.
15. Publikus tagokhoz mindenhonnan hozzáférhetünk, míg private tagok csak az osztály tagfüggvényeiből láthatóak.
16. A függvény átdefiniálás azt jelenti, hogy azonos nevű, de más paraméterlistájú függvényt készítünk.

Javítókulcs:

I, I, H, H, H, H, H, H, I, I, H, I, I, H, I, I.

2. Az alkalmazás vezérlése

2.1. A Main

Minden alkalmazásnak kezdődnie kell valahol. Az alkalmazások végrehajtása a **Main** függvény hívásával indul, és a Main befejeztével végződik. Egy alkalmazás több osztályt is tartalmazhat, de belépési pontja csak egy lehet.

Megjegyzés:

A C# nyelv case sensitive, vagyis megkülönbözteti a kis- és nagybetűket. A Maint és általában a metódusok, valamint osztályok nevét nagybetűvel, a változók neveit kisbetűvel szokás kezdeni. A fordító nem engedi a szóközök használatát a nevekben, így az összetételek mentén nagybetűvel szokás az új szót írni, ezzel segítve a hosszú szavak olvashatóságát. Pl. InitializeComponent.

A Maint a Visual Studio automatikusan static metódusként hozza létre. Mivel a Main indítja az alkalmazást, még bármely objektum létrehozása előtt meghívhatónak kell lennie.

2.2. Az eseménykezelés

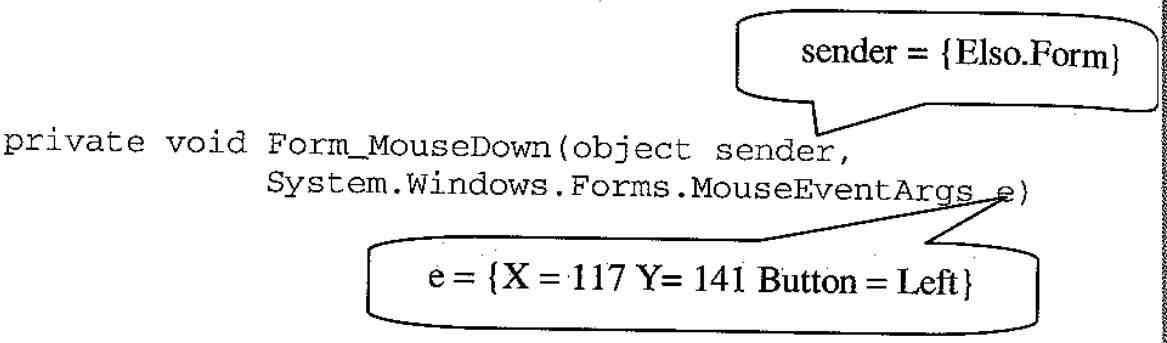
A Windows operációs rendszer biztosít az összes rajta futó alkalmazás számára egy eseménykezelő (üzenetkezelő) mechanizmust, mely során a bekövetkező **események** (**event**) hatására üzenetet küld. A programozó feladata az eseményhez eseménykezelő metódus kapcsolása, mely megadja az eseményt kezelő kódot.

Pl.

Lenyomjuk a baloldali egérgombot a Form felett.

Esemény: **MouseDown**

Metódus:



A metódus a Form osztály tagfüggvénye, és első paramétere tartalmazza a küldő objektum adatait, míg második paramétere az esemény adatait.

A formot, vagy egy konkrét vezérlőt kiválasztva, a Properties ablak Events eszközgombján kattintva az adott vezérlő által kezelhető eseményeket nézhetjük meg.

Hozzárendelés a gyakorlatban:

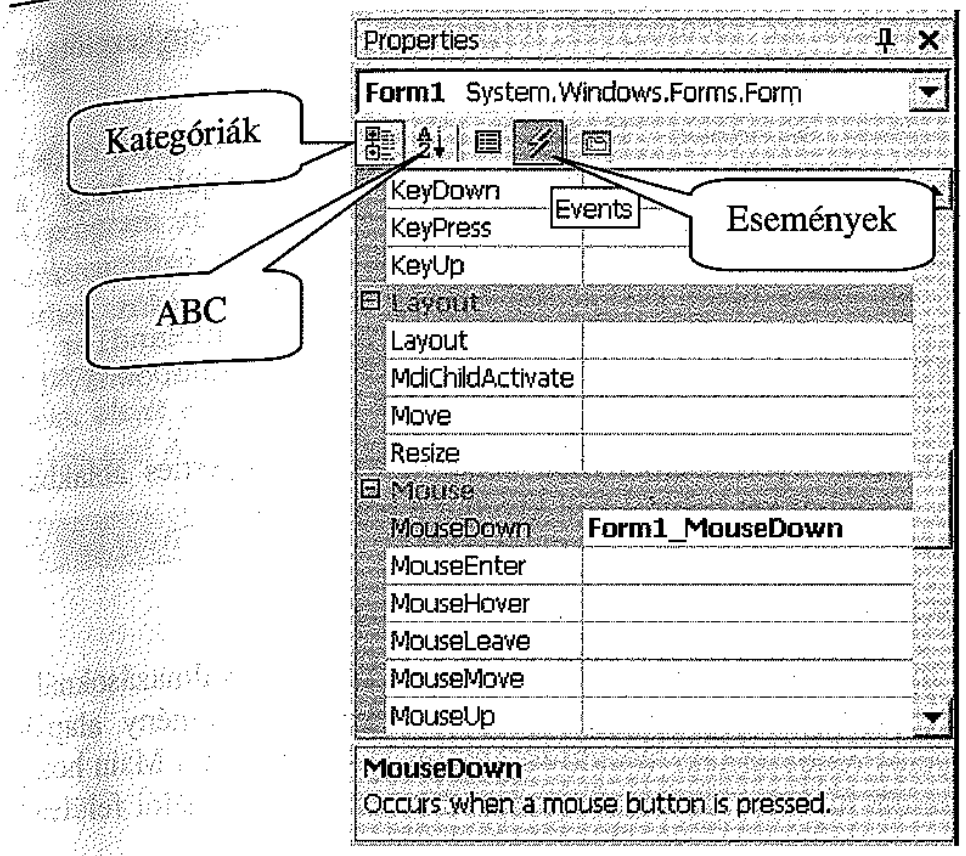
Legyen aktuális a Form

Properties ablak

Esemény gomb

Nézhetjük abc sorrendben,

Kategóriák szerint csoportosítva.



2.1. ábra Eseménykezelő metódus létrehozása

A kezelni kívánt esemény nevén duplán kattintva létrejön a kezelőmetódus, a kurzor pedig a metódus kódjához ugrik.

Ha mi akarjuk megadni a kezelőfüggvény nevét, egyszerűen begépelhetjük az esemény neve mellé a tulajdonságablakban.

2.2.1. Az alapértelmezett esemény kezelése

Minden vezérlőhöz tartozik egy általánosan kezelt (default) esemény, ezt az adott vezérlőn a tervezőablakban duplán kattintva generálja a Visual Studio környezet. Ilyen a form esetén a Load, míg a gomb esetén a Click esemény. Ezzel a gyors lehetőséggel élve nem választhatjuk meg a kezelőfüggvény nevét, azt az objektum nevéből és az esemény nevéből rakja össze a kódgenerátor: Form_Load, HatterszinButton_Click.

A kezelő létrehozásakor a kurzorunk most is a kezelőfüggvény első sorában villog, lehetővé téve a kód írását.

2.2.2. Az eseménykezelő kódja

Az eseménykezelő létrehozása során a fejlesztőeszköz két helyre ír bejegyzést, egyrészt a függvényt írja be a form osztály tagfüggvényei közé, annak blokkjába

írjuk a kódunkat. Másrészt az eseményhez hozzárendeli az eseménykezelőt az EventHandler metódus segítségével.

A kódban a `'#region Windows Form Designer generated code'` régió kinyitásával az **InitializeComponent**függvényben láthatjuk, hogy a MouseDown eseményhez kezelőt rendeltünk.:

```
this.MouseDown +=
            new System.Windows.Forms.MouseEventHandler
                                (this.Form_MouseDown);
```

Ha véletlenül hozunk létre ilyen eseményhez kezelőt, azt a Properties ablakban célszerű törölni, vagy a kódból két helyről kell kivenni a bejegyzést.

2.3. A program szerkezete

A program a **Main** függvénnyel indul. A Main feladata konzol alkalmazásoknál a parancssori paraméterek fogadása, feldolgozása, és a kapott eredmény alapján visszajelzés az operációs rendszernek. Windows alkalmazások esetén a Main hozza létre az eseményvezérelt futtatást biztosító háttérkódot, és a felhasználói felületet, amin keresztül a felhasználó a programmal kommunikálni tud.

Az **Application** osztály statikus **Run** metódusa biztosítja az aktuális szál üzenetkezelő ciklusát. A Run paramétereként jön létre a főablak –ami egy form –, s melynek InitializeComponent metódusa tartalmazza az ablakra vonatkozó egyedi információkat.

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

A főablak létrejötte után a kódban már nincs következő utasítás, a Run metódus elindítja az üzenetkezelést, az alkalmazás üzenetre vár, s ha az adott üzenethez van kezelő metódus, akkor azt hívja meg.

A kilépés üzenet (Bezárás ikon, vagy a menü Bezárás menüpontja) hatására a Run futása befejeződik, és az alkalmazás végrehajtása véget ér.

! A gyakorlat során feltétlen hajtsa végre a debuggerrel a programot lépésenként!

2.4. A vezérlőszerkezetek

2.4.1. A szekvencia

A program egymás után írt utasítások sorozata. Az egymást követő utasításokat szekvenciának nevezzük. Az utasításokat ';' választja el egymástól. Új utasítást új sorban szokás írni. Az utasítások végrehajtása írásuk sorrendjében történik.

A program egymáshoz logikailag kapcsolódó utasításait egy **blokk**ba szokás írni. A blokk elejét a '{', végét a '}' jelzi. A blokk végére nem teszünk ';'-t, mert a '}' lezárja azt! A blokkok egymásba ágyazhatók.

Megjegyzés

Ha a blokk kezdetét és végét új sorba írjuk, a szerkesztő egymás alá igazítja őket, s így jobban áttekinthető kódot kapunk. Ha a blokkot lezárjuk, a '}' épp a '{' alá kerül, a blokk belsejéhez tartozó kódsorokat pedig egymás alá rendezi a szerkesztő. Egy rövid ideig még a blokk eleje és vége vastagon szedve jelenik meg, hogy jobban lehessen látni az összekapcsolódó részeket.

Minden blokk egy **hatókör** (szkóp, **scope**). A blokkban definiált változót **lokális változó**nak (**local variable**) nevezzük. A lokális változó a definíciójától a blokk végéig látható. Az egymásba ágyazott blokkok belsejéből látjuk a külső blokkok változóit, így a fordító nem engedélyezi a külső blokkal azonos nevű változó használatát a belső blokkban.

```
int i = 3;
{
 int i = 5;    // Hiba, azonos nevű változó már szerepel a
               //hatókörben.
}
```

Egymástól elkülönülő blokkokban létrehozhatunk azonos nevű változókat.

```
{
int i = 3;
}
{
int i = 5;
}
```

Célszerű a változókat az őket felhasználó blokkban definiálni. Ez futáskor hatékonnyá, fejlesztéskor pedig áttekinthetővé teszi a programot.

2.4.2. Az elágazás

Egyágú elágazást használunk, ha egy utasítást csak bizonyos feltétel teljesülése esetén hajtunk végre. Tehát az elágazásban szereplő utasítás(ok) nem minden esetben hajtódnak végre.

```
if (logikai kifejezés) utasítás vagy utasítás blokk.
```

pl.:

```
if (jelszoBox.Text != "AAA")   Application.Exit();
```

Azonosság megállapításához az '==' és a '!=' operátorokat használjuk. Az '=' operátor értékadást végez!

Kétágú elágazásról beszélünk, ha egy feltétel teljesülése esetén valamit, míg ellenkező esetben mást tesz a program. Itt az elágazás valamelyik ágán szereplő utasítás (sorozat) biztosan végrehajtódik.

```
if (logikai kifejezés)
        utasítás vagy utasítás blokk
else
        utasítás vagy utasítás blokk
```

Megjegyzés:

Egész típus nem írható a C++-ban megszokott módon a logikai kifejezés helyére. Explicit ki kell írnunk: `if ( i != 0 )`.

Többágú elágazás megvalósításához a C# nyelv két lehetőséget is kínál. Az if, else if technikával tetszőleges számú egymást kizáró feltételt vizsgálhatunk, az else if ágak száma határozza meg az elágazások számát Az utolsó else ág léte vagy elhagyása határozza meg, hogy biztosan végrehajtódik-e valamelyik ág, vagy a feltételek nem teljesülése esetén a program a következő utasításon folytatódik.

```
if (logikai kifejezés)
        utasítás vagy utasítás blokk
else if (logikai kifejezés)
        utasítás vagy utasítás blokk
else if (logikai kifejezés)
        utasítás vagy utasítás blokk
else
        utasítás vagy utasítás blokk
```

Vegyük észre, hogy a fenti elágazás nem azonos az egymás után írt if elágazásokkal! Míg a fenti esetben az else if ág vizsgálata csak az if feltétel nem teljesülése esetén

történik meg, addig egymás után írt if utasítások mindegyikében megtörténik a feltétel vizsgálata, tehát akár az is előfordulhat, hogy minden feltétel teljesül és mindegyik utasítás blokkja végrehajtódik. Egymást kizáró feltételek esetén az else if kód a hatékonyabb.

Az if utasítások egymásba ágyazhatók. Az else if utasítás is tekinthető az else ágba ágyazott újabb if utasításnak, de megtehetjük, hogy az igaz ágban is újabb feltételt vizsgálunk pl.

```
if (jelszoBox.Text != "AAA")
 if (++missed == 3) Application.Exit();
else
 label.Text = "Helyes jelszó!";
```

A ++ operátor eggyel növeli, míg a -- operátor eggyel csökkenti a változó értékét. Mindkettő állhat a változó előtt vagy mögött.

A ++ (increment) operátort a numerikus változó előtt használva (prefix) a művelet végeredménye az 1-gyel növelt változó értéke. A változó után írva (postfix) a ++ operátort a művelet eredménye a változó értéke a növelés előtt. Jelen helyzetben tehát az összehasonlításban a növelt érték vesz részt, míg missed++ esetén még a növelés előtti értéket hasonlítanánk össze, majd azután növelnénk a missed értékét.

Mikor hajtódik végre az else ág?

Az else alapértelmezésben a legbelső if párja. Ha ettől el kívánunk térni, blokkok létrehozásával megtehetjük ezt. Hiába írtuk tehát a külső if-fel egy vonalba, a fenti kód else ága minden olyan esetben végrehajtódik, amikor még nem érte el a missed a 3 értéket. A kódrészlet helyesen:

```
if (jelszoBox.Text != "AAA")
{
 if (++missed == 3) Application.Exit();
}
else
 label.Text = "Helyes jelszó!";
```

A feltételek kiértékelését a C# feltételes logikai operátoraiban (&&, ||) oly módon optimalizálták, hogy ha a kifejezés értéke már egyértelmű, a kiértékelés nem folytatódik. Így egy AND (&&) művelet esetén, ha a baloldali paraméter hamis, a kifejezés értéke már biztosan hamis lesz tehát a jobboldali érték nem értékelődik ki. Hasonló módon OR (||) esetén a baloldali operandus igaz volta eredményezi az operáció biztosan igaz értékét, így a jobboldali operandus kiértékelésére nem kerül sor. Ezt kihasználva a programozóknak nem kell például egymásba zárt

elágazásokat, ciklusokat írniuk olyan esetekben, amikor a baloldali operandus értékétől függ a jobboldali kiértékelhetősége. Írhatják egy kifejezésbe a két feltételt. Ez növeli a kód áttekinthetőségét.

```
if (textBox.Text.Length > 0 && textBox.Text[0]=='$')
 textBox.Text = "Dollár!";
else textBox.Text = "";
```

Figyelem! A bitenkénti logikai (&, |) operátorok mindkét operandust kiértékelik.

A **switch case** utasítás áttekinthetőbb többágú elágazások írását teszi lehetővé, de használata korlátozott. A switch kulcsszó utáni kifejezés típusa csak valamilyen **egész típus**, **char**, **enum** vagy **string** lehet – vagy olyan objektum aminek van implicit cast operátora[1] ezen típusok egyikére –, a case kulcsszó mögé pedig nem írhatunk értéktartományokat, csak egyedi értékeket.

Megjegyzés:

A C, C++ és Java nyelvekkel ellentétben a switch után használhatjuk a string típust is, ilyenkor a case értéke akár null is lehet.

A switch case elágazás tartalmazhat egy default kulcsszót is, mely utáni blokk akkor hajtódik végre, ha egyetlen case utáni értékkel sem egyezett meg a változó.

```
switch (jelszoBox.Text)
{
 case "Anna":
        label.Text="Helyes jelszó";
        break;
 case null:
        label.Text="Még nincs jelszó";
        break;
 case "":
        label.Text="Még nincs jelszó";
        break;
 default:
        label.Text="Hibás jelszó";
        break;
}
```

[1] A castolásról, konverzióról az Öröklésnél, az 5.3.2.szakaszban olvashatunk bővebben.

Amíg az egyes ágak végére nem írjuk be a 'break' kulcsszót – a fordító hibaüzenetet küld, miszerint az ellenőrzés nem eshet át az egyik 'case' címkéről a másikra. Tehát az elágazásnak pontosan egy ága fog végrehajtódni.

2.4.3. A ciklus

Egy utasításblokk többszöri végrehajtására a C# nyelv több lehetőséget is kínál. A ciklusokat feltételes és léptető kategóriába soroljuk.

2.4.3.1. A feltételes ciklusok

Végrehajtásuk egy logikai kifejezés igaz voltától függ. A while után szereplő feltétel kötelezően logikai kifejezés. A C++-tól eltérően nem engedélyezett az int érték használata. A blokkot határoló kapcsos zárójelek egy utasítás esetén elhagyhatók.

Elöl tesztelős ciklus:

```
while (logikai kifejezés)
{
  utasítások;
}
```

A ciklus blokkjában szereplő utasítások mindaddig végrehajtódnak, amíg a feltétel igaz. A blokk végére érve a feltétel újra kiértékelésre kerül. A feltétel hamis értéke esetén a blokk nem hajtódik végre. **Ha a feltétel már az első alkalommal hamis, a blokk utasításai egyszer sem hajtódnak végre**, a vezérlés a blokk utáni első soron folytatódik.

Hátul tesztelős ciklus:

```
do
{
  utasítások;
}while(logikai kifejezés);
```

Az utasítások végrehajtása után a logikai kifejezés igaz értéke hatására a vezérlés újra a do utáni sorra kerül, hamis érték esetén a következő soron folytatódik. **Hátul tesztelős ciklus egyszer biztosan végrehajtódik.**

Megjegyzés

Ne feledjük a logikai kifejezésben szereplő változók értékét a kiértékelés előtt beállítani és a ciklus blokkjában módosítani, mert könnyen végtelen ciklust készíthetünk!

Míg elöl tesztelős ciklus esetén a while feltétele utáni ';' üres ciklust jelent, addig a hátul tesztelős ciklusnál kötelező a ';' a while feltétele után.

2.4.3.2. A léptetős ciklusok

A léptetős ciklusok gyakran a blokk adott számú végrehajtását jelentik. Mivel a ciklus fejében összefoglalva szerepel a belépés, kilépés és léptetés, elkerülhető a feltételes ciklus gyakori hibája: a ciklusváltozó módosításának elfelejtése.

```
for (ciklusváltozó(k) kezdőértéke; bentmaradás feltétele;
ciklusváltozó(k) módosítása)
{
 utasítások;
}
```

Például:

```
for (int i = 0; i < 10; i++)
{
 label.Text += i;
}
```

Megjegyzés:

Bár a léptetést biztosító utasítás fizikailag a blokk előtt helyezkedik el, végrehajtására csak a blokk vége után kerül sor.

Egy for ciklusban több változót is létrehozhatunk, ha azonos típusúak, és külön-külön léptethetjük őket.

```
for (int i = 0, j = 0; i + j < 15; i++, j +=3)
{
 label.Text += i;
}

for (int i = 0, uint j = 0; i + j < 15; i++, j +=3) //Hiba!
```

A hibát az okozza, hogy a két ciklusváltozó csak azonos típusú lehet, és emiatt csak egyszer szabad kiírni a típust.

A ciklusban létrehozott változók (i, j) a ciklus blokkján kívül már nem érhetők el. Tilos a for ciklust tartalmazó blokkban a ciklusban használttal azonos nevű változókat létrehozni, mert azok a ciklus belsejében elérhetők, és névütközés lépne fel! Egymás után végrehajtott for ciklusok viszont használhatnak azonos nevű változót a léptetéshez.

```
for (int i = 0; i < 10; i++) {…}
for (int i = 5; i > 0; i--){…}
for (int i = 0; i < 10; i++){..}
```

2.4.3.3. A foreach ciklus

Gyakran előforduló feladat, hogy egy tömb, vagy más elemeket összegyűjtő objektum minden elemére kell valamit végrehajtani. Ezt segíti a foreach. Mi most csak az egyszerű tömb esetét mutatjuk be.

Pl. egy számokból álló tömb elemei közül meg akarjuk határozni a 10-nél nagyobbak számát.

```
int[] nums = new int [] {0,15,23,5,-71,10,11};

int big = 0;
foreach (int i in nums)
{
 if (i>10)
       big++;
}
```

A foreach használata során a tömb minden elemén végigmegyünk, így csak akkor hatékony, ha ez szükséges. Akkor viszont egy egyszerű, áttekinthető és gyors megoldási lehetőséget kínál.

A ciklusváltozó csak olvasható. (Tehát ha big++ helyett i++-t írunk, az hibát okoz.)

Megjegyzés:

A fenti kódrészlet a feltételes megszámlálás programozási tétele. (Lásd a programozási tételek fejezetben!)

2.4.4. További vezérlő szerkezetek

Rendelkezésünkre állnak még, talán a hagyományőrzés miatt a goto, continue és a break vezérlőszerkezetek.

- **goto**: címkére ugrik.
- **continue**: a ciklus vagy elágazás végrehajtását újrakezdi (switch, while, do, for, foreach).
- **break**: a ciklus vagy elágazás végrehajtását megszakítja és a következő utasításra ugrik. A switch case elágazás case ágain a break utasítás akadályozza meg az 'átcsordulást', vagyis azt, hogy az egyik case ág után a következő is végrehajtódjon.

E vezérlőszerkezetek használata nagyon áttekinthetetlenné teszi a programot, így nem javasolt. A programok bonyolultságának kezelésére kialakult strukturált programozás kifejezetten tiltja használatukat. A nyelvek a korábbi verziókkal történő kompatibilitás megőrzése érdekében tartották meg ezeket az elemeket.

2.5. A kivételkezelés

A **kivételkezelés** (**exception handling**) programozási filozófiája azt jelenti, hogy koncentráljunk a program várható helyes végrehajtására, és a ritkán előforduló eseteket a kivételeket külön technikával kezeljük le. Pl. egy fájl megnyitása várhatóan zökkenőmentes feladat, azonban ha a megadott útvonal hibás (mert áthelyezték egy másik könyvtárba és megszüntették a régit), vagy nincs ilyen nevű fájl (letörölték), esetleg már megnyitotta valaki más, vagy nincs jogunk hozzáférni – ezek kivételes helyzetek, melyeknek előfordulására mégis gondolnia kell a programozónak. Tehát kezelni kell a helyzetet, de úgy, hogy a kód logikáját ne zavarja ezeknek az eseteknek a vizsgálata.

2.5.1. A kivétel dobása

Ha a program végrehajtása nem várt akadályba ütközik, a **throw** kivételobjektum segítségével dobhatunk kivételt.

```
if ((month <0) || (month>12)) throw new SytemException("Nincs
ilyen hónap");
```

Ha nem kapjuk el a hibát, az alapértelmezett kivételkezelő gyakran valamilyen hibaüzenet után leállítja a programot. C#-ban a throw után egy objektumot szokás továbbadni, mely a hibáról több információt tartalmazhat. Pl.

```
throw new FileNotFoundException(fájlnév);
```

Ha a kivételt dobó sort egy **try** blokkba helyezzük. A kivételdobás utáni sorok a kivétel bekövetkeztével nem hajtódnak végre, a vezérlés a kivételt elkapó kódra kerül.

2.5.2. A kivétel elkapása

A kivétel elkapása **catch** blokkal történik. Paraméter nélküli catch blokk minden kivételt elkap, de mivel nem veszi át a throw által dobott változót, így nincs konkrét információnk a kivétel okáról.

```
try
{
if ((month <0) || (month>12))
                throw new SytemException("Nincs ilyen hónap");
További kódsorok;
}
```

```
catch
{
 label.Text = „Hibás dátum";
}
```

A catch paraméterezhető.

Ha a catch blokknak van paramétere, akkor csak a paraméter típusának megfelelő kivételt tudja elkapni. Egymás után több catch blokk is írható különböző típusú paraméterekkel. Ha az első catch nem kapta el, a következő lekezelheti a hibát. Értelemszerűen, ha használjuk a paraméter nélküli catch blokkot, az csak a lista végére kerülhet, hisz minden kivételt elkap. Ha egy kivételt lekezeltünk, a kód az utolsó catch blokkja utáni soron folytatódik.

Ha egy kivételt elkaptunk, de csak bizonyos esetekben tudjuk kezelni, lehetőség van a catch blokkból egy paraméter nélküli throw segítségével az elkapottal azonos értékű kivételt dobni, ami lehetővé teszi a további kezelést.

A kivételkezelés egymásba ágyazható, és a try blokkból hívott függvények is dobhatnak kivételt. A súgó a függvények ismertetése során a függvény által dobott kivételek típusáról is tájékoztat.

```
try
{
 System.IO.File.Open("C:\\myFile.txt",
                     System.IO.FileMode.Open);
}
catch(System.IO.FileNotFoundException fnf)
{
 label.Text = "Nincs "+fnf.FileName+" fájl!";
}
```

Megjegyzés

A try blokkhoz tartozhat egy **finally** klauza is, ami mindig végre lesz hajtva, akár kivételdobás nélkül hajtódott végre a blokk, akár kivétellel. Ide szokás írni olyan tevékenységek kódját, mint a lefoglalt erőforrások felszabadítása vagy a fájlbezárások.

```
System.IO.FileStream file = null;
try
{
      file = new System.IO.FileStream(
      "C:\\myFile.txt", System.IO.FileMode.Open);
}
catch(System.IO.FileNotFoundException fnf)
{
      label.Text = "Nincs "+fnf.FileName+" fájl!";
}
```

```
finally
{
        MessageBox.Show("Finally");
        if (file != null)
              file.Close();
}
```

A 6.1.1 gyakorlati alkalmazása a 'Folyamat indítása statikus metódushívással' fejezetben látható.

Kivételkezelés alkalmazásakor a kivételek elkapása is meghatározhatja a kód végrehajtási sorrendjét.

A program szerkezetét befolyásolhatja a **return** utasítás függvény vége előtti, gyakran a függvényen belüli többszöri, esetenként más-más visszatérési értékkel történő kiadása.

Eseményvezérelt programozásnál a program a Quit esemény bekövetkezésének hatására fejeződik be és hajtódik végre a **Main** függvény kódjának Application.Run utáni szakasza. Az alkalmazás bezárása létrehozható például valamely tagfüggvény belsejében kiadott Application.Exit(); függvényhívással is.

2.6. Teszt

1. Mit látunk a képernyőn a következő kódrészlet végrehajtása után?

```
int i = 10;
if (i<10)
 label.Text = "JÓ";
 if (i>0)
 label.Text += " VÁLASZ";
else
 label.Text += "!";
```

2. Mit látunk a képernyőn a következő kódrészlet végrehajtása után? Írja le másképp a kódot!

```
int i = 10;
while (i>0)
{
 i--;
 label.Text +=i;
}
```

3. Mennyi lesz i értéke a következő kódrészlet végrehajtása után?

```
i=0;
```

```
int[] arr = new int[]{0,1,2,3,4,5,6,7,8,9,10};
foreach(int j in arr)
{
 if (j>3)i++;
}
```

Javítókulcs:

VÁLASZ; 9876543210; for(int i=9;i>=0;i--) label.Text+=i; 7

3. A vezérlők, tulajdonságok, események, ablakok

3.1. A Properties ablak és kezelése

A form és a vezérlők kezdő tulajdonságait a **Properties** (tulajdonság) ablakának beállításával határozhatjuk meg. Az ablak általában a Visual Studio fejlesztőeszköz jobb alsó sarkában található, vagy az ott elérhető megfelelő fül kiválasztásával nyitható ki.

Ha nincs nyitva az ablak, a View menü Properties Window menüpontját választva, vagy a Standard eszközsor Properties eszközgomján  kattintva érhetjük el.

Az ablak tetején a legördülő listában kiválasztható a jelenleg tervezés alatt álló ablaknak és vezérlőinek azonosítója, de a kiválasztást többnyire a tervezőablak (Design) megfelelő elemére kattintva végezzük el.

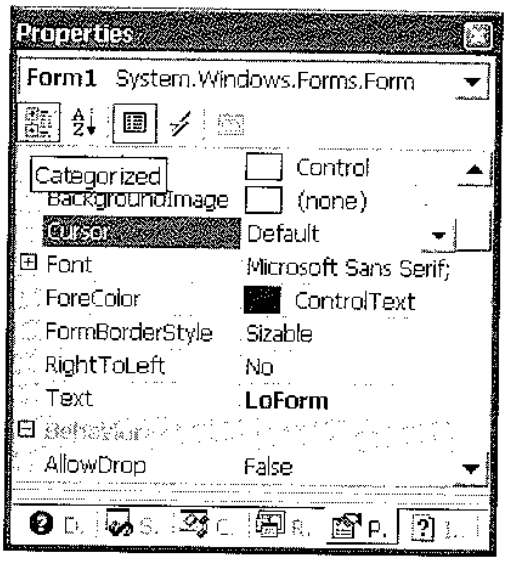
A tulajdonságablak egy eszközsort is tartalmaz a következő eszközgombokkal:

- Categorized: Logikai kategóriákba csoportosít.
- Alphabetic: ABC sorrendben mutat.

Ez a két választás értelemszerűen egymást kizárja.

- Properties: A tulajdonságokat mutatja.
- Events: A kezelhető, kezelt eseményeket mutatja a kezelőmetódusukkal együtt.

E két kategória is egymást kizárja, és e választástól függetlenül az előző kategória módosítható.



3.1. ábra A tulajdonságablak tulajdonságai, kategória szerinti csoportosításban

A módosított tulajdonság értéke vastagon szedve jelenik meg (LoForm az ábrán). Egyes tulajdonságok legördülő listából választhatók (Cursor), mások egy párbeszédablak segítségével állíthatók be (Font, szinek). Az egyes tulajdonságcsoportok becsukhatók (Font) vagy kinyithatók (Behavior) az előttük megjelenő kis + – ikon segítségével.

A kiválasztott tulajdonság rövid leírása elolvasható, ha az ablak gyorsmenüjében (jobbegér) a Description menüpontot kiválasztjuk.

3.2. A vezérlők

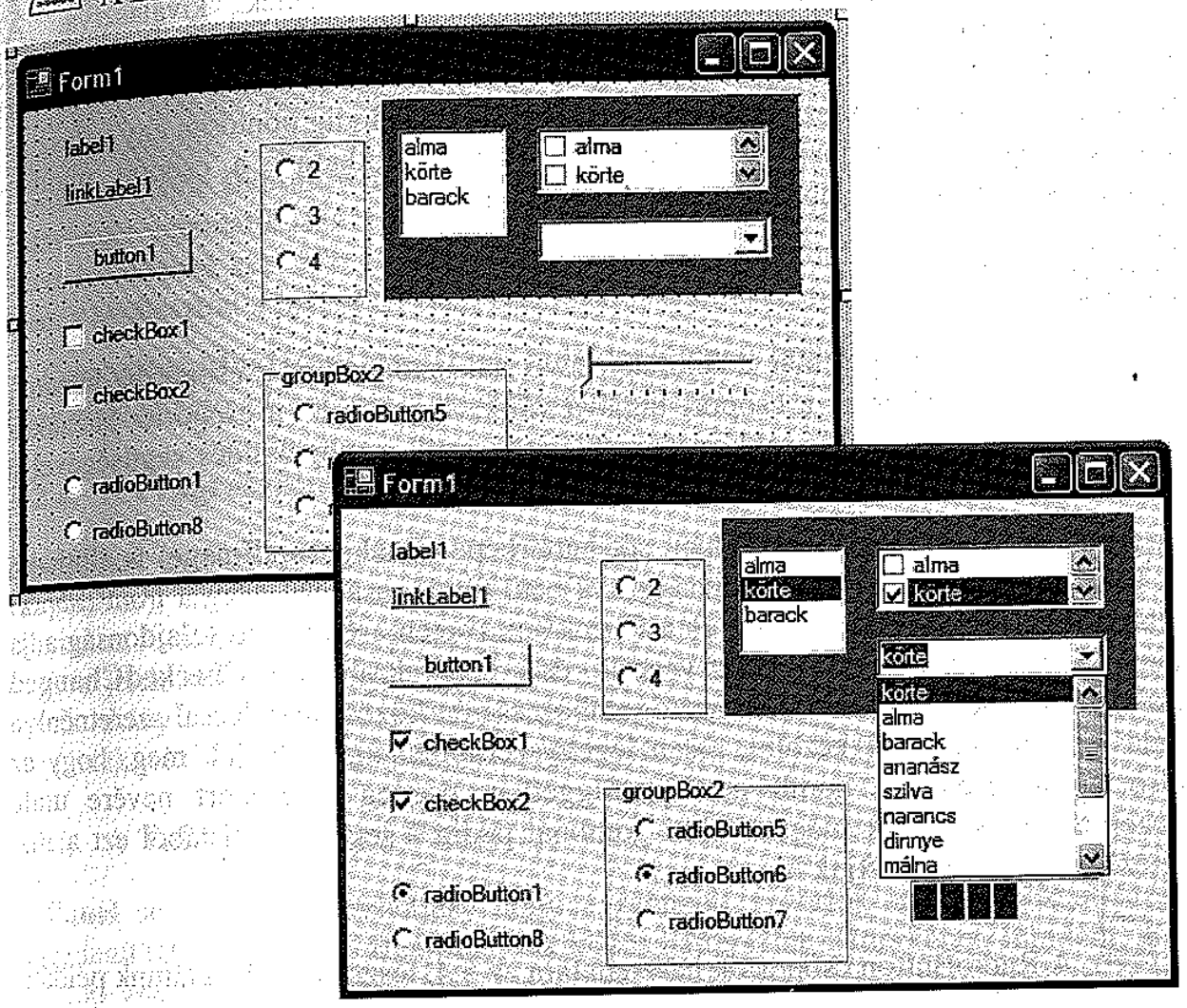
A teljesség igénye nélkül vizsgáljunk meg néhány rendelkezésünkre álló vezérlőosztályt!

3.2.1. A Label és a LinkLabel

Az eddigiekben már találkoztunk a Label vezérlővel, melyet feliratok készítésekor használunk. Location és Size tulajdonsága beállítódik amikor elhelyezzük a formon. Jellemzően a Text tulajdonságát állítjuk be. Időnként speciális fontot is megadhatunk. A Label vezérlőket ritkán módosítjuk a kódból, és eseményt sem szoktunk hozzájuk csatolni, így nincs jelentősége, hogy beszédes változónevük legyen. Emiatt ez az a kivételes eset, amikor ritkán módosítjuk a kódgenerátor által

megadott Name tulajdonságot. Különleges formája a LinkLabel, mely a böngészőkben megszokott hiperhivatkozások megvalósítását támogatja. **LinkVisited** tulajdonsága lehetővé teszi, hogy tároljuk, volt-e már látogatva a link. Színe változtatható a link állapotától függően, pl. hogy választottuk-e vagy sem.

A LinkLabel használatára a 7. gyakorlaton látunk konkrét példát.



3.2. ábra Vezérlők a szerkesztőben és futásidőben

3.2.2. A Button, a RadioButton, a GroupBox és a CheckBox

Találkoztunk már a **Button** vezérlővel is, mely parancsok kiadásra használatos. Text tulajdonsága választásakor végrehajtott parancsra utal. Többnyire a **Click** eseményét kezeljük.

A **választógomb**ok (**RadioButton**) segítségével a felsoroltakból pontosan egy elemet választhatunk ki egyszerre, míg a **jelölő négyzet**ek (**CheckBox**) külön-külön választható tulajdonságokat jelentenek. Ez a különbség a megjelenésükben is

jelentkezik, így első pillantásra tudjuk, hogy pontosan egy vagy akárhány elemet választhatunk egymástól függetlenül. Ha csak egy csoportba tartozó választógombjaink vannak, akkor nem szükséges őket külön csoportba foglalni (radioButton1, RadiButton8), de több csoport esetén egy **GroupBox** vezérlővel határozhatjuk meg, mik az egy csoporthoz tartozó elemek (2, 3, 4). A GroupBox a felületen lehet egy egyszerű keret, de lehet a csoportnak felirata is. **Checked** tulajdonságuk alapértelmezésben False, vagyis nincsenek kiválasztva. A választógombok csoportjában — ha másképp nem állítjuk a Checked értékét — kezdetben egy sincs kiválasztva (2, 3, 4), de az első választás után a felhasználó már nem tudja visszaállítani ezt az állapotot (groupBox2), mindig pontosan egy választás lesz érvényes. A kódban természetesen hamisra állíthatjuk a csoport minden elemének Checked tulajdonságát akár futásidőben is.

Megjegyzés:

Ha egy, már a felületen levő választógombot a csoportba húzunk, akkor az a csoport tagjává válik, azonban ha a csoportot húzzuk a gomb fölé, akkor nem, hiába tesszük a mögéküldés segítségével a választógombot a csoport fölött láthatóvá.

A gombok háttérszíne (BackColor), sőt háttérképe (BackgroundImage) is állítható. A betűszínt a ForeColor tulajdonsággal, a betű tulajdonságait pedig a Font csoport alatt állíthatjuk. A RadioButton és a CheckBox esetén a **Checked** tulajdonság adja meg, ki van-e választva az adott vezérlő, és a leggyakrabban a **CheckedChanged** eseményüket kezeljük. Ha a választógombok esetén közös metódussal szeretnénk a választás módosulását kezelni, ezt a legegyszerűbben úgy tehetjük meg, hogy az első gomb esetén a gomb nevétől független, esetleg a csoport nevére utaló kezelőmetódust adunk meg, a többi gombnál pedig a legördülő listából ezt a már létező metódust választjuk ki.

A RadioButton csoportok használatára az 5.4. tesztfeladatban látunk példát.

3.2.3. A TextBox, a ListBox, a CheckedListBox és a ComboBox

A **TextBox** vezérlő változó adatok kiírására, beolvasására szolgál. Text tulajdonsága szöveges adatot jelenít meg, így a kiírni kívánt nem sztring típusú adatot többnyire a **ToString**() metódussal előbb szöveggé kell alakítanunk. Beolvasáskor legegyszerűbben a **Convert** osztály különböző (**ToInt32**, **ToBoolean**, **ToDateTime**...) átalakító függvényei segítségével konvertálhatjuk a Text tulajdonság szövegét a kívánt típusra. Leggyakrabban kezelt eseménye a **TextChanged**.

Egy lista megjelenítésére szolgál a **ListBox** vezérlő, speciális esete a **CheckedListBox**, mely képes megjegyezni a kiválasztott elemeket, és egy pipával

jelölni őket. Kis helyigénye miatt közkedvelt a **ComboBox**, mely legördülő listaként mutatja meg az elemeket, és rendelkezik egy TextBox résszel, ami a bevitelt teszi lehetővé. Mindhárom esetén az **Items** tulajdonság segítségével adhatjuk meg a lista elemeit. Futásidőben az Items.**Add** segítségével vehetünk fel új elemet, míg az Items.**Remove** segítségével törölhetünk elemet a listából.

A **ComboBox DropDown Style** -ja háromféle lehet: **Simple** esetén mutatja a listaelemeket, és függőleges mérete beállítható. Simple esetén nincs mód új elem bevitelére. **DropDown** stílus esetén legördül, és ha elkezdtük a gépelést, az első karakterek alapján rááll legördítéskor a megadott elemre. Ha megvalósítjuk az eseménykezelést, mód van új elem bevitelére is a listába. Ha ComboBoxStyle.**DropdownList** típusú, akkor a begépelt első karakterrel kezdődő listaelemet automatikusan kiválasztja, s a karakter újbóli leütésére lép a következő adott karakterrel kezdődő elemre. Természetesen, ha legördítjük, itt is az aktuális elem a kiválasztott a listán. Új elem bevitelét ez a stílus sem támogatja.

A **ComboBox.MaxDropDownItems** tulajdonságával adhatjuk meg, maximálisan hány elem látható a ComboBox legördített listájában.

3.2.4. A ProgressBar és a TrackBar

Numerikus értékek folyamatos beállítására, kijelzésére használatos a **csúszka** (**TrackBar**), illetve a **folyamatkijelző** (**ProgressBar**). Mindkettőnél beállítható a maximum / minimum érték és az aktuális érték (Value). Csúszka esetén megadható, hogy a jelek milyen egységet jelöljenek (**TickFrequency**), és hogy a billentyűzet nyilai hány egységgel változtassák az aktuális értéket (**SmallChange**), folyamatkijelzőnél beállíthatjuk, mikor jelenjen meg az új jel (**Step**).

3.2.5. A TabControl és a Panel

A **Panel** segítségével elérhetjük, hogy a ráhúzott vezérlők együtt mozgathatók, megjeleníthetők legyenek. Beállíthatunk közös háttérszínt, háttérképet, mindent, ami az adott vezérlőelemek elkülönítésére szolgál. Segítségével megvalósíthatjuk, hogy bizonyos adatok esetén ilyen, más adatok esetén másmilyen vezérlők jelenjenek meg a form egy adott részén.

Több egymásra lapolt ablak kezelését teszi lehetővé a **TabControl**. A **TabPages** tulajdonságában felsorolhatjuk a különböző füleket, megadva nevüket. A le-fel nyilak segítségével beállíthatjuk sorrendjüket. A tervező ablakon (Design) a kívánt fülre kattintva egymás fölé lapolt (azonos méretű) paneleket kapunk, melyekre tetszőleges vezérlők húzhatók. A lapok közül a fülekre kattintva választhatunk.

3.2.6. A további elemek

Természetesen további vezérlőelemek is rendelkezésünkre állnak, melyekre összetettebb feladatok megvalósítása esetén lehet szükségünk. Nagyméretű ablakot gördítősávokkal kezelhetünk, létrehozhatunk eszközsort, státuszsorban tájékoztathatunk az alkalmazás állapotáról... Ezek megismerését a későbbiekre halasztjuk.

A ToolBox rendelkezik olyan elemekkel is, melyek nem vezérlők, hanem bizonyos funkciókat támogatnak az ablakhoz kapcsolódóan (Menu, ToolTip, Timer ...). Ezeket A komponensek fejezetben tárgyalja a könyv.

3.3. Az ablakok megjelenítése

3.3.1. A modális ablak

Ablakokat modálisan vagy nem modálisan lehet megjeleníteni. Ha egy ablak **modális**, akkor az alkalmazás többi ablaka nem kerülhet fókuszba addig, amíg az ablakunkat be nem zártuk. Modális ablakok tipikusan a hibaüzenet küldő ablakok, de modálisan szokás az adatokat bekérő ablakot is megvalósítani. Modális ablakot a Form osztály **ShowDialog** metódusával tehetünk ki a képernyőre.

Modális ablaknál gyakran csak egy OK gomb jelzi, hogy nincs választási lehetőségünk. Azonban ha fel akarjuk kínálni a felhasználó számára a választást, akkor az ablak bezárása után tudnunk kell, mi volt a választás. Erre szolgál a **DialogResult** tulajdonság. Értéke futásidőben a DialogResult felsorolástípus értékei közül választva állítható. Lehet: OK, Cancel, Yes, No, Retry, Ignore... Az ablak bezárása után az objektum DialogResult tulajdonságát lekérdezve tudjuk meg, mit választott a felhasználó.

A párbeszédablakokat ShowDialog metódus segítségével szokás megjeleníteni.

3.3.2. A nem modális ablak

Ha formunkat a **Show** metódussal tesszük ki a képernyőre, megjelenése nem lesz modális. Vagyis tőle függetlenül dolgozhatunk az alkalmazás többi ablakában is, ha meg van nyitva más ablak is. Amelyik ablak fókuszban van, oda történik az adatbevitel a billentyűzetről. A többi ablak nem záródik be, csak elveszíti a fókuszt.

3.3.3. A Form tulajdonságai

Ha nem az egyes vezérlőket választjuk ki, hanem az egész form ki van jelölve, akkor az ablakra vonatkozó tulajdonságok állíthatók. Ezt használtuk az első feladatban, amikor a háttérképet állítottuk be, vagy amikor a Text mezőt kitöltve megadtuk a form címsorának szövegét. Itt néhány gyakran használt tulajdonságról lesz szó.

Az ablak méretét a **Width**, **Height** (Size) tulajdonságokkal adhatjuk meg. A megjelenítés módosításához előbb a **StartPosition** tulajdonságot kell módosítanunk, mert bár a Location tulajdonság X és Y értékének megadásával a bal fölső sarok koordinátáit adtuk meg, azonban a StartPosition alapértelmezett WindowDefaultLocation beállítása részben felülbírálja e beállítást. Érvényesítéséhez Manual-ra kell állítanunk a StartPositiont. Ha a képernyő közepén szeretnénk látni ablakunkat, értéke legyen CenterScreen! Futásidőben a Left, Top, Right, Bottom tulajdonságokkal állíthatjuk a bal, fölső, jobb, alsó sarok értékét.

A **WindowState** tulajdonság Minimized, Maximized-ra állításával elérhetjük, hogy ablakunk minimalizálva (tehát csak a tálcán) jelenik meg vagy maximális méretben, vagyis a teljes képernyőt elfoglalva. A MinimizeBox, MaximizeBox tulajdonság megadja, hogy választható-e a minimalizáló, maximalizáló ikon a címsorban. A **ShowInTaskbar** tulajdonság hamisra állításával eltüntethetjük a tálcáról az ablakunkat. A **FormBorderStyle** tulajdonság beállításával engedélyezhetjük a méretezhetőséget. Ha igazra állítjuk a **TopMost** tulajdonságot, akkor ez az ablak lesz a legfelső a képernyőn akkor is, ha egy mögötte levő van fókuszban.

Párbeszédablakoknál szokásos az OK gombot Enter hatására is, míg a Cancel / Mégsem gombot Esc hatására is meghívni. Ezt úgy érhetjük el, hogy a form **AcceptButton** tulajdonságánál az OK, míg **CancelButton** tulajdonságánál a Cancel gombot választjuk ki. Értelemszerűen bármely más gomb is kiválasztható a gombok listájából.

Az ablak áttetszőségét az **Opacity** tulajdonságban a láthatóság %-ában adhatjuk meg. Teljesen látható ablak 100%. A láthatatlan 0%. Megadhatjuk azt is, hogy ablakunk melyik színű eleme lesz láthatatlan a **TransparencyKey** tulajdonság színének választásakor. Ha ablakunk egy részét időnként láthatatlanná kívánjuk tenni, csak tegyünk az adott színű panelt a megfelelő rész fölé! Az átlátszó rész alatt értelemszerűen látszanak a mögötte elhelyezkedő ablakok, és az egérrel kiválasztva aktualizálhatók is.

Az ablak **Region** tulajdonsága adja meg az ablak alakját. Pontosabban tartalmazza azon pontok összességét, amit az operációs rendszer kirajzol. A régió tetszőleges számú téglalap, ellipszis és sokszög egyesítésével hozható létre. Így tehát tetszőleges alakú ablakokkal dolgozhatunk. Ha a Region tulajdonság nincs megadva (undefined value), akkor a hagyományos ablak jelenik meg.

 A Region használatára a gyakorlaton látunk példát.

3.3.4. A .NET osztálykönyvtár párbeszédablakai

A Windows operációs rendszeren futó alkalmazások egyszerűbb használatának az is a magyarázata, hogy az alkalmazások által gyakran használt funkciókhoz alapértelmezett párbeszédablakokat biztosít. Például bármelyik alkalmazásunkból

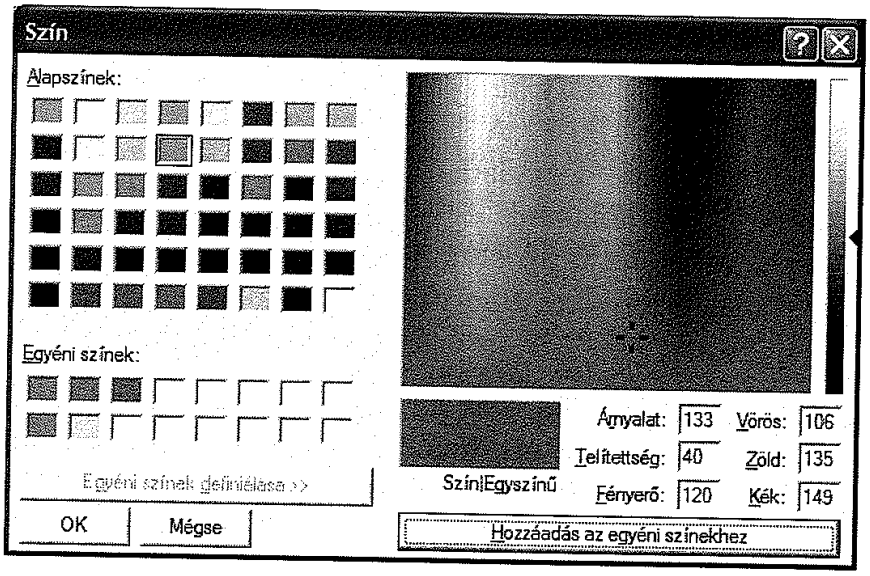
kívánunk kinyomtatni valamit, ugyanazt az ablakot látjuk megjelenni a képernyőn. És bár az angol nyelvű Visual Studioban fejlesztjük alkalmazásainkat, ha operációs rendszerünk magyar nyelvű, akkor magyar nyelvű hibaüzenet ablakok jelennek meg futás közben.

Az operációs rendszer által szolgáltatott ablakokhoz a .NET osztálykönyvtár ablakosztályokat biztosít.

A System.Windows.Forms névtér tartalmazza a szokásos párbeszédablakok leírását. Ilyen párbeszédablakok a fájl megnyitásakor, mentésekor használt OpenFileDialog és SaveFileDialog, a betűtípus kiválasztására használt FontDialog vagy a nyomtatáskor használt PrintDialog.

3.3.4.1. ColorDialog

A színek kényelmes kiválasztására szolgáló ablak. Megnyitása előtt a tulajdonságai beállíthatók. A **Color** tulajdonsága megadja a megjelenésekor beállított színt. Alapértelmezése: Black. Az ablakot a **ShowDialog** metódussal tehetjük ki a képernyőre, melynek visszatérési értéke a **DialogResult**, ami megadja, mely gombot választottuk a bezáráshoz. Magának a **ColorDialog** osztálynak a Formtól eltérően nincs DialogResult tulajdonsága. Az ablak bezárása után a **Color** tulajdonságával kérdezhető le az ablakban kiválasztott szín.



3.3. ábra A ColorDialog ablak

```
ColorDialog dlg= new ColorDialog();       // Új ablak.
if (dlg.ShowDialog()==DialogResult.OK)    // Kiteszi modálisan.
        BackColor = dlg.Color;            // Kiolvassa a színt.
```

3.4. Teszt

1. Hogyan állítható be, hogy ha párbeszédablakot egy gomb választására zártunk be az eredmény OK legyen?

- A gomb felirata (Text) legyen OK!
- A gomb neve legyen OKButton!
- A gombon való kattintás eseményben állítsuk a DialogResult tulajdonság értékét OK-ra!
- A gomb legyen az AcceptButton!

2. A modális ablak bezárása nélkül is dolgozhatunk az alkalmazás más ablakaiban.
3. A TabControl vezérlő segítségével megadhatjuk az ablak tabulátorsorrendjét.
4. A RadioButton vezérlők egymástól független választást kínálnak a felhasználónak.
5. A ProgressBar segítségével állíthatunk be értékeket és a TrackBar csak kijelzésre alkalmas.
6. A ComboBox Simple beállítása a szokásos, a nyíl választására lenyíló listát jelenti.
7. Ha a TextBox Number tulajdonságát igazra állítjuk, akkor csak számjegyeket fogad el bevitelkor.
8. A RadioButton Group tulajdonságának beállításával hozhatunk létre csoportokat.
9. A form akkor jelenik meg párbeszédablakként, ha DialogResult tulajdonságát OK-ra állítottuk.
10. Ha az ablak MinimizeBox tulajdonsága hamis, akkor nincs minimalizáló ikon a címsorban.
11. A Form osztály DialogResult tulajdonságából tudhatjuk meg, hogy melyik gombot választottuk a Form bezárásakor.
12. A ColorDialog osztály DialogResult tulajdonságából tudhatjuk meg, hogy melyik gombot választottuk az ablak bezárásakor.

Javítókulcs:

DialogResult; H; H; H; H; H; H; H; H; H, I, H.

4. Elemi programozási tételek

„Amikor a feladatot úgy akarjuk megoldani, hogy egy általános módszert kell megoldanunk, amellyel minden konkrét esetben meg tudjuk válaszolni az adott típusú kérdést, *algoritmikus problémáról* beszélhetünk."[1]

Amint azt Grady Booch kifejti: A szoftverfejlesztés hagyományosan kialakult módszere az algoritmikus nézőpont. Eszerint a szoftver fő építőeleme a függvény. A fejlesztő a vezérlésre és az összetett algoritmusok kisebb egységekre bontására koncentrál. Azonban ahogy az igények változnak (márpedig változni fognak), és a rendszer növekszik (márpedig növekedni fog), az algoritmikus nézőpontból felépített rendszerek fenntartása nagyon nehézzé válik.

A jelenkori szoftverfejlesztés objektumorientált szemléletű. Fő építőeleme az objektum vagy osztály. Az objektumok állapotai és viselkedése áll a szoftverfejlesztés középpontjában. A módszer értéke bizonyított nagyméretű komplex programok esetén.[2]

Objektumorientált szemléletben írtuk az előző fejezetekben is programjainkat, de az egyes tagfüggvények megvalósítása (implementálása) során nagy hasznát vesszük az algoritmikus gondolkodásnak. A strukturált filozófia szerint algoritmusaink értékadó utasítás, szekvencia, elágazás és ciklus felhasználásával építhetők fel. A feladatok

---

[1] Demetrovics, Denev, Pavlov: A számítástudomány matematikai alapjai, 244. old.

[2] Booch, Rumbaugh, Jacobson: The Unified Modeling Language User Guide. 10-11. old.

megoldása során kikristályosodott általános algoritmusokat, melyek helyességét matematikai úton is bizonyították, programozási tételeknek nevezzük.

A könyv nem ismerteti az összes programozási tételt, csupán az algoritmikus gondolkodás fejlesztése érdekében bemutatja, milyen algoritmusok használatosak a programozók körében a gyakran előforduló problémák megoldására.

A programozási tételek jelentős részét az osztálykönyvtár tagfüggvényei szolgáltatják, így azokat leggyakrabban mint egyszerű függvényhívásokat használjuk. Sokszor azonban magunknak kell megoldanunk a problémát, melyhez nagy segítséget nyújt a programozási tételek ismerete.

Mielőtt a konkrét tételekre rátérnénk, ismerjünk meg egy egyszerű, de gyakran használt algoritmust, mely nem tesz mást, mint két változó értékét felcseréli.

A csere

A csere nem egy programozási tétel, hanem gyakran használt technika.

Adott két azonos típusú változó 'a' és 'b', szeretnénk felcserélni az értéküket! A feladat megoldásához szükségünk lesz egy segédváltozóra! Ha ugyanis pl. azt mondanánk a=b; elveszítenénk 'a' értékét! A segédváltozó legyen 'temp', és típusa egyezzen meg a két változó típusával!

A csere algoritmusa három értékadó utasítás szekvenciája:

```
temp=a;
a=b;
b=temp;
```

A következő tételek egy adott – értékekkel rendelkező – sorozatra vonatkoznak, és egy érték eredményt adnak. A sorozat elemeit egy t tömbben tároljuk. A tömb elemszáma: n, az indexet, amivel végigmegyünk a tömb elemein, i-vel jelöljük.

4.1. A megszámlálás és az összegzés

4.1.1. A megszámlálás

Hány 'db' adott feltételt teljesítő eleme van az adott 'v' 'n' elemű sorozatnak? A feltétel legyen: >5!

```
db=0;
for (int i=0; i<n; i++)
 if (v[i]>5) db++;
```

Feltétel nélküli megszámlálást valósít meg a **ListBox** és a **ComboBox Items.Count** tulajdonsága és a **String** osztály **Length** tulajdonsága.

4.1.2. Az összegzés

Mennyi az adott 'v' 'n' elemű sorozat elemeinek összege 'sum'?

```
sum=0;
for (int i=0; i<n; i++)
 sum=sum+v[i];
```

4.1.3. A feltételes összegzés

Ha az adott feltételt (>5) teljesítő elemek összegét szeretnénk kiszámolni:

```
sum=0;
for (int i=0; i<n; i++)
 if (v[i]>5) sum=sum+v[i];
```

4.2. A keresés

4.2.1. A lineáris keresés és az eldöntés

Van-e a 'v' 'n' elemű sorozatnak adott feltételt (>5) teljesítő eleme? Az eldöntés logikai (bool) eredményt hoz. Az eredményt célszerű a kód olvashatósága érdekében úgy nevezni, hogy az igaz állítást takarja (van).

```
van=false;
i=0;
while (!van && i<n)
{
 van = v[i]>5;
 i++;
}
```

Vagy másként megvalósítva:

```
i=0;
while (!(v[i]>5) && i<n)
{
 i++;
}
van=i<n;
```

A ciklusból vagy akkor lép ki a program, ha megtalálta az első, a feltételt teljesítő elemet, vagy ha elért a sorozat végére. A kilépéskor van értéke igaz, ha megtalálta a megfelelő elemet, és hamis, ha nem.

Lineáris keresést, eldöntést valósít meg a **ListBox**, és a **ComboBox.Items,Contains** metódusa, mely igaz értékkel tér vissza, ha a paraméterében megadott elem szerepel a lista elemei közt, egyébként hamissal.

4.2.2. A lineáris keresés és a kiválasztás

Ha alaposabban végiggondoljuk, az előző tétel valójában két további információt is szolgáltat. Az i a kilépés után épp a feltételt teljesítő index után következő szám. Így egy i-1; utasítással megkaphatjuk az első a feltételt teljesítő elem indexét. Az index ismeretében pedig tudjuk az elem értékét is v[i].

A feladatot tovább bővíthetjük, ha megadjuk az indexet 'm', ami után a következő elemekre végezzük el a vizsgálatot.

```
van=false;
i=m;
while (!van && i<n)
{
 van = v[i]>5;
 i++;
}
if (van)
{
 i--;
 MessageBox.Show("Az "+i+". helyen van a(z) "+m+". utáni
       5-nél nagyobb elem. Az elem értéke: "v[i]".");
}
else
 MessageBox.Show("Nincs keresett elem.");
```

Vagy másként megvalósítva:

```
i=m;
while (!(v[i]>5) && i<n)
{
 i++;
}
if (i<n)
{
 MessageBox.Show("Az "+i+". helyen van a(z) "+m+". utáni
       5-nél nagyobb elem. Az elem értéke: "v[i]".");
}
else
 MessageBox.Show("Nincs keresett elem.");
```

A feladatokban mindig az adott feltételt teljesítő első előfordulást kerestük. Természetesen az utolsó előfordulás könnyen megtalálható, ha hátulról előre keresünk.

Lineáris keresést, kiválasztást valósít meg a **RichTextBox** osztály **Find** metódusa, mely egy karaktertömböt vagy sztringet keres a vezérlő szövegében. A visszatérési értéke a keresett string első karakterének indexe. A **RichTextBoxFinds** paraméterét megadva meghatározhatjuk, hogy a kis- és nagybetűk megkülönböztetésével, vagy sem, teljes szóra keressen-e és kiemelje-e a szövegből (a szöveg többi részétől eltérő módon írja-e) a megtalált szót, az utolsó előfordulást keresse-e.

Kiválasztást valósít meg a **ListBox** és **ComboBox Items** tulajdonságának **IndexOf** metódusa, mely megadja a keresett elem indexét a listában. Ha az elem nem szerepel a listában -1-gyel tér vissza.

4.2.3. A bináris vagy logaritmikus keresés

A fenti algoritmussal természetesen egy adott értékű elemet is megtalálhatunk. Ha tudjuk, hogy sorozatunk rendezett, akkor hatékonyabbá tehető az eljárás. Először is ekkor elegendő az első a keresett elemnél nagyobb elemig menni, hisz rendezett sorozatban az összes következő már mind nagyobb lesz.

Gondoljunk azonban a telefonkönyvre (és feledkezzünk meg arról, hogy a név kezdőbetűjéből mi már tudjuk, hogy a könyv elején vagy végén keressük)! Keresünk egy Ford Prefect nevű személyt. Felcsapjuk a telefonkönyvet középen, majd megállapítjuk, hogy az ott található nevekhez képest előbbre vagy hátrébb van a keresett név. Ha előbbre, akkor az első részt felezzük meg, ha hátrébb, a második részt felezzük. Az itt található névnél megint előbbre vagy hátrébb van a keresett az adott részben. Az algoritmust addig folytatjuk, míg meg nem találtuk a nevet vagy azt a két nevet, amik között kellene lennie. Ez a módszer sokkal hatékonyabb, mintha elkezdenénk elölről olvasni a telefonkönyvet, amíg meg nem találjuk a keresett nevet. Persze, ha Arthur nevű fickót keresünk lehet, hogy gyorsabban megtaláljuk az előző eljárással, de Zaphodot már nem. Tehát nem minden esetben, de sok keresés esetén bizonyosan gyorsabb.

Nézzük az algoritmust!

A bináris keresés előfeltétele, hogy az adott sorozatunk rendezett legyen. Keressük az 'a' értékű elemet!

```
also=0;
felso=n;
kozepso=(also+felso)/2;
while(v[kozepso]!=a && also<felso)
{
 if (a<v[kozepso]) felso=kozepso-1
 else also=kozepso+1;
 kozepso=(also+felso)/2;
}
```

```
if(v[kozepso]==a)
 MessageBox.Show("Az "+a+" értékű elem a "+kozepso+". helyen
van.");
else
 MessageBox.Show("Nincs "+a+" értékű elem a sorozatban.");
```

4.2.4. A minimum keresés

Keressük az adott – nem rendezett – sorozat legkisebb elemét!

A keresést úgy valósítjuk meg, hogy egy segédváltozóban (min) tároljuk az aktuális legkisebb elemet, és ezzel hasonlítjuk össze az összes többit, miközben végigmegyünk a sorozat elemein.

```
if (n>0) min=v[0];
else MessageBox.Show("A sorozat üres, így nincs legkisebb
                                                    eleme.");
for (int i=1; i<n; i++)
 if (v[i]<min) min=v[i];
MessageBox.Show("A sorozat legkisebb eleme: "+min+".");
```

Hasonló módon a legnagyobb elem is megtalálható.

4.3. A rendezés

Nagyon sok rendezési algoritmus ismert, mi most az eddig tanultak alapján a legegyszerűbbel ismerkedünk csak meg.

4.3.1. Rendezés minimum kiválasztással

A rendezés elve: megkeressük a legkisebb elemet, kiemeljük, és a még nem rendezett rész első elemével kicseréljük. A rendezetlen részben ismételjük a feladatot.

A megoldásban tehát két egymásba ágyazott ciklus van. A külső ciklus blokkjának minden végrehajtása után eggyel nő a sorozat elején a rendezett elemek száma és rövidül a rendezetlen sorozatrész. A belső ciklus egy minimumkeresés és egy csere. A csere miatt nem a minimum értékét, hanem a minimális elem indexét tároljuk. A belső ciklust csak akkor hajtjuk végre, ha van még legalább két rendezetlen elem. Ha egy 'rendezetlen' elemünk van, az már a legnagyobb, tehát befejezhetjük a rendezést.

```
for (int j=0; j<n-1; j++)
{
 minInd = j;
 for (int i=j+1;i<n; i++)
        if (v[i]<v[minInd]) minInd=i;
 if (minInd>j)
 {// Csere
        min=v[minInd];
        v[minInd]=v[j];
        v[j]=min;
 }
}
```

A **ListBox** és a **ComboBox.Sorted** tulajdonságát ha lekérdezzük, megtudjuk, hogy a vezérlő rendezett-e, de ha igazra állítjuk, a tulajdonság set metódusa abc sorrendbe rendezi a vezérlő listájának elemeit. Alapértelmezésben hamis a tulajdonság értéke.

4.4. Teszt

1. A bináris keresést csak rendezett sorozatra alkalmazhatjuk.
2. A lineáris keresést csak nem rendezett sorozatra alkalmazhatjuk.
3. A következő kódrészlet két változó értékét cseréli ki: a=b; b=a;
4. Mit csinál a következő kódrészlet, ha feltételezzük, hogy a v sorozat legalább egy elemű?

```
a=0;
for (int i=1; i<n; i++)
 if (v[i]>v[a]) a=i;
```

5. Keresse meg a v sorozatban a második 5-tel megegyező elemet!
6. Mi a hiba a következő kódrészletben? Az algoritmus az első 3 értékű elemet keresi.

```
i=0;
while (v[i]!=3)
 i++;
```

7. Keresse meg a 'v' sorozatban a 10-nél nagyobb legkisebb elemet!

Megoldások:

I, H, H, A maximális elem indexét keresi,

```
db=0;
i=0;
while (db<2 && i<n)
{
 if (v[i]=5) db++;
 i++;
}
if (db==2) MessageBox.Show(„A második 5 értékű elem indexe: „
                                                    +i--+".");
else MessageBox.Show(„Nincs két 5 értékű eleme a sorozatnak.");
```

Üres tömb esetén is lekérdezi a 0. elemet. Ha nincs 3 értékű elem, akkor az utolsó tömbindexen túl fogja címezni a tömböt. Vagyis nem áll meg a tömb végén.

```
i=0;
while (v[i]<=10 && i<n) i++; // Az első 10-nél nagyobb szám.
if (i<n) min=v[i];
while (i<n)
{
 if (v[i]>10 && v[i]<min) min=v[i];
 i++;
}
```

5. Az öröklés és a .NET osztálykönyvtár

Az objektumorientált programozás elterjedésének egyik oka, hogy nagymértékben támogatja a kód-újrafelhasználást. Az osztályok egy egységben írják le az adatokat és a rajtuk végzett műveleteket. Az objektumorientált nyelvekben gyakran önálló forrásfájlban (C#, Java) vagy forrásfájlokban (C++) tárolják az egyes osztályok leírását. Egy alkalmazáshoz megírt osztály kódja így könnyen felhasználható egy másik alkalmazás fejlesztése során.

Újrafelhasználást jelent, ha egy már korábban megírt osztályból objektumot hozunk létre. Ezt tettük amikor a formunkra vezérlőket húztunk. A vezérlőkhöz tartozó objektumok egy – a Microsoft által – már megírt osztály példányai, melyeket a Name tulajdonság megadásával neveztünk el. Ezért nem kell megírnunk azt, hogyan állíthatjuk be a gomb szövegét vagy a megjelenítését biztosító függvényeket.

A kód-újrafelhasználás további lehetőségét jelenti az öröklés, mely az objektumorientált programozás egyik alapfogalma. Az öröklés során a kész osztály kódját az utódosztály fejlesztése során használjuk. Nagyméretű, grafikus felülettel rendelkező, eseményvezérelt alkalmazások gyors és hatékony fejlesztéséhez elengedhetetlen a kód-újrafelhasználás.

5.1. Az öröklés

5.1.1. Specializáció és absztrakció

- **Specializáció**

Gyakran előforduló helyzet a programozás során, amikor a feladat megoldásához egy már meglevő osztályt szeretnénk használni, de annak kódja nem tartalmaz valamilyen adatot, amit nekünk tárolni kellene. Vagy az adott osztály objektumán nem lehet elvégezni az általunk várt műveletet, esetleg el lehet végezni, de nem úgy, ahogy mi szeretnénk. Tehát szükségünk lenne egy osztályra, ami olyan, mint a már meglevő, de vagy több adattagja van, vagy több tagfüggvénye, vagy egyes függvényeinek kódja eltérő. Az eltérések akár mindegyike egyszerre is megjelenhet. Azt mondjuk specializációra van szükségünk, s ennek eszköze az öröklés.

Az öröklés technikáját használtuk már eddig írt kódjaink során is, hisz a speciális, csak az adott alkalmazásra jellemző form az System.Windows.Forms.Form osztály utódosztálya volt.

- **Absztrakció**

Az öröklés másik indoka az absztrakció lehet. Ha vannak hasonló feladatokat megvalósító osztályaink, melyek közt csak speciális feladatokban van eltérés, akkor ezekben a közös függvényeket többször kódoljuk le. Ilyenkor természetesen másolni szoktuk a kódot, de a későbbi módosítások során bizony mindenütt meg kell változtatni a kódrészletet, majd az összes módosítást újra tesztelni. Ez tipikusan tévedésekhez, hibás kódhoz vezet. Valamelyik függvényt biztosan elfelejtjük módosítani, és aztán tesztelni is. Programozás technológiai alapelv, hogy kerüljük az ismétlődő kódrészleteket! Osztályok esetén ezt úgy valósíthatjuk meg, hogy az ismétlődő funkciókat egy közös ősosztályban valósítjuk meg, s ebből származtatjuk az egyedi tulajdonságokkal rendelkező speciális osztályokat.

5.1.2. Az öröklés fogalma

Az utódosztály objektuma mindent tud, amit az ősosztály objektuma tudott, hacsak a későbbiekben nem szűkítjük az objektum lehetőségeit. Vagyis az ősosztály minden adattagja adattagja az utódosztálynak is, bár ezek az adattagok gyakran az egységbezárás elvének betartása miatt csak metódusokon keresztül érhetők el (pl. tulajdonságok). Az ősosztály private metódusain kívül bármely metódusa meghívható az utódosztályból. Tehát az utódosztály nem tud mindent, amit az ősosztály tudott, hisz annak private tagjait nem érheti el. Az objektumok viszont egyébként sem érhetik el a private tagokat, tehát az utódobjektum mindent tud, amit az ős tudott.

- **Az öröklés célja:**
  - Új adattagok tárolása az utódosztályban.
  - Új tagfüggvények létrehozása az utódosztályban.
  - Meglevő tagfüggvények módosítása az utódosztályban.

Ha egy névtérben hozzuk létre az osztályt, hozzáférési szintje public vagy internal lehet. Mivel eddig egy assemblyben kódoltuk alkalmazásainkat, a két hozzáférés közti különbséget még nem érzékeljük, de a komponensekkel foglalkozó következő fejezetekben látni fogjuk az eltérést. Alapértelmezésben, tehát ha nem írjuk ki, akkor internal. Vagyis az osztály csak az adott assembly (dll, exe) számára érhető el. Ha azt akarjuk, hogy más programegységek is hozzáférhessenek, public láthatóságot kell megadnunk!

- **Szintaxisa:**

```
public class UtódosztályNév : ŐsosztályNév
{
 Az osztály tagjainak deklarációja.
};
```

Az osztály deklarációjakor többnyire megadjuk az osztály adattagjait és tagfüggvényeit is a kapcsos zárójelek között, de ez nem kötelező, megtehetjük, hogy a kód egy későbbi részében definiáljuk az osztályt.

- **A sealed módosító:**

A C# nyelv lehetővé teszi sealed (lezárt) módosítóval ellátott osztály létrehozását, mely azt jelenti, ebből az osztályból nem lehet származtatni. Például a String osztály sealed, így tehát nem tudunk a String osztályból utódosztályt létrehozni. A sealed módosítót többnyire a program hatékonysága érdekében használjuk. Ha megnézzük az CLR osztálykönyvtár számunkra felkínált osztályait, elég sok sealed módosítóval ellátott osztályt találunk.

View menü

Object Bowser

mscorlib

System névtér

String

vigyük az egeret a String szöveg előtti osztályt jelző ikonra!

A többi változó osztály, vagy struktúra is sealed.

system.drawing

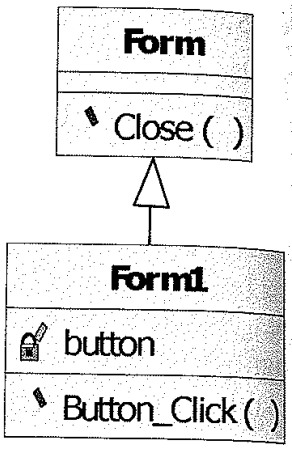
System.Drawing névtér

Bitmap, Font, Icon, Pen ... osztályai is lezártak.

5.1.3. Az eddig használt öröklés

Bár nem beszéltünk róla, de a Visual Studo fejlesztőeszköz, amikor új Windows Application projektet generáltunk, mindig készített nekünk egy Form1 osztályt – amit aztán mindig átneveztünk –, és ami a CLR osztálykönyvtárban megírt Form osztály utóda.

```
public class Form1 : System.Windows.Forms.Form
```



Az ősosztály a System.Windows.Forms névtér Form osztálya. Az Object Browser ablakban a system.windows.forms komponens (dll, assembly) System.Windows.Forms névtere alatti felsorolásban szerepel. Ha kiválasztjuk, a Members of 'Form' ablakban láthatjuk az osztály számunkra elérhető tagjait. A private tagokat nem találjuk, mert azok az osztály saját belső tagjai. Mivel mi nem módosíthatjuk az osztály kódját, csak származtathatunk belőle, vagy példányokat hozhatunk létre, így nincs szükségünk a private tagok ismeretére, hisz azokat úgysem érhetjük el.

A Form osztály maga nem alkalmas egyetlen alkalmazásban sem felhasználói felületnek, hisz nincs rajta egyetlen vezérlő sem. Ezért származtatunk belőle minden alkalommal egy utódosztályt, amiben mi az adott feladatnak megfelelően biztosítjuk a kívánt felületet. Vagyis specializáljuk a formunkat. A Form osztály mint ősosztály biztosít számukra egy már megírt kódtömeget, melyet mi felhasználhatunk az utódosztály működése közben. Például nem írtuk meg a megjelenítő függvényt, mégis megjelenik az alkalmazás elindítása után a Form a képernyőn. Vagy a minimalizáló vagy maximalizáló ikon hatására minimalizálódik vagy teljes képernyős lesz, a bezáró ikon választásakor pedig bezáródik az ablak.

Amikor készítették ezt az osztályt, összegyűjtötték a tapasztalatok alapján a formokon szokásos műveleteket, és az ezek megvalósításához szükséges adatokat. Az osztálykönyvtár fejlesztői elkészítették a Form osztályt, hogy mi élhessünk fejlesztésünk során a kód-újrafelhasználás eszközével, és gyorsan, egyszerűen készíthessük el saját formjainkat.

Megjegyzés:

A Form osztály tagjai közt, de az ősosztályainak tagjai közt sem találunk adattagokat. Ez nem azt jelenti, hogy az osztálynak nincsenek adattagjai, hanem azt, hogy a fejlesztők korrektül alkalmazták az encapsulation (egységbezárás) elvét, és elrejtették előlünk az adattagokat private tagváltozókba. Hozzáférésüket pedig többnyire public tulajdonságokkal teszik lehetővé. Természetesen a BackColor vagy a Size tulajdonság mögött tárolt private adatok vannak.

A Form osztály kapcsán mindjárt példát láthatunk több egymásból származtatott osztályra is. Például a saját Form1 osztályunk Location tulajdonságát a Form ősosztályai közti Control osztályban definiálták, onnét örökölte a ScrollableControl, a ContainerControl, a Form majd a Form1.

Az öröklött tagokat az osztály leírásakor nem szokás feltüntetni, az ezekre vonatkozó információt az öröklés tényével adjuk meg. Ezért a saját form osztályunk megjelenítésekor a ClassViewban sokkal kevesebb tagot találunk, mint amikor a this. segítségével súgót hívunk megtudni, milyen tagjai érhetőek el az osztálynak. Ha egy osztály rendelkezik egy tulajdonsággal vagy metódussal, nem lényeges információ számunkra melyik ősétől örökölte azt. Fő, hogy van neki ilyen vagy nincs.

5.1.4. A protected hozzáférés

Ha egy osztály egy tagja (adattag, metódus) protected láthatóságú, akkor az csak az osztály belsejéből vagy az utódosztály belsejéből érhető el. Tehát a protected tag az objektumok számára a privatehez hasonlóan láthatatlan, de az utódosztályok számára a publichoz hasonlóan látható. Vagyis a private és public láthatóság közötti hozzáférést biztosít.

A protected UML jele a #, a vizuális felületen kulcs képével jelzi, hogy a tag védett. Eddig nem használtuk ezt a láthatóságot, de a Visual Studio által generált kódban a form osztályunk Dispose metódusa protected láthatóságú, mert az ős Form osztályban is ez a hozzáférés szintje.

Nézzük meg az Object Browserben a Form osztályt! A Members of 'Form' ablakban láthajuk, hogy jónéhány protected láthatóságú tagja van.

Ha a való világban keressük e hozzáférési szintek megfelelőit, a család jó példának bizonyulhat. Egy személy nyilvános (public) adatának tekinthető a neve, a telefonkönyvben szereplő telefonszáma. Ezt bárki megtudhatja. Privát adata a személyes levelei, melyet a legközvetlenebb hozzátartozónak sem illik felbontani, s védett (protected) a pénztárcája, amibe az utódok (a gyerekek) bejáratosak, de nem lenne szerencsés, ha a szomszédok vagy gyerekeik is onnét vennék a zsebpénzüket.

A protected láthatóságot működés közben is kipróbálhatjuk, ha megnyitjuk egy eddigi alkalmazásunk form osztályának bármelyik függvényét, vagy generálunk egy új WindowsApplication alkalmazást, ráhúzunk a formra egy gombot, és duplán kattintva a ButtonClick eseménykezelő metódusába ugrunk.

- A Form utódosztályából látjuk a Form osztály protected tagjait.

  Most mi egy a Form osztályból származó formban vagyunk. Ez utódosztály, és mi most a kódját írjuk, tehát látnunk kell a protected láthatóságú tagokat. Ha azt írjuk, hogy this. a súgó fel is kínálja nekünk a meghívható tagokat, közöttük kulccsal jelezve a protectedeket. Válasszunk ki egy protected metódust az Object Bowserben pl. CenterToScreen() – mert neki nincs paramétere –, és látni fogjuk, hogy végrehajtja-e! A this.CenterToScreen() kiválasztható, és futás közben meg is hívjuk, mert az ablakunk a képernyő közepére kerül.

- A Button objektumból nem érhetjük el a Button osztály protected tagjait.

  Most vizsgáljuk meg a gombunkat! A form számára ez egy adattag. A Button_Click metódus nem a Button osztály kódja, hanem a mi utód form osztályunké. A button objektum tehát a Button osztályon kívül van. Válasszunk most az Object Browserben egy protected metódust a Button osztályból! Pl. OnClick, vagy OnMouseUp leírva a button objektum nevét, a pont operátor után a súgó nem kínálja fel ezeket a protected metódusokat, sőt láthatóan egyetlen protected metódust sem. Ha mi begépeljük a metódus nevét, fordításkor a „Cannot access protected member…" hibaüzenetet kapjuk. Természetesen, hisz a protected tagok csak az osztály és utódosztályai számára érhetőek el. Az objektum pedig az osztályon kívül van.

Vagyis az osztály, utódosztály belsejéből (a this segítségével) elérhetjük, kívülről nem a protected tagokat.

5.1.5. Az ősosztály tagjainak elrejtése

Az utódosztályban lehetőségünk van új tagok létrehozására és már létező tagok elrejtésére. Ha az a cél, hogy egy már létező függvény másként viselkedjen, akkor azt a függvényt el kell rejtenünk. Azonos szignatúrájú függvény írásával az ősosztály azonos szignatúrájú metódusát elrejtettük, azonban a fordítás után a fordítótól figyelmeztető (kék) üzenetet kapunk, ha nem használtuk a **new** kulcsszót. A figyelmeztető üzenet a fordítás után Build esetén az Output ablakban, Run esetén pedig a Task List ablakban érkezik.

Ugyanez történik azonos nevű adattag elrejtése esetén is.

Ez az az eset, amikor az utódobjektum nem tudja ugyanazt mint az ős, hisz ősobjektumra hívható a Close(), míg utódra nem. De ha szűkítettük a láthatóságot, ez is volt a célunk.

Az öröklés alkalmazását mutatja az 5. gyakorlat szavazatszámlálás feladata. Formok további öröklését láthatjuk a feladatok közt a szereplőválogatás feladat megoldásában, és a Tili-toli is örökléssel valósul meg. A 7. gyakorlat 2. feladatában a Panel osztályból származtatunk írható Panelt.

5.2. A .NET osztálykönyvtár

A .NET osztálykönyvtár osztályai névterekbe csoportosítva érhetők el. A logikai csoportosítást jelentő névtereket fizikailag különböző dll-ekben (dinamic link library) tárolják. Ez biztosítja, hogy a futó alkalmazások ne legyenek óriás méretűek, hisz mindig csak a szükséges részeit tartalmazzák az osztálykönyvtárnak. A Visual Studio fejlesztőeszköz által generált Windows alkalmazások hivatkoznak a legfontosabb dll-ekre (References), és használják a szükséges névtereket (using). Ha további névterekben található osztályokra van szükségünk, akkor azok hivatkozásait az alkalmazásunkba be kell írnunk. Erre a következő fejezetekben bőségesen látunk példákat.

A súgóban részletes információt találunk az osztálykönyvtárral kapcsolatban a

Help

Contents

Visual Studio.NET

.NET Framework

Reference

Class Library

Ha kinyitjuk a mappát, néhány mondatos leírást találunk a főbb névterekről, de a + ikont kinyitva azonnal továbbléphetünk a kívánt névtérre. A névterek két csoportra oszthatók. A **Microsoft** kezdetűek pl. a Microsoft.CSharp – a fordítást és kódgenerálást segítik az adott nyelven. A Vsa a script generálást, míg a Win32 az operációs rendszer által generált események és a Registry kezelését támogatja.

A névterek másik nagy csoportja a **System** kezdetűek. A névterek egymásba ágyazhatóak. A System névtérbe sok további névtér van beágyazva. Ha a névtér szerkezeti felépítésére vagyunk kíváncsiak, akkor a System mappa megnyitása után nézzük meg a Namespace hierarchy linket! Ha viszont a System névtérben közvetlenül megvalósított osztályokra vagyunk kíváncsiak, akkor a névtér neve

előtti + ikonra kattintva láthatjuk az osztályok listáját. Válasszuk ki az Object osztályt!

5.2.1. Az Object osztály

Ez az osztály az őse minden osztálynak, amit a .NET osztálykönyvtár definiál. Úgynevezett alaposztály. Minden osztály őse. A típushierarchia gyökere. Mivel minden osztály az utóda, mindegyik örökli a tagfüggvényeit.

Equals metódusa lehetővé teszi két objektum összehasonlítását. Az alapértelmezett megvalósítás referenciák esetén az azonos referencia. Ha azonos objektumra hivatkoznak – nyilván egyenlők. Értéktípusok esetén bitenkénti egyenlőség. Sok utódosztályban felüldefiniálták az Equals metódust pl. a String osztályban is, ahol a két sztring karaktereinek egyenlőségét vizsgálja.

ToString metódusa minden osztály példányainak szöveges leírását adja.

Finalize metódusa hívódik meg minden objektumra, mielőtt az automatikusan megszűnik.

5.2.2. A String osztály

Unikód karakteres szöveget tárol. A **unikód** azt jelenti, nemcsak 256 angol karaktert képes tárolni, hanem a különböző nyelvek ékezetes és egyéb különleges karaktereit is egyértelműen, és mindenki számára egységes megállapodás szerint kezeli a különböző matematikai és egyéb használatos jelekkel együtt. Még a kínai és japán írásjelek is a unikód karakterek közé kerültek.

Mint már említettük, a String sealed osztály, tehát nem származtatható belőle új osztály.

A **String** osztály kisbetűvel – string – is hivatkozható. A sztring értéke a benne tárolt karakterek sorozata, mely érték nem módosítható, csak létrehozható. string str = "alma"; Azok a függvények, melyek látszólag az értékét módosítják, valójában egy új sztringgel térnek vissza.

A string speciális szerepet játszik a változók között, mert a képernyőn vagy más periférián történő értékek megjelenítésekor többnyire sztring típusú változókat várunk. Az Object osztály **ToString** metódusa biztosítja, hogy bármely objektum értékét sztringgé tudjuk alakítani. A felhasználói felület kezelését támogató vezérlők pedig rendelkeznek Text tulajdonsággal, melynek értékéül sztringeket adhatunk meg.

Gyakran fordul elő, hogy a sztring számjegyeket tartalmaz. A sztring típuson nem végezhetők matematikai műveletek, arra a szám típusok alkalmasak. A sztringek más értéktípusokká átalakíthatók. Erre a Convert osztály statikus tagfüggvényeit

5.2.4. A fájlba írás – olvasás

A különböző típusú adatok fájlba írásának és fájlból olvasásának legegyszerűbb módja, ha adatfolyamokat használunk. Ez azt jelenti, a különböző adatok egymás után folyamatosan kerülnek a fájlba, és értelemszerűen ugyanabban a sorrendben szükséges kiolvasni őket.

Az adatfolyam fájl elérését a System.IO.FileStream osztály támogatja. Konstruktora 9 különböző paraméterlistás változatban létezik. Mi csak a legegyszerűbbet említjük meg: ha a fájl az aktuális könyvtárban van, elég megadni a nevét, és második paraméterként a megnyitás módját.

A **FileMode** felsorolástípus a következő elemeket tartalmazza:

- **Append**: Megnyitja a fájlt, ha létezik és a végére áll, vagy létrehoz egy új fájlt.
- **Create**: Létrehoz egy új fájlt, ha már létezik ilyen nevű, azt felülírja.
- **CreateNew**: Létrehoz egy új fájlt, ha már létezik ilyen nevű, IOException kivételt dob.
- **Open**: Megnyit egy létező fájlt. Ha a fájl nem létezik, FileNotFoundException kivételt dob.
- **OpenOrCreate**: Megnyit egy létező fájlt, vagy létrehozza ha még nem létezik.
- **Truncate**: Megnyit egy létező fájlt, majd kiüríti, hogy a mérete 0 byte lesz.

```
System.IO.FileStream fs =
                    new System.IO.FileStream("jelszo.pwd",
                    System.IO.FileMode.OpenOrCreate);
```

A fájlba írást egyszerű típusok esetén a System.IO.BinaryWriter osztály támogatja. Konstruktorában meg kell adnunk a fájl objektumot, ahova írni szeretnénk.

```
System.IO.BinaryWriter bw =
                          new System.IO.BinaryWriter(fs);
```

Ezután a Write metódus 18 különböző változata segítségével a különböző típusú adatokat a fájlba a típusnak megfelelő méretű helyre írja.

```
bw.Write(str);
```

A bináris fájlból a System.IO.BinaryReader osztály objektuma segítségével tudjuk kiolvasni az adatokat az írásnak megfelelő sorrendben. A BinaryReader a különböző

típusok olvasásához különböző függvényeket kínál: pl. ReadInt32, ReadChar, ReadString...

```
System.IO.FileStream fs = new
System.IO.FileStream("jelszo.pwd", System.IO.FileMode.Open);
System.IO.BinaryReader br = new System.IO.BinaryReader(fs);
numOfUsers = br.ReadInt32();
...
fs.Close();
```

Ne feledjük el, hogy írás – olvasás után a fájlokat le kell zárni!

Megjegyzés:

A fájlkezelés további feltétele természetesen, hogy legyen megfelelő jogosultság a fájl kezelésére.

 Alkalmazása az 5. gyakorlat 12. feladatában.

5.2.5. A Control osztály és utódai

A System.Windows.Forms névtér legtöbb osztálya a Control osztályból származik. A Control az őse az összes vezérlőnek, de a Formnak is.

Súgó

Index

Control class (System.Windows.Forms)

Control overview

A Control osztály információkat szolgáltat a felhasználó számára, és biztosítja az adatbevitelt és üzenetkezelést a billentyűzet és az egér segítségével.

A Control osztály un. ambient propertiest (környező tulajdonságokat) használ, ami azt jelenti, hogy ha nem állítjuk be őket, akkor a szülőablak beállításai érvényesek rá. Ezt megfigyelhetjük, ha az ablak háttérszínét, betűszínét, kurzorát vagy fontját állítjuk, minden vezérlő háttérszíne, betűszíne, kurzora vagy fontja vele együtt változik. Ezért mutat egységes képet egy-egy ablak.

A Control osztály **Text** tulajdonsága megadja a vezérlőhöz tartozó szöveget. Ez ablakok esetén a címsor, egy nyomógombnál a gomb felirata. Mivel a Text tulajdonság a Control osztályban lett definiálva, minden vezérlő és ablak rendelkezik ezzel a tulajdonsággal, hisz örökölte azt.

Derived classes

5.3.2. Az explicit típuskonverzió

A kódrészletben jól látható, hogy a form objektum számára a valóságban Form2 méretű helyet foglaltunk, mégis, ha a form objektumra meg szeretnénk hívni egy a Form2-ben létrehozott új függvényt, azt már a gyorssúgó sem támogatja. Nem szerepel a legördülő listában a form-ra meghívható metódusok közt az új metódus. Ha mégis meghívjuk, a fordító a `'System.Windows.Forms.Form does not contain a definition for Uj'` hibaüzenetet küldi.

Ha bizonyosak vagyunk benne, hogy az objektum valójában utódtípusú területre mutat a memóriában, akkor konvertálhatjuk az utód típusára mielőtt meghívnánk a függvényt.

```
((Form2)form).Uj();
```

Ezt **explicit típuskonverzió**nak vagy **castolás**nak hívjuk, mivel explicit ki kell írni a konverziót. Ez a konverzió a programozó felelőssége! Az ő feladata, hogy megbizonyosodjék arról, hogy a form számára lefoglalt hely Form2 típusú. Ha ez nem igaz (pl. `form = new Form()`), a fordító nem jelez hibát, de futáskor a 'System.InvalidCastException' kivételt kapjuk, a `'Specified cast is not valid'` magyarázó szöveggel, ami arra utal, hogy a castolás nem érvényes.

Explicit típuskonverzióval az előbb még hibás értékadás is megvalósítható.

```
//      form2 = form;           // Hiba!
        form2 = (Form2)form;
```

Megjegyzés:

Az előre definiált típusok előre definiált konverziós lehetőségekkel rendelkeznek. Azokhoz a típusok közötti átalakításokhoz, melyek biztonságosan végrehajthatók, és soha nem eredményeznek adatvesztést (pl. int átalakítása long típussá) **implicit cast** operátorokat biztosít az osztálykönyvtár. Az implicit konverziót nem kell kiírnunk.

```
int a = 5;
long b = a; // long b = (long)a; helyett.
```

Osztálykönyvtárak használatakor alkalmazzuk az értékadás kínálta lehetőségeket. Az osztálykönyvtár által szolgáltatott függvények nem dolgozhatnak a majd általunk, a fejlesztő által később létrehozott típusú paraméterekkel vagy visszatérési értékekkel. Csak valamelyik az osztálykönyvtár által szolgáltatott ősosztály típusú

paramétert várhatják, vagy ilyen típusú objektummal térhetnek vissza. Ilyenkor, ha tudjuk az objektum által foglalt hely típusát – és használni akarjuk e típus metódusait – a castoláshoz kell folyamodnunk.

A **Form** osztályok **Controls** tulajdonsága visszaadja a Form felületére húzott vezérlők tömbjét. Mivel a Visual Studio fejlesztői nem tudhatták mi majd milyen vezérlőket használunk, no meg így mindegyikre egy ős típusokat tartalmazó tömbben hivatkozhatunk, a tömb Control típusú elemeket ad vissza. Hiába tudom, hogy a Controls[0] egy listadoboz, nem kérdezhetem le az elemeinek számát, mert a Controls osztálynak még nincs Items tulajdonsága! Csak ha konvertálom ListBox típusúra az egyébként valóban ListBox objektumot.

```
int db = ((ListBox)Controls[0]).Items.Count
```

Vagy eseménykezelők sender objektuma object típusú, de szükség esetén átkonvertálható bármely utódtípusra, feltéve, hogy tudjuk valójában egy olyan típusú utódra mutat. Az Object osztálynak nincs Text tulajdonsága, de az általa hivatkozott objektumnak lehet. Ha az adott eseménykezelőt több vezérlő is meghívhatja, és le akarjuk kérdezni a Text értékét a kódban:

```
private void Button_Click(object sender,
                                System.EventArgs e)
 {
   RadioButton r = (RadioButton)sender;
   MessageBox.Show(r.Text)
 }
```

 Az 5. gyakorlat mindkét feladatában, a Tili-toliban és a választógomboknál is láthatunk példát explicit típuskonverzióra.

A fenti kódok, ha a változók nem az konverzió során használt típusúak, kivétel dobását eredményezik.

5.3.3. Az 'is' operátor

A C# nyelv '**is**' operátorával tetszőleges objektumról eldönthetjük, hogy az adott objektum konverziója az adott típusra sikeresen végrehajtható-e. Az 'is' operátor igaz értéket ad vissza, ha az objektum futásidőben adott típusú, vagy a típus utódosztályának megfelelő területre hivatkozik.

```
form = new Form2();
string str = "A form objektum";
if (form is Form)
{
```

10. Egy osztály ősosztályairól a súgó adott osztályra vonatkozó about class bejegyzésében kapunk információt.
11. Ősosztály típusú objektum értékül kaphat utódosztály típusú objektumot.
12. A Form2 form2 = new Form(); típusú értékadás gyakran alkalmazott az osztálykönyvtárat használó fejlesztések során.
13. Ha tudjuk, hogy az Object o; objektum valójában Control-ra mutat, akkor castolással elérhetjük ForeColor tulajdonságát.
14. Az 'is' operátor csak akkor tér vissza igaz értékkel, ha a jobboldalán megadott típus megegyezik a változó deklarációban megadott típusával.
15. Ha egy ősosztályban megírt metódust szeretnénk meghívni, előbb az 'as' operátor segítségével az objektumot a kívánt ős típusára kell konvertálnunk.

Javítókulcs:

H, H, H, I, H, I, I, H, H, I, I, H, I, H, H.

6. A felügyelt kód és a Toolbox komponensek

Az alkalmazásokkal szembeni elvárások folyamatos növekedésével új és új technikákat hoznak létre a hatékonyság, skálázhatóság és áttekinthetőség biztosítása érdekében. Az elterjedt programozási nyelvek mindegyike azért tud fennmaradni a piacon, mert valamilyen speciális igény kielégítésére a többitől eltérő szolgáltatást tud nyújtani. Összetettebb alkalmazások esetén az elvárások az alkalmazás egyes területein egymástól nagyon eltérőek lehetnek. Ez azt jelenti, hogy az alkalmazás egyik része egy adott nyelv, míg más része egy másik nyelv alkalmazását kívánná. Felmerült az igény olyan fejlesztői és futtató környezet kifejlesztésére, mely a különböző nyelveken írt programelemek együttműködését támogatja. Így minden egyes részhez azt a programozási nyelvet választhatják a fejlesztők, ami annak a leginkább megfelel. A döntésben szerepet játszhat a projekt tagjainak programozási nyelv ismerete is.

Hagyományosan a forráskódú programot az adott operációs rendszerhez fordították. A C és C++ nyelv platformfüggetlenségét, hordozhatóságát az jelentette, hogy a standard C++ -nak minden operációs rendszerhez készítettek fordítót, így a forráskódot csak le kellett fordítani a különböző környezetekben, és már működött is a program. Az internet és intranet elterjedésével az állandó fordítások – melyek szakember jelenlétét és hozzáértését követelték – a működés gátjává váltak.

A Java nyelv és futtató környezet, valamint a .NET környezet azzal a céllal jött létre, hogy támogassa az alkalmazások operációs rendszertől független futtatását. A Java

objektumra később a programban mégis hivatkozunk, a felszabadított memóriaterületet már egy másik változó adatai foglalják el, mi mégis írjuk, olvassuk egy már megszűnt objektum adataként értelmezve. Ennek a hibának a felderítése rendkívül összetett feladat. Gyakran nem is vesszük észre, hogy egy most épp nem figyelt változó értéke módosult, vagy ha észrevesszük, nem értjük az okát.

A CLR biztosítja számunkra az automatikus szemétgyűjtést. Így minden .NET környezethez fejlesztő programozási nyelv, a C# is élvezi az automatikus memóriakezelés előnyeit. Mintha a programozók kaptak volna egy takarító robotot, mely összegyűjti utánuk a szemetet.

Vagyis felszabadítja helyüket, ha már nincsenek használatban. Ezután összetömöríti a megmaradt objektumokat, és kijavítja a rájuk mutató hivatkozásokat.[1] Azokat az objektumokat, melyek életciklusa ily módon szabályozott, felügyelt adatoknak (managed data) hívjuk, a memóriafelszabadítás menedzselésének ezt a módját pedig automatikus szemétgyűjtésnek **GC** (**garbage collection**) nevezzük.

Megjegyzés:

Ha nincs több hely a memóriában elindul az automatikus szemétgyűjtés. Mivel a folyamat automatikus, nincs mód arra, hogy megmondjuk, a szemétgyűjtő milyen sorrendben törölje az objektumokat.

A fent leírt előny mellett azt is tudnunk kell, hogy a szemétgyűjtés jóval több processzoridőt igényel, mint a hagyományos objektumok felszabadítása, hisz itt nem egy objektum felszabadítása zajlik, a hivatkozások felderítése és a foglalt helyek egymás mellé „tömörítése" is időigényes feladat.

Az adatok tömörítése során az objektumok helye folyton változik a memóriában, tehát felügyelt kódban nincs lehetőség a memória közvetlen címzésére, mely gyakran nagyon hatékony megoldás a kódolásban.

A C# kód alapértelmezésben felügyelt, de lehetőség van nemfelügyelt (**unmanaged**) kód írására is az **unsafe** kulcsszó segítségével. A felügyelt kód együttműködhet nemfelügyelt kóddal.

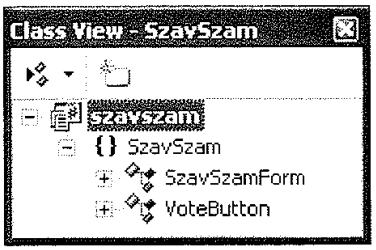
[1] David S. Platt: Bemutatkozik a Microsoft .NET, Szak Kiadó Kft, Bicske, 2001, 72. old.

6.1.3. A névterek

A .NET környezet két legfontosabb eleme a CLR és a .NET osztálykönyvtár.[1] A felhasználói igényekhez igazodva egyre több osztály szerepelt egy alkalmazásban. A .NET osztálykönyvtár sok száz osztályát rendszerezni kell ahhoz, hogy eligazodjunk közöttük. Az osztályok logikai csoportosítását szolgálják a névterek (**namespace**).

Ha megnézzük az eddig elkészült kódjainkat (Class View), minden új alkalmazás létrehozott az alkalmazás nevével azonosított névteret (SzavSzam). Ebben a **névtér**ben volt az általunk létrehozott osztályok leírása (SzavSzamForm, VoteButton). Ha a form forráskódjának elejét megnézzük, jól látható a form által gyakran használt névterek listája a **using** direktíva után.

A névterek egymásba ágyazhatók. Az egymásba ágyazott névterek nevét a '.' operátorral választjuk el.




```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

6.1. ábra A saját névterünk

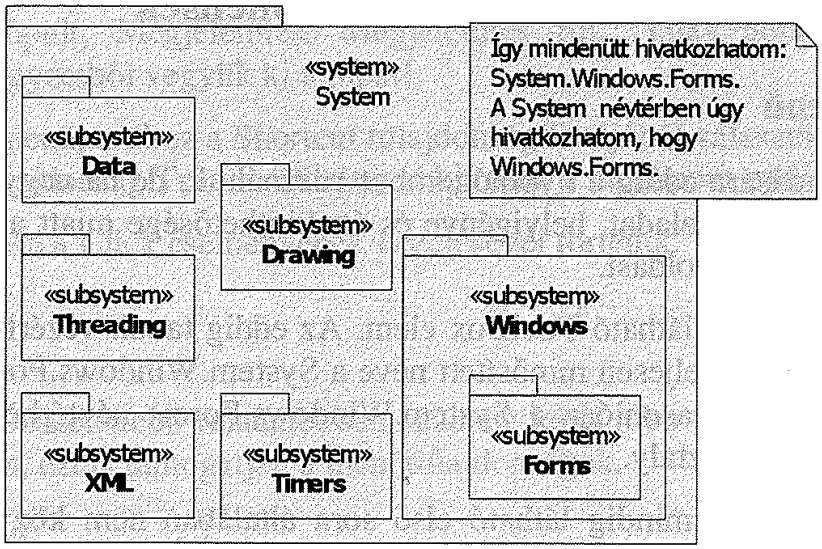
```
namespace SzavSzam
{
 public class SzavSzamForm : System.Windows.Forms.Form
}
```

Az osztályokra a **teljesen minősített nev**ükkel (**fully qualified name**) kell hivatkoznunk, amint azt a fenti kód Form osztálya esetén látjuk, ha nem a velünk azonos névtérben helyezkednek el. Mivel ez meglehetősen áttekinthetetlenné teszi a kódot, ezért importálhatjuk a névtereket a **using** kulcsszó használatával. Az importált névterek osztályai esetén elég az osztály nevét kiírnunk. A kódban gyakran használt osztályok esetén célszerű az importálást elvégezni.

A közvetlen munkatársakat mi is csak keresztnéven szólítjuk, a távolabbi munkatársat a teljes nevükön nevezzük meg, de nagyobb intézmény esetén hozzátesszük az osztályt is. „A pénzügyes Kiss Kati." Ha azonban a beszélgetés a pénzügyesekről szól, akkor elég csak Katiként megnevezni a kolléganőt, mindenki tudja kiről van szó.

---

[1] Súgó, .NET Framework Developer's Guide, Overview of the .NET Framework



6.2. ábra A System névtér néhány alnévtere az UML jelöléseit használva

A névterek nemcsak a logikai csoportosítást szolgálják, hanem lehetővé teszik azonos nevű osztályok létrehozását különböző névterekben. Ekkor az osztályokat a teljesen minősített nevükkel különböztetjük meg. Pl. nézzük meg a súgó Index fülét használva a **Button** osztály leírását! Két Button osztályt találunk, az általunk ismert és használt System.Windows.Forms névtérben leírtat, és a System.Web.UI.WebControls névtérben megadottat. De a **Timer** osztályból hármat is találunk, a System.Threading, a System.Timers és a System.Windows.Forms névtérben.

A **System** névtér tartalmazza az operációs rendszer funkcióit. A System névtérben szerepel az összes osztály ősének, az **Object**-nek a leírása. Itt találjuk a **String** osztály és a különböző típusok közötti konverziót lehetővé tevő **Convert** osztály leírását is. A System.Drawing-ból használtuk a **Color**-t, a System.Windows.Forms-ban nemcsak a **Form** osztályt, de az összes vezérlő osztályt is megtaláljuk.

Ha egy osztályról nem tudjuk, melyik névtérben található a leírása, és a fordító nem ismeri fel, akkor a súgó Index fülébe az osztály nevét beírva, az about class alcímet választva, vagy az osztály nevén F1-et ütve, az overview linket követve megtaláljuk a névtér leírását. Ezután az osztály neve elé ki kell írnunk a teljesen minősített nevet, vagy a using kulcsszóval importálnunk kell a névteret a forráskódba.

Megjegyzés:

A using direktívának meg kell előznie a névtér összes elemét! Emiatt a forráskód elején helyezzük el, különben fordítási hibát kapunk.

6.2. A ToolBox által felkínált komponensek

6.2.1. A menü

Parancsok kiadására eddig a nyomógombot használtuk, de ha nagyobb mennyiségű parancsot igényel a feladat, helyigénye és áttekinthetősége miatt a legördülő **menü** kínálja a legjobb megoldást.

A menü egy további látható ToolBox elem. Az eddig tanult vezérlőkhöz hasonlóan kezelhető. A **Menu** teljesen minősített neve a System.Windows.Forms.Menu. Tehát a többi vezérlőhöz hasonlóan a System.Windows.Forms névtérben definiálták, és őse a **Component** osztály.

A **főmenü**, a menü mindig látható első sora általában nem közvetlen parancsot, hanem végrehajtható menüpontok egy-egy csoportjának nevét tartalmazza. A főmenü valamely menüpontját kiválasztva lenyílik az **almenü**, mely már többnyire valóban parancsokhoz tartozó menüpontokat tartalmaz. Hogy egy menüpont legördülő, vagy parancsot hajt végre, azt a DefaultItem tulajdonsága határozza meg. Ha a DefaultItem True, a menüpont parancsot tartalmaz.

A menüpontok csoportosíthatók elválasztók segítségével. Elválasztót a következőképpen hozhatunk létre. Vagy a menüszerkesztőben a gyorsmenüben (jobbegér) kiválasztjuk az Insert Separator menüpontot, vagy a menüpont Text tulajdonságát '-'-re állítjuk.

Egy menüpont adhat tájékoztatást a program állapotáról is. Például gondoljunk a mindenki által jól ismert szövegszerkesztőkre! Az Edit menü Copy parancsa csak akkor használható, ha van kijelölt szöveg, Paste menüpontja pedig tájékoztat arról, van-e a vágólapon elem, amit beszúrhatunk. Ezt a menüpont Enabled tulajdonságának futásidejű állításával érhetjük el.

Ha a Checked tulajdonság igaz, egy pipa jelenik meg a menüpont előtt – csak nem legördülő menüpont esetén alkalmazható –, s ha azt akarjuk, hogy pipa helyett egy pontot lássunk, akkor a RadioCheck tulajdonságot is igazra kell állítanunk. Ne feledjük, hogy itt is illik betartani az elvet, Radio esetén a csoport pontosan egy eleme, míg pipa esetén tetszőleges számú elem lehet kiválasztva!

Menüpont eltüntethető és megjeleníthető a Visible tulajdonsággal, és természetesen szövege is módosítható a Text tulajdonság módosításával.

Előfordulhat, hogy nem tudunk egeret használni. A menüpont kiválasztása ilyenkor az Alt és az aláhúzott betű segítségével történhet. Ennek támogatásához a tulajdonságlap Text tulajdonságában az aláhúzni kívánt betű elé egy & jelet írjunk! Ekkor a szerkesztőablakban (Design) már látjuk az aláhúzást. Az, hogy a futó alkalmazásnál látjuk-e, az operációs rendszer beállításától függ. Ha fut az alkalmazásunk, és a főmenü választható billentyűzetről, akkor az Alt gomb

lenyomására megjelennek az aláhúzások. Ha azt akarjuk, mindig láthatóak legyenek, akkor a Vezérlőpult, Megjelenítés, Megjelenítés fül, Hatás gomb, Hívóbetűk elrejtése jelölőnégyzetből vegyük ki a pipát!

Gyorsgombot a menüponthoz a Shortcut tulajdonság listájából választhatunk, mely a ShowShortcut tulajdonság True-ra állításával kiíródik a menüpont mellé.

Menüpont kezelése a Click eseményének kezelésével történik.

Megjegyzés:

Összetettebb feladatokban szükség lehet több menü összefésülésére is, ezt is támogatják a menütulajdonságok.

6.2.2. A ToolTip

A Toolbox tartalmaz a futó alkalmazásban nem látható elemeket is. Ezeket, ha a Design ablak fölé húzzuk, megjelenik egy szerkesztő sáv az ablak alján, és a komponens a sávra kerül. A továbbiakban úgy dolgozhatunk vele, mintha az ablakon volna.

Húzzunk egy **ToolTip** vezérlőt az ablakunk fölé! A segédablakban megjelenő toolTip1 nevű objektumot átnevezve myToolTip-re, minden vezérlő tulajdonságlapján megjelenik a ToolTip on myToolTip tulajdonság, ahova begépelhetjük a sárga téglalapba várt szöveget. Futtatáskor vigyük az egeret a vezérlő fölé, s egy kicsit várva már olvashatjuk is az általunk megadott ToolTip szöveget.

Tulajdonságai:

- Az InitialDelay: megadja hány ezredmásodperc múlva jelenjen meg az ablak.
- Az AutoPopDelay: Hány ezredmásodpercig látható az ablak.
- A ReshowDelay: Az újramegjelenítés időtartama.
- Az AutomaticDelay: Az előző hármat állítja úgy, hogy az InitialDelay vele egyező, az AutoPopDelay a tízszerese, a ReshowDelay az egyötöde lesz.

6.2.3. A Timer

Amint a névtereknél szó volt róla, több időzítőnk is van, mi itt a **System.Windows.Forms.Timer**-rel foglakozunk. Ez a ToolBox Windows.Forms-ról húzható az ablakunkra.

Adott időközönként egy **Tick** esemény következik be. Az **Interval** tulajdonságában adhatjuk meg az időköz hosszát milliszekundumban. Tehát ha az Interval=500, fél

másodpercenként fog a Tick esemény bekövetkezni. A Tick eseményhez eseménykezelőt rendelve adhatjuk meg a végrehajtandó kódot.

Az időzítőt a **Start** metódussal indíthatjuk el, és a **Stop** metódussal állíthatjuk meg.

 A 6. gyakorlat 2. feladata az időzítő használatát mutatja.

Megjegyzés:

A System.Windows.Forms.Timer Windows.Forms alkalmazásokhoz optimalizált. Egyszálú, felhasználói felülettel rendelkező alkalmazások használják, és pontossága 55 milliszekundum. Ez a ToolBox Windows.Forms ablakából húzható a szerkesztőfelületre.

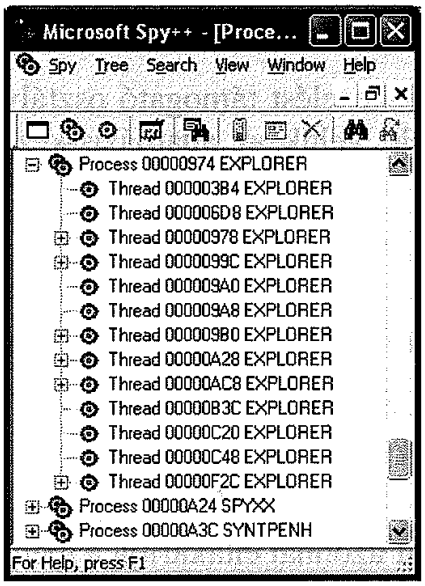
A **System.Timers.Timer** a hagyományos időzítő, szerveres környezetre optimalizált, server-based timernek is hívják. Többszálú, felhasználói felülettel nem rendelkező háttérfeladatok (workerthread) kezelésére alkalmas. Milliszekundum pontosságú. A ToolBox Components ablakában érhetjük el.

A System.Threading.Timer osztály többszálú műveletek esetén biztosítja egy függvény adott időközönkénti végrehajtását. Callback metódusok kezelésére alkalmas eseménykezelés helyett. Programkódból érhető el.

6.2.4. A Process

Folyamatok indításához, leállításához és kezeléséhez nyújt támogatást a **System.Diagnostics.Process** osztály. A folyamatok többnyire futó alkalmazások. A folyamat legalább egy, de akár több szálon is futhat. A Windows feladatkezelőjében (Ctrl + Alt + Del) megnézhetjük a gépünkön éppen futó alkalmazásokat és folyamatokat.

A Visual Studio .NET Tools menüpontjából vagy a Start menü / Visual Studio .NET / Visual Studio .NET Tools alatti Spy++ alkalmazást indítva, a Spy / Processes menüpontját választva megnézhetjük az aktuálisan futó folyamatokat és szálaikat.



6.3. ábra Az Explorer folyamat szálai és a Spy++ folyamat a Spy++ ablakában

A Process osztály statikus metódusai objektum nélkül is hívhatók. A paraméteres **Start** metódus osztályszintű, csak paraméterként meg kell adnunk az indítani kívánt program nevét. Ekkor azonban az elindított folyamat futásába a programból beavatkozni nem tudunk.

A **GetProcesses**, **GetCurrentProcess**, GetProcessesByName osztályszintű tagfüggvények visszaadják a futó folyamatok tömbjét, az aktuális folyamatot, illetve az adott nevű futó alkalmazások tömbjét adja vissza.

Process objektumot a ToolBox Components ablakából húzhatunk a Formra. A Process objektum is egy láthatatlan komponens, így a segédablakba kerül. A paraméter nélküli Start metódus a **Process.StartInfo** tulajdonságból olvassa ki az elindítani kívánt alkalmazás adatait.

A Process objektum a kódunkból elérhető, így futás közben is kezelhetjük. Lekérdezhetjük adatait, ha van felhasználói felülete, a **MainWindowTitle** tulajdonsága adja meg a főablak címsorát, és a **CloseMainWindow** segítségével állíthatjuk le a normal úton. A **Kill** metódus azonnali leállást eredményez, de az abnormal program termination adatvesztéshez vezethet. Pl. ha a futó folyamat egy Word alkalmazás, a Kill metódus úgy állítja le az alkalmazást, hogy nem menti a megnyitott és szerkesztett fájlok adatait, míg a CloseMainWindow visszakérdez, ha nem mentettük az utolsó módosításokat.

Egy folyamat **Exited** eseményéhez tudunk üzenetkezelőt rendelni. A folyamat csak akkor tud üzenetet küldeni, ha az **EnableRaisingEvents** tulajdonsága true-ra van állítva.


A folyamatok kezelésére a 6. gyakorlat 1. feladata mutat példát.

Megjegyzés:

Az adatbázis-kezelést támogató osztályok is futásidőben láthatatlan komponensek, velük az adatbázis-kezeléssel foglalkozó fejezetben találkozunk.

6.3. Teszt

1. A .NET futtatókörnyezet biztosítja az automatikus szemétgyűjtést.
2. A fejlesztőeszköz a CLS segítségével köztes kódra fordítja az alkalmazást, melyet a CTS értelmez az adott környezetben.
3. Az automatikus szemétgyűjtő folyamatosan dolgozik a háttérben, és a nem hivatkozott objektumokat azonnal eltakarítja a memóriából.
4. Ha egy osztályt nem akarunk mindig a teljesen minősített nevével megadni, akkor a kód elején a namespace kulcsszóval importálhatjuk a névterét.
5. A különböző névterekben azonos nevű osztályokat is megadhatunk, sőt akár egy programban is használhatjuk őket, de ilyenkor a nevükön kívül a névterüket is meg kell adnunk (fully qualified name).
6. Írhatunk a felügyelt kódon kívül unmanaged kódot is, de egy alkalmazáson belül a kettő nem keveredhet.
7. A felhasználói felületen meg nem jelenő komponenseket csak kódból érhetjük el, a ToolBox a látható elemek szerkesztését segíti.
8. A menüt a program állapotának kijelzésére is használhatjuk.
9. A menü megjelenése az alkalmazás futása során változtatható.
10. A ToolTip segítségével a Design módban megnyitott ablak felhasználói felületére kényelmesen húzhatunk be vezérlőket.
11. A Timer.Interval tulajdonságában másodpercekben adhatjuk meg a Tick esemény bekövetkezésének gyakoriságát.
12. A Process osztály statikus metódusai lehetővé teszik, hogy alkalmazásunk folyamatosan vezérelje a statikus metódus segítségével létrehozott folyamatot.
13. A folyamatok Kill metódusa lehetővé teszi az adatok mentését a folyamat lezárása előtt.

Javítókulcs:

I, H, H, H, I, H, H, I, I, H, H, H, H.

7. A DLL és a szerelvény

Osztályaink logikai csoportosítását a névterek tették lehetővé. Azonban ilyen sok osztályt nem lehet minden alkalmazásba befordítani, mert akkor nagyon nagyméretűek lennének. Alkalmazásaink nem használják az osztálykönyvtár által kínált összes lehetőséget, csak a probléma megoldásához szükségeseket. Ezért tehát célszerű osztályainkat fizikailag más-más fájlban, úgynevezett libraryben tárolni, és a fordítás során csak a szükségeseket a kódhoz szerkeszteni.

Ha több fizikai fájlban tároljuk az alkalmazást, lehetővé válik azok különálló fejlesztése. Lehetőség nyílik rá, hogy egy DLL-t több alkalmazás használjon, és a fejlesztés és karbantartás során a DLL-eket önálló verziószámmal ellátva azok önálló frissítése is megvalósítható.

7.1. A DLL

A **DLL** (**dynamic-linked library**) olyan könyvtár, melyet dinamikusan szerkeszthetünk alkalmazásunkhoz. Bár .NET esetén az exe csak akkor indul el, ha minden dll rendelkezésünkre áll, futás közben a szükséges library-t olvassa be az alkalmazás. Ez a következő előnyökkel jár:

- Csökken az exe fájl mérete.
- Ugyanazt a dll-t több alkalmazás is használhatja akár egyszerre is. Ehhez elegendő egyszer tárolni.

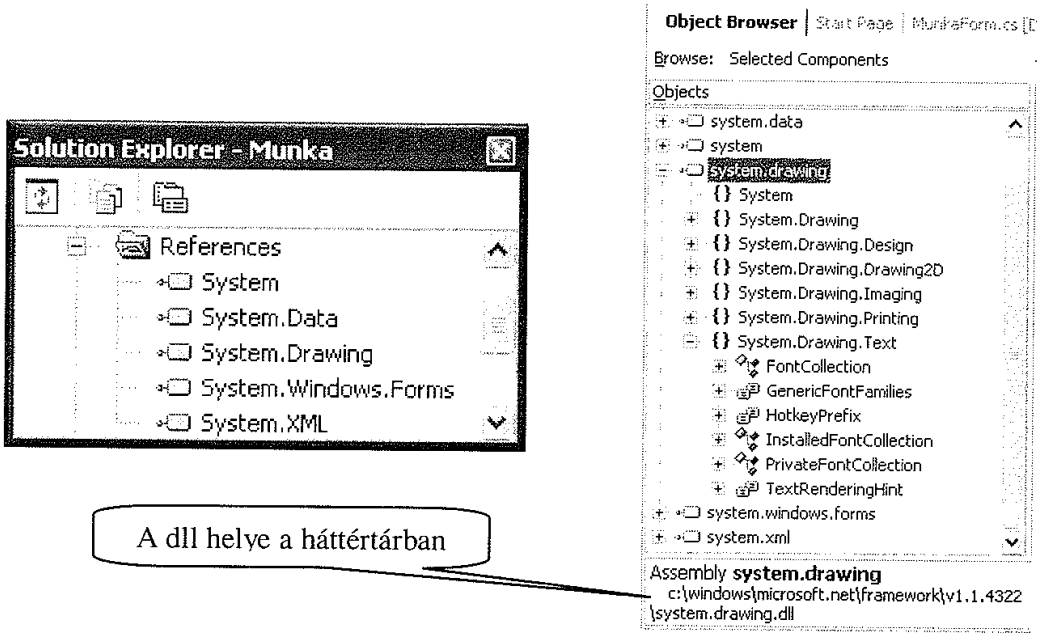
- A kód fordításakor nem kell pl. az összes meghajtó kódjával rendelkeznünk, futás közben az éppen használt eszköz telepített meghajtóját használja kódunk. Pl. nem kell tudnunk a felhasználó nyomtatójának adatait ahhoz, hogy alkalmazásunk az adott nyomtatón nyomtatni tudjon.

- A hibás dll önállóan javítható, frissíthető anélkül, hogy az őt használó alkalmazásokhoz nyúlnánk. Ezt tesszük pl. a Windows frissítések letöltése és telepítése során.

Tudnunk kell viszont, ha a dll-t elfelejtjük az exe-vel együtt telepíteni, nélküle nem futtatható, vagy csak részben fog működni alkalmazásunk. Ezért nem lehet egyszerűen az exe fájl másolásával programokat elvinni egy gépről. A részleges működésre jó példa, hogy amíg egy gépre nincs nyomtató meghajtó telepítve, addig a nyomtatási funkciók nem működnek. Nincs nyomtatási kép, nem lehet jelentést készíteni Excelben...

A dll önállóan nem futtatható állomány. Kell hozzá egy exe, hogy a szolgáltatásait elérjük. Ezért először a kész dll felhasználásának módját tanulmányozzuk, hisz az általunk készített dll teszteléséhez is szükségünk lesz erre a tudásra.

7.1.1. Az eddig fejlesztett alkalmazások komponensei



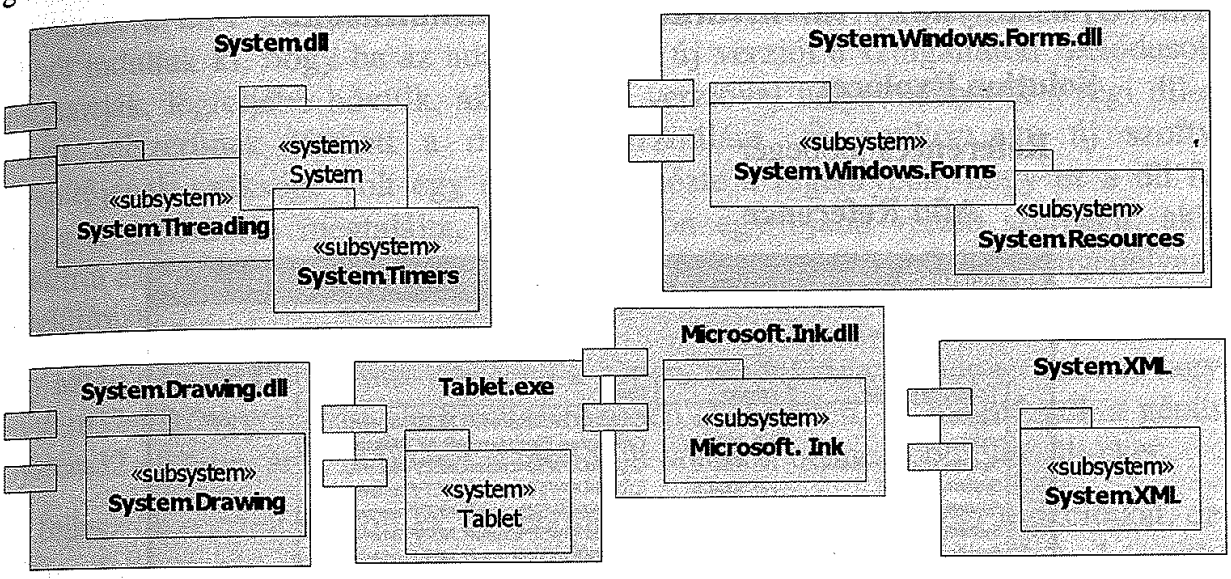
7.1. ábra A References alatt és az Object Browserben az alkalmazásban használt osztálykönyvtárak

Az eddig készített alkalmazásaink is használtak osztálykönyvtárakat, csak eddig a kódgenerátor előkészítette nekünk a használatukat. A Solution Explorer ablakban a References folder tartalmazza az alkalmazásunkba beszerkesztett osztálykönyvtárakat. Ha egy kiválasztott elem gyorsmenüjében a Properties

menüpontot választjuk, a Description tulajdonság alatt megtaláljuk a dll kiterjesztésű fájl nevét, a Path tulajdonság alatt pedig az elérési útját.

Az Object Browser ablakban megnézhetjük, hogy mely névterek mely osztályait fordították be az adott dll-be. Kiválasztva egy osztályt, a mellette levő ablak kilistázza az osztály tagjait.

A logikai csoportosítás a névterekben és a fizikai csoportosítás az osztálykönyvtárban többnyire fedi egymást, de nem kötelezően. Vagyis az egyes névterek osztályai jellemzően egy osztálykönyvtárban vagy végrehajtható állományban tárolódnak, és egy library vagy exe néhány névtér osztályait tárolja. A 7.2. ábra komponensdiagramja a System névtér és néhány gyakran használt alnévterének, a Microsoft.Ink névtérnek, valamint a Tablet.exe alkalmazásban generált Tablet névtérnek a fizikai komponenseit mutatja.



7.2. ábra A névterek és fizikai tárolásuk exe és dll fájlokban

7.1.2. A keresett osztály osztálykönyvtára

Amikor olyan osztályt kívánunk használni a projektünkben, amit a fordító nem ismer fel, de mi a súgóból tudjuk, hogy a .NET osztályok egyike, akkor először meg kell keresnünk a névterét, hogy vagy a using direktívával importáljuk, vagy megadjuk a teljesen minősített nevét. Ezt tettük az előző fejezetben a Process osztály Diagnostics névtérénél. Ha a fordító változatlanul nem találja az osztályt vagy névterét, annak az az oka, hogy az osztályt tároló osztálykönyvtár nincs a projektünkhöz csatolva.

Az osztályt tároló dll-ről a súgó Index ablakába az osztály nevét írva, az about osztálynév elemet választva kaphatunk információt. Az ablak alján, a Requirements címszó alatt pl. a Process osztály esetén a következőt olvashatjuk:

Requirements

Namespace: System.Diagnostics

Platforms: Windows 98, Windows NT 4.0, Windows Millennium Edition, Windows 2000, Windows XP Home Edition, Windows XP Professional, Windows Server 2003 family

Assembly: System (in System.dll)

Információt kapunk a névtérről, melyben az osztály definiálva lett, az operációs rendszer platformról, amelyen használható, és az osztálykönyvtárról, melyben fizikailag tárolódik.

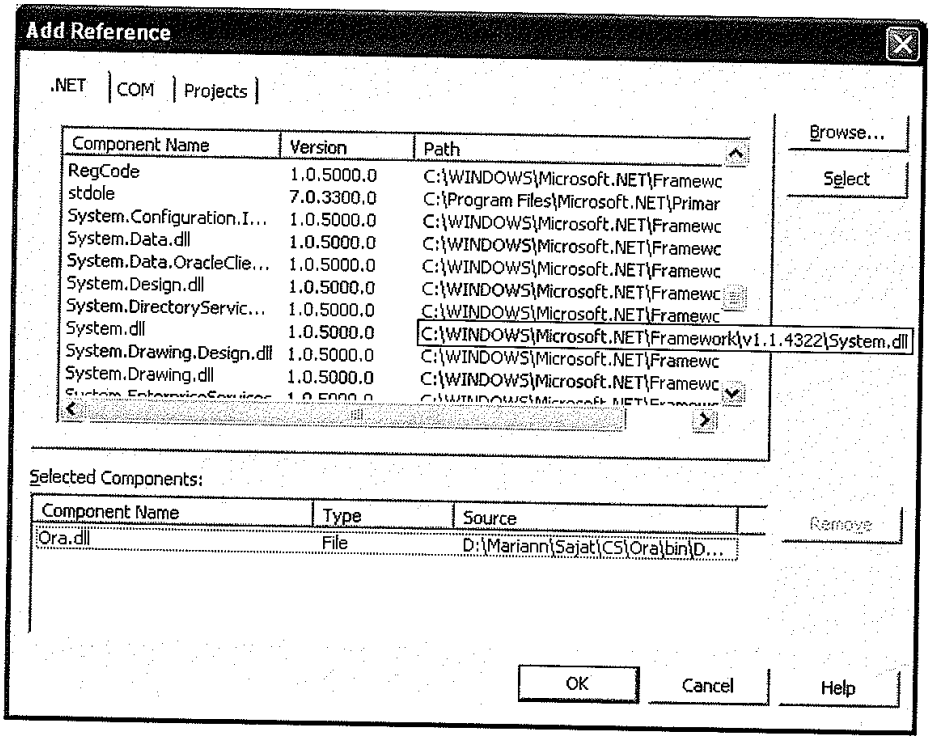
Ha azt akarjuk, hogy projektünk hozzáférjen az osztálykönyvtárhoz, fel kell vennünk a referenciák közé!

7.1.3. Új osztálykönyvtár-hivatkozás az alkalmazásban

Solution Explorer

References jobbegér

Add Reference



7.3. ábra A .NET komponensei az Add Reference ablakban

A párbeszédablakban három füllel találkozunk:

- .NET fül választásakor a felügyelt kódú osztálykönyvtárakat választhatjuk ki.

- COM fül alatt a régebben létrehozott COM komponenseket, melyek nem felügyelt kódúak.
- A Projects fül az adott gépen általunk létrehozott dll-eket kínálja föl.

A kiválasztott komponenst a Select gomb segítségével adhatjuk a Selected Components listához. A lista kijelölt elemét a Remove gombbal távolíthatjuk el.

Ha valahonnan letöltött, vagy egy meghajtón rendelkezésünkre álló dll-el szeretnénk dolgozni, azt a Browse gomb segítségével érhetjük el.

Az OK gomb hatására az osztálykönyvtár bekerül projektünk referenciái közé.

7.1.4. A hagyományos dll

Ugyanazt a dll-t jellemzően több alkalmazás használta (használja). A fejlesztők abból indultak ki, hogy ha az adott dll új és új verziói a szolgáltatott metódusokat megtartják, legfeljebb bővítik, akkor az új verzióval helyettesíthetők a régi dll-ek. Ezért célszerűnek látszott a dll-ek regisztrálása, és amikor egy új szoftver telepítőjének szüksége volt egy dll-re, megnézte a registryben, hogy ez a dll fent van-e a gépen. Ha megtalálta, és a telepítőn egy újabb verzió szerepelt, akkor felülírta az előzőt. Többnyire visszakérdezett, hogy a telepítő személy engedélyezi-e a felülírást.

Így aztán minden dll egyszer szerepelt a gépen, többnyire a Windows system vagy a system32 alkönyvtárában platformtól függően. XP esetén, ha megnézzük a gépünk system32 alkönyvtárát, 5000-nél is több fájlt találunk benne, többnyire dll-eket.

A fejlesztők tesztelték a dll-ek új verzióit, hogy megőrizték-e régi funkcióikat, de a széles felhasználói kör minden igényét nem tudták tesztelni. A hibák, bár ritkán fordultak elő, a tömeges használat miatt mégis egyre gyakrabban jelentkeztek.

Minden új program telepítésekor jónéhány dll felülíródott anélkül, hogy ez a felhasználóban tudatosult volna. És ez ritkán, de hibákhoz vezetett. Gyakran nehezen felderíthető hibákhoz. Mert elég nehéz rájönni, hogy a hónapok óta jól működő, de ritkán használt alkalmazásunk azért nem fut, mert a múlt héten kipróbáltunk egy játékprogramot, ami felülírta az általa használt dll-t. Már rég elfelejtettük a játékot, le is töröltük a gépről, de az eddig használt szoftver érthetetlen hibáját a gyenge játék hibás dll-je okozza.

További problémát okozott, hogy a közös dll-eket közös könyvtárba szokás telepíteni (system, system32). Ha nem használunk tovább egy alkalmazást, és nem az eltávolítóval töröljük le, akkor dll-jei ott maradtak a gépen. Másrészről a fejlesztő gépén a közös könyvtárból használt dll-t, ha nem telepítették és regisztrálták a felhasználó gépén, akkor ott már nem működött az alkalmazás.

A .NET előtti dll-ek nem köztes kódra fordultak, hanem natív kódra, tehát a JIT nélkül is azonnal felhasználhatók voltak.

A feladat tehát adott: úgy megvalósítani a dll cseréket, hogy ha akarjuk helyettesítsék egymást, de ha nem akarjuk, akár egymás mellé több verziót is feltehessünk! Ez utóbbi helypazarló, de biztonságos megoldás, és ma már sokkal nagyobbak a háttértárak, mint néhány évvel ezelőtt. A felügyelt kód erre is megoldást kínál az assembly létrehozásával.

7.2. A szerelvény

A **szerelvény** (**assembly**) egy vagy több fájl együttese, melyeket közös verzió információval látunk el, és együtt telepítünk. Az assembly a .NET alkalmazás első építőeleme, melynek egyik legfontosabb feladata a kód biztonságának biztosítása. Ha a Solution Explorerben vagy az Object Browserben a dll-ek előtti ikonra mutatunk, a ToolTip ablak assemblynek nevezi őket. Egy dll vagy egy exe gyakran egy szerelvényt alkot.

7.2.1. Az internal láthatóság

Az **internal** láthatóságú tagok csak az adott szerelvényből érhetők el. Tehát bővebb a hozzáférésük, mint a **private** tagoknak, hisz az assembly többi osztályai is hozzáférnek, de szűkebb, mint a **public** tagokénak, amiket bármely más assemblyből is elérhetünk.

Ha a **protected** láthatósághoz hasonlítjuk, akkor abban az esetben, ha az utódosztály és az ősosztály egy assemblyben van, akkor az internal hozzáférés a bővebb, hisz az assembly nem utódosztályai is hozzáférhetnek. Azonban, ha az utódosztály másik assemblyben található — pl. a Form a System.Windows.Forms.dll assemblyben, míg a MyForm a My.exe assemblyben —, akkor a protected láthatóságú tagokhoz hozzáfér az utód, míg az internalokhoz nem.

Ezért van szükség a **protected internal** láthatóságra, melyhez az assemblyből és az utódosztályokból is hozzáférhetünk.

Az új láthatóságok bevezetése a kód biztonságát szolgálja.

7.2.2. A managed kód

A köztes kódra fordított osztálykönyvtár előnye a különböző operációs rendszerek különböző platformjain történő futtathatóságon kívül az, hogy az egyes nyelveken írt alkalmazások nemcsak objektumok létrehozására használhatják osztályaikat, hanem akár saját osztályokat is származtathatnak belőlük.

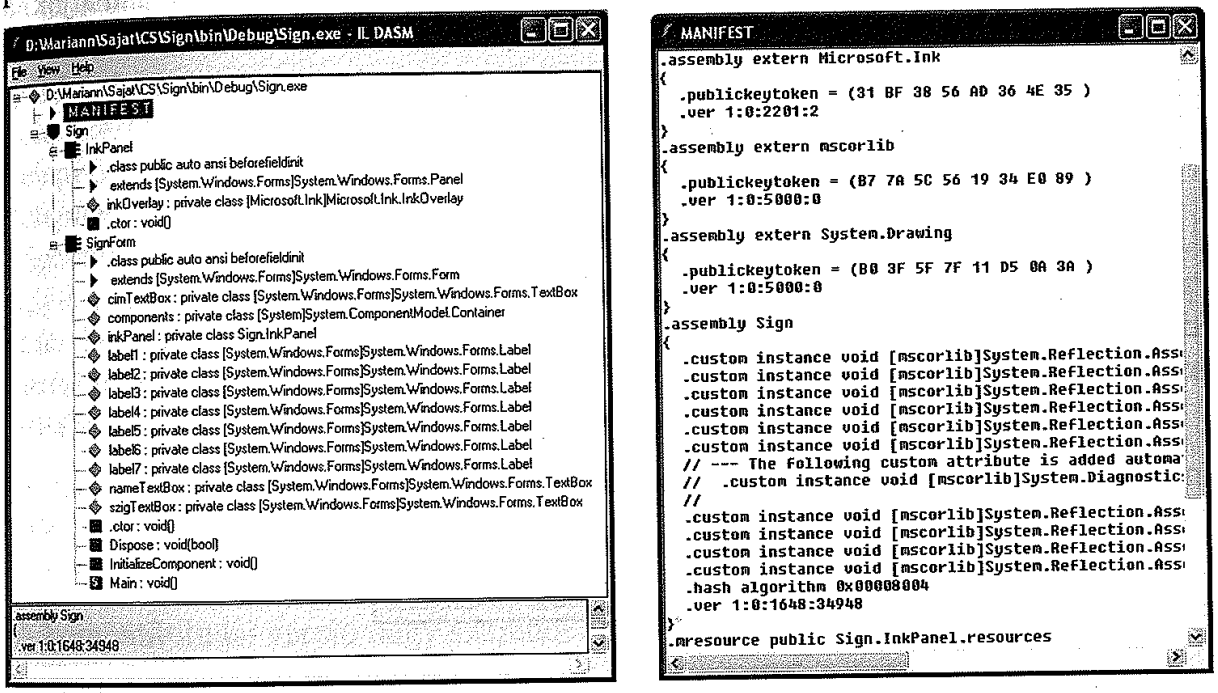
Ahhoz, hogy a CLR biztosítani tudja szolgáltatásait, a nyelvi fordítóknak metaadatokat kell a kódba írniuk.

7.2.3. A manifeszt

„Az assembly az alkalmazás kódját és erőforrásait tartalmazó EXE vagy DLL-fájl(ok) logikai gyűjteménye. Az assembly **önleíró adat**okat, ún. **manifeszt**et is tartalmaz, amely az assemblyben található kód és erőforrások leírása **metaadat**ok formájában. Az assembly gyakran egyetlen fájl – EXE vagy DLL –. Egy projekt általában egy assemblynek felel meg. Bár az assembly gyakran egy fájlban található, nem ritka eset, hogy az assembly több, azonos könyvtárban található fájl logikai (nem fizikai) gyűjteménye. Az assemblyt alkotó fájlokat leíró manifeszt vagy az assembly egyik EXE- vagy DLL-állományában található, vagy pedig maga külön, önmagában alkot egy EXE- vagy DLL-fájlt."[1]

A szerelvény tehát felügyelt kódú dll, exe fájl vagy ezek gyűjteménye, mely a futtatáshoz szükséges adatokon kívül metaadatokat is tartalmaz.

A szerelvény manifesztjét az ILDASM.exe segítségével nézhetjük meg. Az exe fájlt a legegyszerűbben a Start menüből indítható Visual Studio .NET Tools alatti Visual Studio .NET Command Prompt ablakból indíthatjuk úgy, hogy begépeljük a nevét a parancssorba.



7.4. ábra A Sign alkalmazás manifesztje az ILDASM-ben

Köztes kódú fájl nem fog végrehajtódni, ha nincs assembly manifestje.

[1] David S. Platt: Bemutatkozik a Microsoft .NET, Szak Kiadó Kft, Bicske, 2001, 51-52. old.

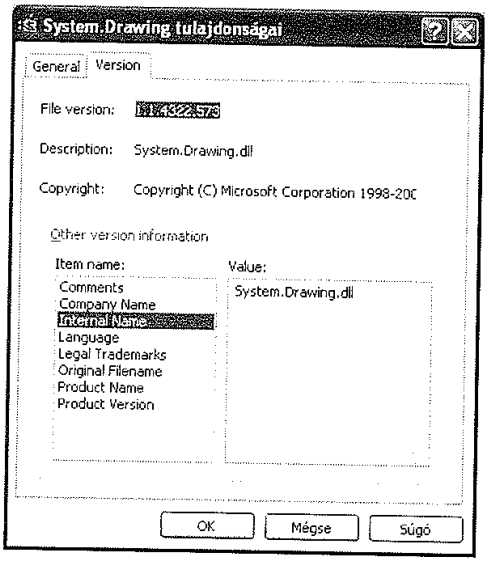
7.2.4. Privát és osztott szerelvény

A **privát szerelvény** (**private assembly**) csak a vele azonos könyvtárstruktúrából (többnyire azonos könyvtárból) érhető el, csak egy alkalmazás hivatkozhat rá. Ez a szerelvény készítésekor az alapértelmezés.

Az **osztott szerelvény** (**shared assembly**)t több alkalmazás is használhatja. Az osztott szerelvénytt a **global assembly cash**-ben (**GAC**) tároljuk, és **erős névvel** (**strong name**) látjuk el. A global assembly cash egy könyvtár, a Windows / assembly nevű könyvtára. Ha megnézzük, itt találjuk a System névterek dll-jeinek szerelvény fájljait.

Minden szerelvénynek van neve, verzió száma és kultúra információja. A kultúra információ opcionális. Az erős név az osztott assembly azonosítója. A néven, a verzió számon, és a kultúra információn kívül egy nyilvános kulcsú azonosítóból és digitális aláírásból áll.

Ha megnézzük az eddigi alkalmazásaink hivatkozott system névtereinek tulajdonságait, láthatjuk, hogy Copy Local tulajdonságuk false, és Strong Name tulajdonságuk True. Ha pedig kinyitjuk a GAC könyvtárat, akkor láthatjuk a nyilvános kulcsokat is a Public Key Token oszlopban.



7.5. ábra A System.Drawing assembly tulajdonságai a Windows / assembly könyvtárban

Ha viszont egy saját dll-t hivatkozunk az alkalmazásunkban, akkor annak alapértelmezett Strong Name tulajdonsága False, és Copy Local tulajdonsága True. A saját assembly alapértelmezésben egy privát assembly, hiába is keressük a GAC-ben. A privát assembly dll fájlját célszerű az alkalmazással azonos könyvtárban elhelyezni, mert telepítéskor, törléskor ez leegyszerűsíti a feladatokat. (Nézzük meg az alkalmazás bin / Debug alkönyvtárát, ott lesz a Sajat.dll!) Ha nem másoljuk

azonos könyvtárba, vagyis a Copy Local tulajdonság False lesz, akkor nem készül másolat a Sajat.dll-ről az alkalmazás könyvtárában, de akkor biztosítanunk kell, hogy a telepített gépen azonos legyen a fejlesztői géppel a könyvtárstruktúra, és azonos elérési úttal rendelkezzen a Sajat.dll. Ez intranetes alkalmazások esetén, amikor egy vállalat minden gépe azonos módon lát egy könyvtárat praktikus lehet, máskor nagyon körülményes feladat.

7.2.5. Verzió szám

Ahhoz, hogy az együttműködő komponenseket kezelni tudjuk, meg kell különböztetni az egyes változatokat a fejlesztés során. Erre a feladatra alakult ki a **verziószám**. Az újabb komponens magasabb verziószámot kapott. A programozásban íratlan szabály, hogy a magasabb verziószámú, tehát újabb komponens nyújtsa a régi szolgáltatásokat. Ennek a szabálynak a szoftverek jelentős részben megfelelnek, azonban az egyre összetettebb felhasználás során egyre gyakrabban fordul elő, hogy bizonyos speciális esetekben az új verzió nem nyújtja a régi szolgáltatásainak egy kis részét.

A közösen használt dll-ek esetén a .NET előtti időkben az új verzió megjelenésével a régi verzió felülírása volt a szokás. Így nem tároltuk ugyanazt a kódot több példányban a gépen. A kompatibilitási problémák azonban egyre gyakrabban jelentkeztek. Új programok telepítésekor az is előfordult, hogy az új program telepítője régebbi verziójú dll-t tartalmazott, és bár rákérdezett, hogy felülírja-e a telepítési hibák miatt gyakran felülírták a magasabb verziószámú dll-t egy régebbivel. Ennek hatására az újat kívánó program nem működött helyesen. Az egyik program telepítése elrontotta a másik működését. A hiba gyakran a telepítés után hosszú idővel jelentkezett, így senki nem gondolt erre az okra. Az ilyen problémák egyre több felhasználó és fejlesztő idejét töltötték ki bosszantó hibakereséssel.

A .NET környezethez a Microsoft egy szabványosított verziókezelést valósított meg, mely lehetővé teszi a különböző verziók egyidejű tárolását nem csak privát assembly esetén, de a GAC-ban is.

A kompatibilitási verzió négy számból áll.

- major version – fő verziószám
- minor version – másodlagos verziószám
- build number – fordítás sorszám
- revision – revízió szám

A verziószámot szerelvény **manifeszt**jében[1] találjuk. A fejlesztés során az AssemblyInfo.cs fájl tartalmazza a verzió létrehozásához szükséges információkat. A build number és a revision helyettesíthető *-al is. Ez esetben a fordító generálja értéküket.

```
[assembly: AssemblyVersion("1.0.*")]
```

A manifesztben található verzió a fordítás után pl.: .ver 1:0:1716:19100.

Az assemblyt használó másik szerelvény fordítása során szintén a manifesztben megadja a szükséges assemblyk verziószámát. Így futtatáskor pontosan lehet tudni, hogy az esetlegesen több azonos nevű csak verziószámban különböző assemblyből melyikre van szükség az adott program futásakor (8.1. ábra).

Az assembly technikával biztosíthatjuk, hogy az adott osztott szerelvény az alkalmazások által közösen használt legyen, sőt a verzióinformációval még azt is meghatározhatjuk, mely verziótartományon belül helyettesíthető, és mely esetben nem. Másrészről a saját dll-jeinket nem kell kitennünk a felülírás veszélyének, ha privát assemblyként hozzuk létre őket. Ez tehát növeli a biztonságot, és egyszerűsíti a telepítést és törlést, hisz egy alkönyvtár összes fájlját másoljuk a telepítés során, és összes fájlját töröljük az alkalmazás eltávolításakor. A hagyományos registry regisztráció .NET alatt nem szükséges.

7.3. Teszt

1. A névterek hierarchiája egyfajta logikai csoportosítása az osztályoknak, a dll-ek egy másik logika szerint csoportosítják őket.
2. A dll dinamic-linked libraryt jelent, vagyis a dll-ben tárolt osztályok kódja nem növeli az exe fájl méretét, és futásidőben az éppen aktuális dll töltődik be a memóriába.
3. A dll metaadatok nélkül is futtatható, csak akkor a futás elején meg kell adnunk az adatok értékét.
4. A dll osztályait bármely .NET programozási nyelvből használhatjuk objektumok létrehozására, de származtatni belőlük csak ugyanazon a nyelven lehet, amiben íródtak.
5. A privát assembly dll-t telepítéskor regisztráltatni kell a regisztryben.
6. Hibás dll cseréjekor újra kell fordítani az őt használó alkalmazásokat.
7. Az exe az általa használt dll nélkül nem vagy csak részlegesen működik.

---

[1] A manifesztről bővebben az Osztálykönyvtár készítése, 8.1.2 szakaszában olvashatunk.

8. A Visual Studioban a Solution View References alatt találjuk az adott projekt által használt dll-eket.
9. A privát assemblyt a GAC-ben tároljuk.
10. Az internal láthatóságú tagok az assemblyből hozzáférhetőek, azon kívülről nem.
11. A protected internal láthatóságú taghoz hozzáférhetünk az assemblyt használó alkalmazás minden olyan osztályából, mely a tagot tartalmazó osztály utóda.

Javítókulcs:

H, I, H, H, H, H, I, I, H, I, I.

8. Osztálykönyvtár készítése

Az előző fejezetekben megismertük a komponensalapú programozás elvét, és készítettünk dinamikus osztálykönyvtárakat (dll) felhasználó alkalmazásokat. Ebben a fejezetben megtanulunk osztálykönyvtárat készíteni.

8.1. A Class Library fogalma

Azokat az osztályokat, amelyeket több alkalmazásból is használni szeretnénk, **osztálykönyvtár**ban gyűjtjük össze. A dinamikusan szerkesztett osztálykönyvtárak biztosítják számunkra, hogy ugyanazt a dll-t több alkalmazás használhassa akár egyidőben is.

Az osztálykönyvtár önmagában nem futtatható állomány, arra szolgál, hogy osztályait különböző futtatható alkalmazások tudják felhasználni.

Az osztálykönyvtár **köztes kód**ra (MSIL) vagy natív kódra fordított osztályok gyűjteménye. A **névterek** az osztályok logikai, hierarchiába rendezett csoportosítását teszik lehetővé. Az osztálykönyvtárban több névteret vagy névterek egy részét is tárolhatjuk. Jellemzően a gyakran együtt használt névterek kerülnek be egy dll-be. Az osztálykönyvtár feladata a kód minél kisebb méretének biztosítása is, tehát a gyakran használt osztályokat célszerű néhány osztálykönyvtárban implementálni, és a bizonyos esetekben használtakat külön dll-ben megvalósítani. Így érhetjük el, hogy lehetőleg minél kevesebb nem használt osztályt szerkesszünk be alkalmazásunkba.

8.1.1. Az osztályok láthatósága

A C# nyelvben nemcsak az osztályok tagjai rendelkeznek láthatósággal, hanem az osztályok maguk is. Az osztály alapértelmezett láthatósága internal. Ez azt jelenti, ha nem írunk az osztály deklarációban az osztály elé láthatóságot, akkor osztályunk internal láthatóságú lesz. Ez esetben az osztályt csak az assemblyből érhetjük el. Osztálykönyvtárunk osztályait, amiket mások számára is elérhetővé kívánunk tenni, public láthatósággal kell ellátnunk!

Osztály az internal és public láthatóságon kívül más láthatósággal nem rendelkezhet.

8.1.2. Elnevezési konvenciók

Már eddigi fejlesztéseink során is betartottunk elnevezési szokásokat, de osztálykönyvtár fejlesztésekor hangsúlyozottan figyelni kell ezekre, hisz az osztálykönyvtár azért készül, hogy mások is felhasználhassák.

8.1.2.1. A kis- és nagybetűk használata

- PascalCase

  Az azonosító első betűje és a szóösszetétel minden tagjának első betűje nagybetű, a többi kicsi.

  Névterek System.Windows.Forms, osztályok InkPanel, metódusok GetChildIndex(), tulajdonságok BackColor esetén használjuk.

- camelCase

  Az azonosító első betűje kicsi, a szóösszetétel minden tagjának első betűje nagybetű, a többi kicsi.

  Privát láthatóságú adattagok mint billButton, függvényparaméterek mint sender, lokális változók mint ar. Az ajánlás szerint a public és protected láthatóságú példányokat javasolt tulajdonsággal megvalósítani, ami 'Pascal case' írásmódú. Ez megfelel az objektumorientált programozás egységbezárás elvének.

- UPPER case

  Az azonosító csupa nagybetű.

  Pl. System.IO, System.Web.UI[1]

Megjegyzés:

Sajnos a fejlesztői környezet (egyelőre) nem támogatja az elvek betartását, mert ha kisbetűs privát tagváltozóhoz eseménykezelő

[1] MSDN Library, Visual Studio .NET 2003, Capitalization Styles.

metódust rendelünk, annak neve is kisbetűvel kezdődik, ha át nem írjuk. És lustaság fél egészség, nem szoktuk átírni.

Ne használjuk ki a nyelv kínálta case sensitivitást az azonosítók elnevezésében! Az osztálykönyvtárunk akkor lesz teljes értékű, ha használható lesz case-sensitive és case-insensitive nyelvekből egyaránt. Vagyis ne készítsünk azonos betűkből álló, csak a kis- és nagybetűkben különböző azonosítókat![1]

Ha betartjuk az egységbezárás elvét, akkor az a szokás, hogy C#-ban a kisbetűvel kezdődő (tag)változó neve megegyezhet a nagybetűvel kezdődő osztályának nevével, vagy a hozzá kapcsolódó nagybetűs tulajdonság nevével (pl. BackColor backColor), nem mond ellent annak az elvnek, hogy ne használjunk azonos neveket, melyek csak a betűk kis és nagy voltában különböznek. Az osztálykönyvtár felhasználói ugyanis csak a public, vagy protected láthatóságú tagokhoz férnek hozzá, az egységbezárás elve szerint viszont a kisbetűs tagváltozó legyen private, ami az osztálykönyvtáron kívülről nem látható. A kisbetűs lokális változók sem láthatók kívülről. A függvényparamétereknél már oda kell figyelni erre az elvre!

8.1.2.2. Azonosító kezdetek és végződések

Az interfészek azonosítója I betűvel kezdődik, IDisposable.

A kivételosztályok Exception taggal végződnek, WebException.

Az eseménykezelők neve a vezérlő nevével kezdődik, amin az eseményt kiváltottuk, és '_' az esemény nevével fejeződik be, `BillButton_Click` lenne a konvenciók betartásával, de a kisbetűs `billButton`-ból `billButton_Click`-et generál a Visual Studio.

Lehetőség szerint kerüljük az azonos azonosítók használatát! Pl. a névtér neve ne egyezzen az osztály nevével! Ezt úgy kerülhetjük el, ha használjuk a névben az osztály ősére vonatkozó azonosítót. Sign névtér, SignForm, de lehetne SignButton, SignPanel, SignView stb. osztály is. Az azonos név használata alól kivétel a konstruktor, melynek neve kötelezően azonos az osztály nevével. De a függvényképző operátor, a ( ) megkülönbözteti őket.

8.2. Class Library készítése

8.2.1. Osztálykönyvtár létrehozása

File menü

New

[1] MSDN Library, Visual Studio .NET 2003, Case Sensitivity.

Project

Project types: Visual C# Projects

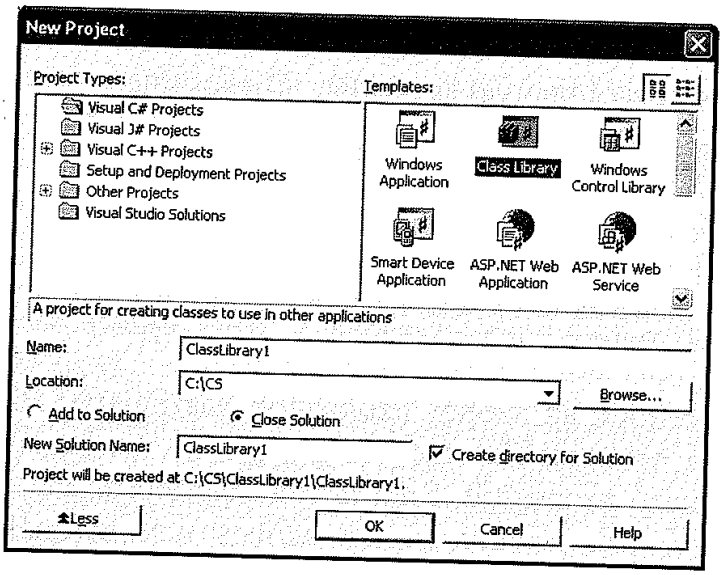
Templates: Class Library

Name: MyClassLibrary

Location: Többnyire az előzőleg beállított

Add to Solution / Close Solution

Eddig mindig Close Solution-t választottunk, mert önálló alkalmazást fejlesztettünk. Most viszont a Class Library önállóan nem futtatható, tehát kell neki egy futtató környezet. Ha a futtató alkalmazást és az osztálykönyvtárat egymástól függetlenül kívánjuk kezelni, akkor a Close Solution választása az indokolt. De ha azt szeretnénk, hogy egyszerre dolgozzunk mindkettőben, és mindkettő forduljon le, ha módosult a kódja, és most épp a tesztalkalmazás van betöltve, akkor választhatjuk az Add to Solution választógombot. Vagy **most a Close Solution**-t, de a tesztalkalmazás létrehozásakor az Add to Solution-t.



8.1. ábra Osztálykönyvtár projektjének generálása

A Visual Studio a következő kódot generálja:

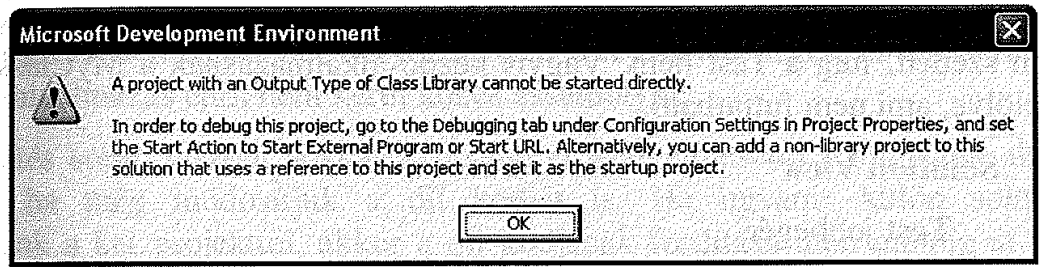
```
using System;

namespace MyClassLibrary
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>
 public class Class1
 {
```

```
        public Class1()
        {
                //
                // TODO: Add constructor logic here
                //
        }
    }
}
```

Ha megnézzük, a kód csak a System névteret importálja, és a References is kevesebb osztálykönyvtárra hivatkozik. A legfeltűnőbb a System.Windows.Form hiánya. Mivel egy osztálykönyvtár magában nem futtatható, nem kell rendelkeznie felhasználói felülettel. Természetesen, ha akarjuk, rendelkezhet vele. Nem kell mást tennünk, mint a referenciákhoz hozzáadni a System.Windows.Forms.dll-t, és a using System.Windows.Forms; sort a kód elejére beírni.

A kód egy üres osztálydeklaráció egy üres konstruktorral. Az osztály nevét még módosítanunk kell! Az osztály kódját sem lehet helyettünk megírni.

Ha a megszokott Start paranccsal akarjuk felépíteni és futtatni a könyvtárat, az Output ablak `Build: 1 succeeded, 0 failed, 0 skipped` üzenete után a következő ablak jelenik meg. (Ha nincs tesztalkalmazás, vagy nem a teszt a kezdő projekt.)



8.2. ábra Hibaüzenet ablak, ha az osztálykönyvtárat futtatni akarjuk

A hibaüzenet tájékoztat bennünket, hogy osztálykönyvtár közvetlenül nem futtatható, de ha egy nem könyvtár projektet hozzáadunk a Solution-höz, és azt beállítjuk kezdő projektként, akkor megoldódik a probléma.

A hibaüzenetet elkerülhetjük, ha a Start parancs helyett a Build menü Build Solution parancsát, vagy a projekt nevén (MyClassLibrary) jobbegérrel kattintva a legördülő menüben a Build parancsot adjuk ki. Ezt minden alkalommal megtehetjük, amikor a fejlesztés alatt még nem akarunk tesztelni, de szintaktikai hibáinkat ellenőrizni akarjuk.

8.2.2. Egy solution több project

Készítsük el a tesztalkalmazás projektjét! A tesztalkalmazás a szokásos Windows Application, csak most célszerű az Add to Solution gombot választani a generáláskor.

Ha elfelejtettük volna az Add to Solution választást, vagy egy már létező alkalmazásban kívánjuk használni osztálykönyvtárunkat, utólag a következőképpen tehetjük a két projektünket egy solutionbe:

Solution View

Solution 'MyClassLibrary' (1projekt) jobbegér

Add

Existing project

Az ablakban egy csproj kiterjesztésű fájlt kell választanunk.

Az alkalmazás a már ismerős Form1.cs [Design] ablakkal jelenik meg. Ha futtatni akarjuk megoldásunkat, bár az Output ablak immár azt írja ki, hogy `Build: 2 succeeded, 0 failed, 0 skipped,` mindkét projekt sikeresen lefordult, mégis megjelenik a hibaüzenet ablak.

Ha ránézünk a Solution View ablakunkra, azt látjuk, hogy a MyClassLibrary vastagon szedett, míg a Test nem. Vagyis kezdő alkalmazásnak a MyClassLibrary van kijelölve, ami nem futtatható.

Solution View

Test jobbegér

Set as StartUp Project

Ezzel a Test alkalmazást állítottuk be kezdőnek, így már futtathatjuk is az alkalmazást. Most a Test van vastagon szedve.

Bár most mindkét alkalmazást felépíti a Visual Studio, és futtatja is a Test-et, de még nincs köze az osztálykönyvtárunkhoz. Ez a rész viszont már az előző fejezet ismétlése, hisz ott megtanultuk az osztálykönyvtárak használatát.

8.2.3. Az osztálykönyvtár használata

A Test References folderéhez hozzá kell adnunk a MyClassLibrary hivatkozást!

Solution View

Test

References jobbegér

Add Reference

A Projects lapon megtaláljuk a MyClassLibrary-t. Ha nem akarjuk kiírni a teljesen minősített osztálynevet, akkor még a using MyClassLibrary; sorral a Class1 osztály névterét is importáltuk a Test projektbe.

Ezzel Test alkalmazásunk kódja használhatja a Class1 osztály public tagjait.

Bár most már felvettük a referenciák közé a MyClassLibrary-t, az be is másolódott a Test exe-vel azonos könyvtárba, de ha megnézzük a Test.exe manifestjét, nem látjuk benne az osztálykönyvtárat. A fordító csak akkor teszi bele, ha a kódból hivatkozunk is az osztályra. Vegyünk fel például a konstruktorba egy lokális objektumot a Class1 osztályból!

```
public Form1()
{
 InitializeComponent();

 Class1 obj = new Class1();
}
```

Ha most megnézzük a Test.exe manifestjét, most már benne lesz a MyClassLibrary assembly hivatkozás.

A két projekt közös solutionbe helyezése csak a fejlesztést segíti, az osztálykönyvtár és a tesztalkalmazás ettől nem kerül közös assemblybe.

A fejlesztés során viszont sokkal gyorsabb és kényelmesebb a munkánk abban az esetben, ha még módosítjuk a dll kódját is. Ha ugyanis külön solution-be dolgoznánk a két projekttel, akkor a dll kódjának fordítása után mindig át kellene tölteni azt a másik solutionbe. Vagy úgy, hogy a Visual Studioban váltogatjuk a betöltött projekteket, vagy úgy, hogy párhuzamosan két Visual Studio ablakban dolgozunk, de akkor a tesztelő alkalmazásból minden módosítás után ki kell venni az előző dll-t, majd hivatkozni kell a módosított dll-t. Mindkét eset nagyon kényelmetlen és időrabló.

8.3. Teszt

1. Az osztályok, ha nem írunk eléjük láthatóságot, akkor privát hozzáférésűek.
2. Az osztálykönyvtár azon osztályait, melyet más alkalmazásokból el akarunk érni – public láthatóságúra kell deklarálni.
3. A dll lehet assembly, az assembly nem mindig dll.
4. Az osztálykönyvtár azért íródik, hogy osztályait több alkalmazásból felhasználhassuk.

5. Ha olyan public azonosítókat használunk, melyek csak abban térnek el egymástól, hogy kis- vagy nagybetűvel írtuk őket, akkor azok case-insensitive nyelvekből nem lesznek használhatóak.
6. A Form osztályok nevében szerepelnie kell a Form végződésnek (SignForm JelszoForm), mert ennek hiányában a fordító nem tudja, hogy itt egy Form osztályról van szó.
7. Ha a Solution-ben egy osztálykönyvtár van kijelölve StartUp projektnek, akkor futtatáskor hibaüzenetet kapunk.
8. Ha a Solution több projektből áll, akkor a közös fordítás során egy közös assembly keletkezik, mely tartalmazza az összes projekt fordításakor keletkező állományt.
9. Amikor felveszünk egy dll-t a projekt referenciái közé, akkor a projekt assemblyjének manifesztjébe bekerül a dll hivatkozása.
10. A dll önállóan nem futtatható állomány, ezért a generálása során az Add to Solution-t kötelező választanunk.

Javítókulcs:

H, I, I, I, I, H, I, H, H, H.

9. Windows vezérlő készítése

A Windows Control Libray vagy vezérlő könyvtár, amint a nevében is benne van, az osztálykönyvtár egy speciális esete. Amíg az osztálykönyvtár tetszőleges osztályt, osztályokat tartalmazhat, amelyek a felhasználó kódjából elérhetőek, addig a Control Library kifejezetten egy vezérlő kódját tartalmazza. Ez nem jelenti azt, hogy a kód nem tartalmazhat a háttérben megírt további osztályokat, sőt összetett vezérlő esetén további vezérlőket is, a könyvtár célja mégis az új több alkalmazásban használható vezérlő létrehozása.

9.1. A Windows Control Library

Új vezérlőt új projekt létrehozásakor a Windows Control Library template választása segítségével készíthetünk.

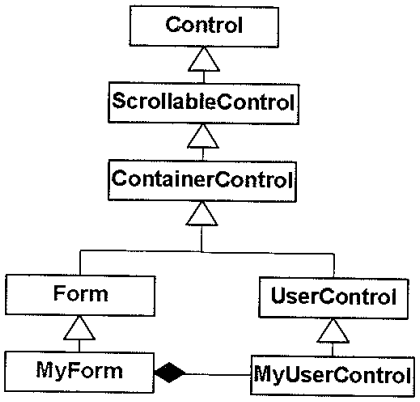
9.1.1. A vezérlő felhasználói felülete

Lényeges különbség az osztálykönyvtár létrehozásához képest, hogy a generált projekt egy felhasználói felület szerkesztővel – [Design] ablak – jelentkezik be. A vezérlő ugyanis, mint a neve is mutatja, az alkalmazás lefutásának vezérlésére alkalmas osztály. Feladata, hogy irányítsa az alkalmazás végrehajtásának módját. Napjaink felhasználója igényli, hogy beavatkozhasson az alkalmazás végrehajtásába, ehhez a vezérlőnek felhasználói felületet kell kínálnia.

A Visual Studio a vezérlők felhasználói felületének szerkesztésére ugyanazt a környezetet biztosítja, amit már megszokhattunk a formok szerkesztése során.

9.1.2. A vezérlő osztály őse

Ha a forráskódot nézzük (jobbegér, View Code), láthatjuk, hogy a `WindowsControlLibrary1` névtérben létrehozott `UserControl1` osztályunk a `System.Windows.Forms.UserControl` osztály utóda. Tehát nem egy egyszerű üres konstruktorral ellátott osztályt generált a Visual Studio, hanem egy olyan osztályt, mely felhasználói felülettel rendelkezik, és biztosítja a UserControl osztály szolgáltatásait.



9.1. ábra A UserControl osztály ősei

A UserControl osztály őse – a Form osztályhoz hasonlóan – a ContainerControl, melynek őse a ScrollableControl, annak pedig a Control. Az ősöknek köszönhető, hogy a Form-hoz hasonlóan tud vezérlőket kezelni, és öröklött Controls tulajdonságán keresztül vezérlői akár tömbként is kezelhetőek, amint arról az 5.2.4 szakaszban már olvashattunk.

Az ősosztályok biztosítják az üzenetküldő mechanizmust, egér, billentyűzet, valamint más eseményekhez való hozzáférést.

9.1.3. Saját vezérlő felépítése

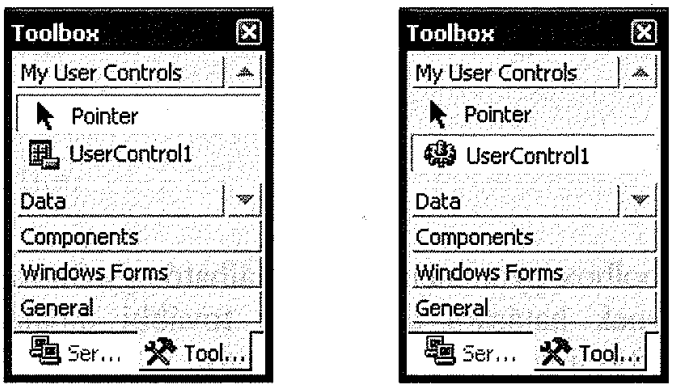
A Windows Control Library az osztálykönyvtárhoz hasonlóan nem futtatható önállóan. Az assemly készítéshez használjuk a Build menü Build parancsait vagy a WindowsControlLibrary1 Build parancsát, mely létrehozza az alkalmazásokból felhasználható dll állományt.

Megjegyzés:

A Visual Studio .NET segítségével létrehozott vezérlő komponens a dll kiterjesztést kapja. Azonban a COM alapú ActiveX vezérlők, jelezve, hogy speciális osztálykönyvtárról van szó, ocx kiterjesztésűek. Tehát, ha egy ocx kiterjesztésű fájllal találkozunk tudjuk, hogy egy ActiveX vezérlőt tartalmazó osztálykönyvtárat látunk.

9.2. Saját vezérlő felhasználása

Vezérlő könyvtárunk alkalmazásból történő felhasználását legegyszerűbben egy tesztalkalmazással próbálhatjuk ki. Ismerve a projektek közös solutionben történő kezelésének előnyeit, célszerű a fejlesztés során tesztalkalmazásukat a vezérlő könyvtárral közös solutionben – esetleg a vezérlő alkönyvtára alatt – létrehozni.



9.2. ábra A UserControl1 vezérlő a Toolboxban baloldalt egy solution esetén, jobboldalt, ha hozzáadjuk a Toolbox eszközeihez

Ha ezt a technikát használjuk, akkor tesztalkalmazásunk Toolboxának My User Controls kategóriája alatt meg is jelenik a saját vezérlőnk, amint azt a baloldali ábra mutatja.

Ha a tesztalkalmazásunkat kezdő projektként állítjuk be, vezérlőnket a form felületére húzva már ki is próbálhatjuk az új vezérlőt.

Azonban, ha más alkalmazást indítva kinyitjuk a Toolbox My User Controls kategóriáját, abban nem lesz bent az új vezérlő.

Azt is nézzük meg a kódban, hogy a vezérlő formra húzásával a tesztalkalmazás references listájába bekerült az új vezérlőt leíró dll!

9.2.1. Saját vezérlő a Toolboxban

Tetszőleges alkalmazás Toolbox ablakának My User Controls kategóriáját nyissuk ki! Az egér jobbgombjával a menüt legördítve:

Add / Remove Items

A .NET és a COM komponensek a megfelelő fület választva, a saját vezérlők a Browse gomb segítségével érhetők el.

A Toolbox ábra jobboldali képe mutatja a saját vezérlőt a Toolboxban, miután felvettük a vezérlők közé.

Ezután a beállítás után minden alkalmazás hozzáfér a vezérlőnkhöz egészen addig, míg ki nem töröljük a Toolboxból.

Ezzel a technikával természetesen a Toolbox bármely kategóriájába felvehetünk vezérlőket, a saját vezérlőket mégis külön kategóriába szoktuk tenni.

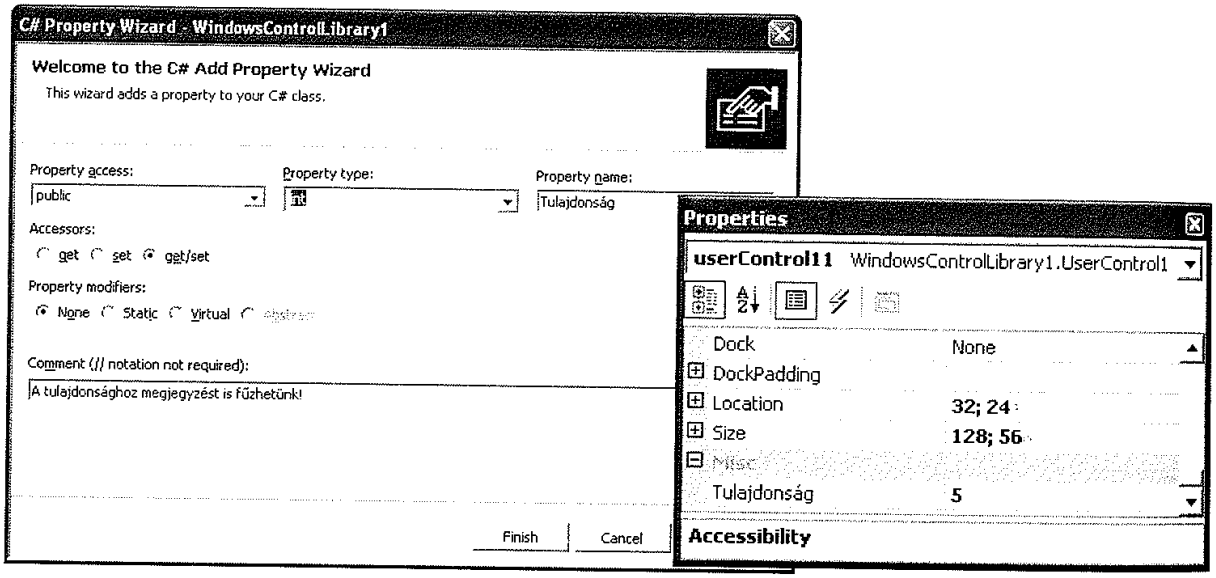
! Vegyük észre, hogy itt további, a Toolboxban eddig nem látott vezérlők állnak rendelkezésünkre! Például a COM kategóriában rendelkezésünkre áll egy olyan **ProgressBar** vezérlő, melynek Appearance kategóriában megjelenő Scrolling tulajdonsága ccScrollingSmooth értékűre állítható, s így folyamatos kijelzésű lesz.

Megjegyzés:

A Toolbox gyorsmenüjében található egy Show All Tabs menüpont, melynek hatására a Toolbox további oldalai jelennek meg. A beállítás ugyanitt ki is kapcsolható.

9.2.2. A saját vezérlő tulajdonságai

A Visual Studio fejlesztő környezet szolgáltatása, hogy a vezérlők felhasználásakor tulajdonságaikat megjeleníti egy Properties ablakban. A tulajdonságok kezdeti beállításai szerkeszthetők a Properties ablakban.



9.3. ábra Tulajdonság létrehozása a vezérlő projektben, és megjelenése a tesztalkalmazásban

```
private int t;

/// <summary>
/// A tulajdonsághoz megjegyzést is fűzhetünk!
/// </summary>
public int Tulajdonság
{
 get
 {
        return t;
 }
 set
 {
        t=value;
 }
}
```

Megjegyzés:

A tulajdonság csak az első fordítást követően jelenik meg a vezérlő tulajdonságlapján. Ez érthető, hisz a felhasználó kód a dll-ből építkezik.

Ha nem vettünk fel saját tulajdonságot a vezérlőnkhöz, öröklött tulajdonságai akkor is beállíthatók a tulajdonságlapon. A beállított tulajdonság kezdőértéke az `InitializeComponent()` kódjában ellenőrizhető.

```
private void InitializeComponent()
{
 this.userControl11.Tulajdonság = 5;
}
```

Megjegyzés:

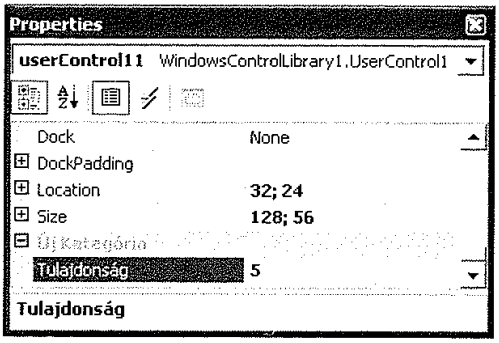
A tulajdonság kezdőértéke csak akkor állítható be, ha van a tulajdonság mögött tagváltozó, ami az értéket tárolni tudja.

A tulajdonságok a tulajdonságlapon kategóriánként vagy abc sorrendben jelennek meg a megfelelő eszközgomb választásától függően. Az általunk felvett tulajdonság a Misc kategóriába kerül. Ha mi akarjuk megválasztani a tulajdonság kategóriáját, azt a kódban kell elé írnunk!

```
[Category("Behavior")] public int Tulajdonság
```

Új kategóriát is létrehozhatunk:

```
[Category("Új Kategória")] public int Tulajdonság
```



9.4. ábra A tulajdonság kategóriája egy új kategória

9.2.3. Saját vezérlő ikonjának beállítása

Ezt a **System.Drawing.ToolboxBitmapAttribute** osztály segítségével tehetjük meg. Az osztály konstruktora egy 16x16 képpontos bitmapet vár, mely erőforrásként van beépülve a projektbe.

A ToolboxBitmapAttribute osztály GetImage metódusa egy kis 16x16-os vagy egy nagy 32x32-es képpel tud visszatérni, melyet a kis képből készít.

Adjunk a vezérlő projektünkhöz egy 16x16 pixeles bitmapet!

A projekt nevén jobbegér

Add

Add Existing Item

Files of type: Image Files kijelöljük a fájlt (net.bmp)

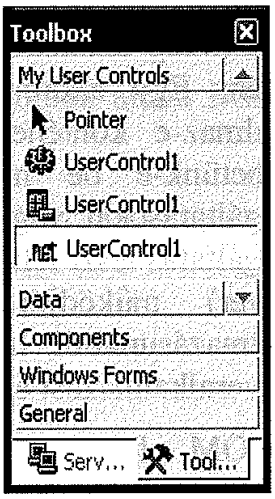
Open

Állítsuk a net.bmp állomány **Build Action** tulajdonságát **Embedded Resource**-ra! A tulajdonság beállításának hatására felépítéskor a kép beépül az erőforrás-állományba.

Ezután írjuk a vezérlőosztályunk deklarációja elé:

```
[ToolboxBitmapAttribute(typeof(UserControl1),"net.bmp")]
public class UserControl1 :
                    System.Windows.Forms.UserControl
{
...
}
```

A dll felépítése után az ikon csak akkor változik, ha a Toolbox-ba újra beszúrjuk az új komponenst. A Costumize Toolbox ablak listájában már szerepel a UserControl1, de az egy régebbi verzió. A Browse gomb segítségével újra fel kell vennünk a listára, és legyen kijelölve ahhoz, hogy bekerüljön az eszközgombok közé! Láthatóan az új vezérlő ikonja az általunk készített kép. Sőt, a vezérlő kijelölésekor a szerkesztőfelületen a kurzor mellett is megjelenik az ikon.



9.5. ábra A vezérlőnk a három ikonnal a Toolboxban

Az egy solutionben használt vezérlő frissül a tesztalkalmazásban. Ez a Solution szolgáltatása. A Toolboxban már kész vezérlőket szokás felvenni, s azok a felvétel pillanatában aktuális verziójú vezérlők. Ha módosítunk rajtuk, az előző verzió törlése után vagy amellé az új verziót is fel kell vennünk, hogy használhassuk alkalmazásaink fejlesztéséhez.

Megjegyzés:

Az általunk írt projektekben alapértelmezésben teljesül, de nem árt tudni, hogy a működés feltétele a szerelvény (assembly) és a névtér nevének megegyezése. A fent leírt beállítások bármelyikének hiánya hibához vezethet. A hiba következtében eltűnhet a vezérlő a szerkesztő [Design] ablakból és az alkalmazásból. A hiba javítását a gyakorlat leírásában találjuk.

A gyakorlat 5. feladatának megoldásában látunk példát a megvalósításra.

9.3. Alkalmazások beépítése a programba

Az eddigi fejezetekben volt szó arról, hogyan indíthatunk más alkalmazásokat programunkból, és hogyan tudjuk őket a mi alkalmazásunkból kezelni. Az előzőekben pedig láttuk, hogyan készíthetünk és használhatunk a Toolboxból olyan komponenseket, melyek egy vezérlőt valósítanak meg.

A Toolbox egy további szolgáltatást is nyújt a fejlesztőknek. Lehetővé teszi beágyazott alkalmazások indítását az általunk fejlesztett programból. Technikailag a megvalósítás nem tér el a vezérlő felvételétől a Toolboxba.

A Toolbox kívánt kategóriáját megnyitva, a gyorsmenüből az Add/Remove Items menüpontot választva a .NET vagy a COM oldalon találjuk a rendelkezésünkre álló komponensek listáját.

Tartalmilag azonban ebben az esetben másról van szó, mint a vezérlő felvétele esetén. Egy alkalmazás fejlesztésekor beállítható, hogy az adott alkalmazás tartalmaz-e konténer (container) szolgáltatásokat – vagyis a fejlesztés során építhetünk-e be más alkalmazásokat –, illetve szerveralkalmazás (server) szolgáltatásokat, vagyis alkalmazásunk beépíthető-e más alkalmazásokba. Ez utóbbi szolgáltatás teszi lehetővé, hogy egy alkalmazás más alkalmazások kiszolgálójaként (server) működjön. Ez esetben tehát nem egy vezérlőt használunk alkalmazásunkból, hanem egy speciális – vezérlő tulajdonságokkal rendelkező – szerveralkalmazást építünk be mint szolgáltatást.

A COM oldal listáján található a Windows Media Player komponens. Ennek segítségével a Windows Media Player alkalmazás szolgáltatásait vehetjük igénybe a saját alkalmazásunk egy ablakában. Lejátszhatunk vele videofájlokat és hangfájlokat. Hangfájlok esetén ha az objektum láthatóságát (Visible tulajdonság) hamisra állítjuk, akkor használhatjuk az eszközt az alkalmazásunk alatt futtatott hangfájlok indítására anélkül, hogy a felhasználó tudná, hogy a hangokat a Windows médialejátszója segítségével hallja.

 A gyakorlat 8. feladatában látunk példát használatára.

COM komponens a Microsoft webböngésző is, amivel internetes oldalakat tehetünk be az alkalmazásunk egy ablakába.

 A gyakorlat 10. feladatában látunk példát használatára.

9.4. Teszt

1. Minden class library Windows control library, de nem minden Windows control library class library.
2. A Windows control library nem futtatható önállóan.
3. A saját vezérlőt nem mutatja a Toolbox, a References listára felvéve a kódból hivatkozhatunk rá.
4. Ha a saját vezérlő projekt és az azt felhasználni kívánó projekt egy solutionben van, akkor a Toolbox mutatja a saját vezérlőt.
5. A Toolboxba felvehetők további, a Microsoft által rendelkezésünkre bocsátott vezérlők.
6. A saját vezérlő objektumának tulajdonságai megjelennek az objektum Properties ablakában, és kezdőértékük beállítható.
7. A tulajdonság a Misc kategóriába kerül alapértelmezésben, de megadhatjuk kategóriáját, mely csak a létező kategóriák valamelyike lehet.

8. Tulajdonság ikonjának megadásához egy 32x32 pixeles bitmapre van szükség.
9. A Toolboxba bármely dll felvehető, csak a Costumize Toolbox ablak Browse gombjával meg kell keresni.

Javítókulcs:

H, I, H, I, I, I, H, H, H.

10. Adatbázis-elérés

Az eddigi egyszerű problémák megoldása során is többször fölmerült bennünk a gondolat, jó lenne az alkalmazásba bevitt adatokat tárolni, és a későbbi futtatások során felhasználni. Az adatokat a forráskódban is felvehetjük, de az adatbázisban tárolt adatok a szoftver rugalmasságát növelik. A forráskód adatai fordításkor rögzülnek, míg az adatbázisban tárolt adatok a felhasználás során módosíthatók.

Az adatok adatbázisokban történő tárolása az informatika önálló tudományterülete, ennek elméletével és gyakorlatával a könyv nem foglalkozik. Csupán annak bemutatására vállalkozik, hogy a Visual Studio fejlesztői környezet milyen szolgáltatásokat nyújt az adatok elérésében és feldolgozásában. A .NET osztálykönyvtár osztályai segítségével hogyan férhetünk hozzá a legegyszerűbben egy C# alkalmazásból az adatbázisban tárolt adatokhoz.

Két különböző adatbázis-kezelőt vizsgálunk, ezzel azt is bemutatva, hogy a különböző adatbázisok valóban egységes felületen érhetők el. A Microsoft SQL Server, mint a nevéből is kiderül, az adatok szerverben történő tárolását és elérését támogatja, míg az Access adatbázis fájl a Data Connections alatt érhető el.

Microsoft ajánlás szerint kis, néhány felhasználós alkalmazások esetén használhatjuk a Microsoft SQL Server X.X Desktop Engine-t, mely a Microsoft Office XP Premium része, vagy a Visual Studio 6-os verziójának telepítő CD-in megtalálható. A Desktop Engine a Visual Studioból az SQL szerverhez hasonlóan használható. Optimális felhasználás esetén legfeljebb 5 egyidejű felhasználó

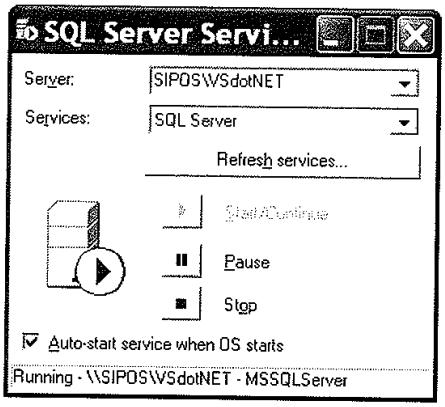
csatlakozása engedélyezett, mely a fejlesztéshez és teszteléshez elegendő. A segítségével létrehozott adatbázis módosítás nélkül futtatható SQL Server alatt.[1]

Megjegyzés:

A Microsoft Campus csomag tartalmaz egy Microsoft SQL Server Desctop Engine-t. Telepítéséhez a Visual Studio .NET telepítése során ki kell jelölni ezt az elemet. Később a Visual Studio.NET / Add or Remove Features / SQL Server Desctop Engine / Update Now segítségével készíthetjük elő a telepítést. Majd a telepítőcsomag 3. CD-jén a Program Files \ Microsoft Visual Studio.NET \ Setup \ MSDE \ setup.exe (SQL Server Setup Microsoft Corporation) fájl futtatásával telepíthetjük.

Az SQL Server a gép újraindítása után automatikusan elindul. Default neve VSdotNET. Futását az asztal jobb sarkában egy ikon jelzi.

Telepített SQL Server esetén a tálca jobb szélén megjelenik az SQL Server Service Manager. Ha zöld nyilat látunk rajta, akkor fut az SQL Server. Az eszköz segítségével leállítható vagy elindítható.

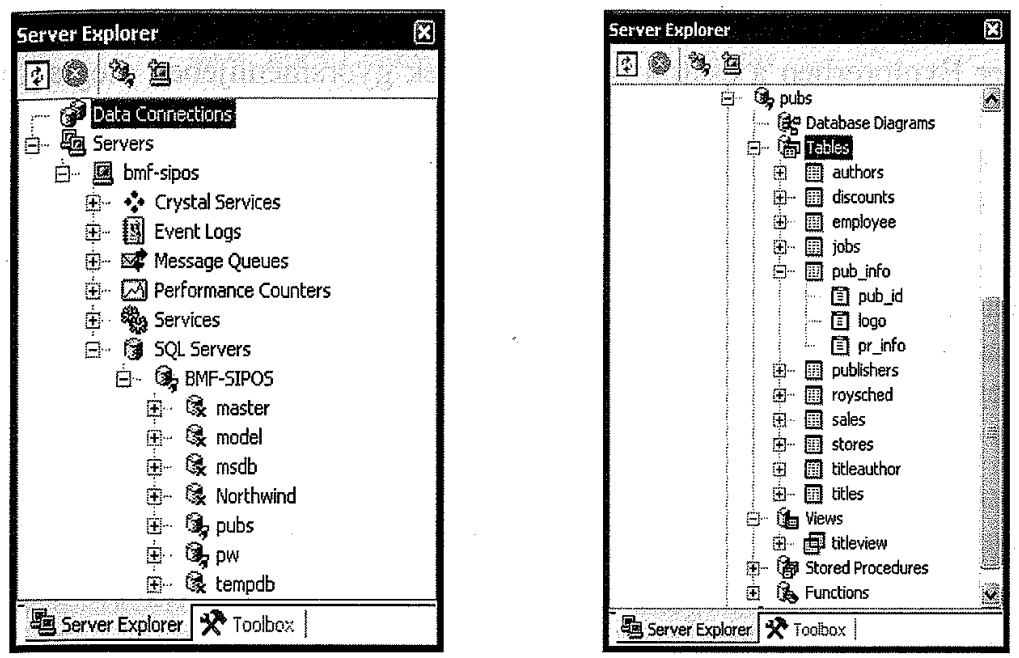


10.1. ábra Az SQL Server fut

A Microsoft SQL Server, ha fut a gépünkön, vagy egy számunkra elérhető szerveren, akkor a Visual Studio **Server Explorer** segítségével elérhető. A Server Explorer a Visual Studio ablak bal szélén a Toolbox ikon mellett található. Képe két monitor. Ha nincs megnyitva az ablak, a View menü Server Explorer menüpontjával nyithatjuk ki.

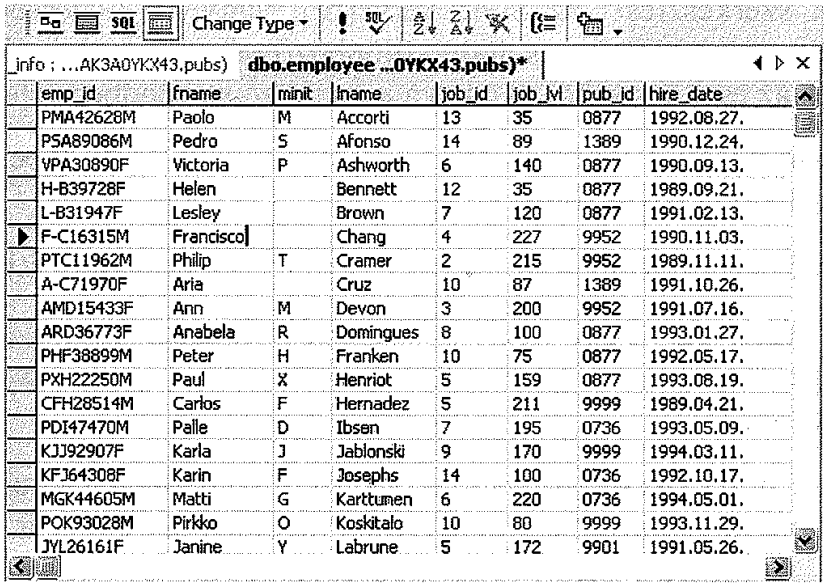
[1] MSDN Library for Visual Studio .NET, Selecting a SQL Server Data Source. 2001.

A Server Explorer ablak mutatja a gépen futó, vagy távoli elérésű szervereket és adatkapcsolatokat. A futó szerverek közt találhatók az SQL szerverek és azon belül a konkrét SQL Server. Neve alapértelmezésben azonos a számítógép nevével.



102. ábra A Solution Explorer szerverei és az SQL Server pubs adatbázisa

A jobboldali ábra a pubs adatbázist mutatja a táblákkal, a táblákon belül a mezőnevekkel. Egy mezőnevet (oszlopot) kiválasztva a Properties ablakban megnézhetjük a típusát, méretét. A Views alatt a nézetek felsorolása következik a mezőikkel, majd a tárolt eljárások paramétereikkel és mezőikkel.



emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date
PMA42628M	Paolo	M	Accorti	13	35	0877	1992.08.27.
PSA89086M	Pedro	S	Afonso	14	89	1389	1990.12.24.
VPA30890F	Victoria	P	Ashworth	6	140	0877	1990.09.13.
H-B39728F	Helen		Bennett	12	35	0877	1989.09.21.
L-B31947F	Lesley		Brown	7	120	0877	1991.02.13.
F-C16315M	Francisco		Chang	4	227	9952	1990.11.03.
PTC11962M	Philip	T	Cramer	2	215	9952	1989.11.11.
A-C71970F	Aria		Cruz	10	87	1389	1991.10.26.
AMD15433F	Ann	M	Devon	3	200	9952	1991.07.16.
ARD36773F	Anabela	R	Domingues	8	100	0877	1993.01.27.
PHF38899M	Peter	H	Franken	10	75	0877	1992.05.17.
PXH22250M	Paul	X	Henriot	5	159	0877	1993.08.19.
CFH28514M	Carlos	F	Hernadez	5	211	9999	1989.04.21.
PDI47470M	Palle	D	Ibsen	7	195	0736	1993.05.09.
KJJ92907F	Karla	J	Jablonski	9	170	9999	1994.03.11.
KFJ64308F	Karin	F	Josephs	14	100	0736	1992.10.17.
MGK44605M	Matti	G	Karttunen	6	220	0736	1994.05.01.
POK93028M	Pirkko	O	Koskitalo	10	80	9999	1993.11.29.
JYL26161F	Janine	Y	Labrune	5	172	9901	1991.05.26.

10.3. ábra A pubs adatbázis emloyee tábla adatai. Fent a lekérdezés eszközsor

Ha egy tábla adatait kívánjuk megnézni, a tábla nevén duplán kattintva megkapjuk őket.

10.1. Az adatbázis létrehozása

A Server Explorerben a csatolt SQL szerverek gyorsmenüjében megtaláljuk a New Database menüpontot. Az adatbázis nevét megadva, és kiválasztva, hogy az operációs rendszer vagy az SQL szerver azonosítását kívánjuk-e használni, létrehozzuk az adatbázist.

Ha az SQL szerver Windows alapú intraneten érhető el, akkor ha a szerver az IIS-el (Internet Information Server) azonos gépen fut, és azonos domain névhez tartozó felhasználók használják, akkor használhatjuk a felhasználók Windows azonosítóit. Ez a megoldás biztonságosabb, de nem használható nyilvános Web oldalakon, amik pl. Unix alól is elérhetőek.

Az adatbázis tábláihoz a gyorsmenü New Table parancsával adhatunk táblákat, megadva a mezők nevét és tulajdonságait a megjelenő ablak adatait kitöltve. Mentéskor nevezzük el a táblát! A tábla az adatok megjelenítése funkcióval feltölthető, módosítható.

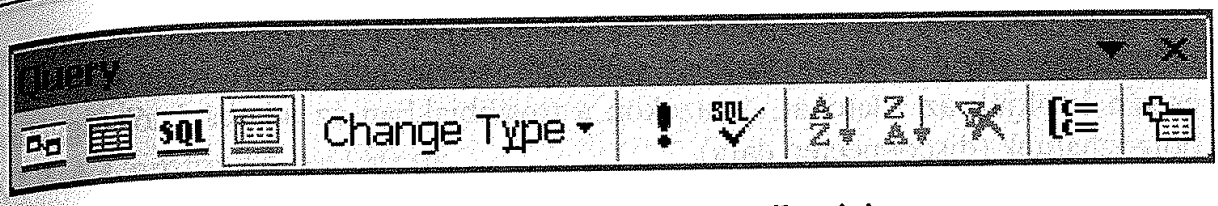
Access adatbázist az Access segítségével hozzuk létre!

Az Access adatbázist a Server Explorer Data Connections segítségével érhetjük el. A csatolt adatbázis adatai az SQL Serverhez hasonlóan szerkeszthetők.

Tehát az adatbázis elérése a Server Explorer más kategóriájából történik, de az adatok megjelenítése és szerkesztése mindkét esetben azonos.

A lekérdezések szerkesztését az SQL Serverben vagy az Accessben megszokott Query eszközsor, és a hozzá kapcsolódó különböző nézetek segítik. A nézetek ki-be kapcsolhatók, és az eszközsor bal oldalán a következő sorrendben érhetők el:

- Diagram nézet: a lekérdezésben szereplő táblák közti kapcsolatot mutatja áttekinthető grafikus ábrán.
- Tervező nézet (Grid Pane): A lekérdezés tervezését támogatja.
- SQL nézet: Az SQL nyelv jó ismerőinek teszi lehetővé a lekérdezés szöveges begépelését, vagy a tervező nézetben megszerkesztett lekérdezés átolvasását.
- Adatlap nézet (Results Pane): A lekérdezés eredményét mutatja. A kapott adatokból következtethetünk helyesen fogalmaztuk-e meg a lekérdezést.



10.4. ábra Lekérdezés eszközsor ikonjai

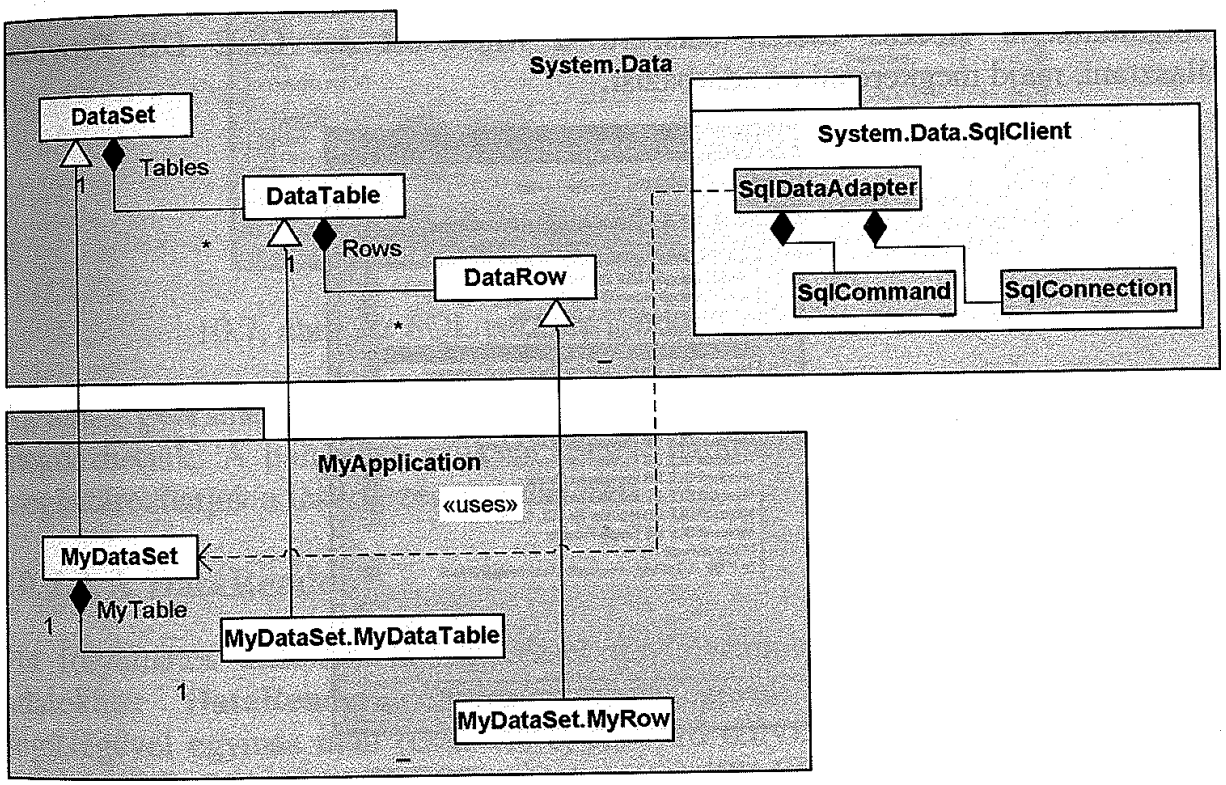
A nézetek a 10.7. ábra lekérdezés szerkesztőjében láthatóak.

A Change Type eszközgomb segítségével adhatjuk meg, hogy a lekérdezéssel az adatokat valamilyen csoportosításban vagy szűrve, csak olvasni szeretnénk (Select), vagy módosítani is (Insert, Update, Delete).

A további eszközgombok a lekérdezés futtatását, ellenőrzését, szerkesztését szolgálják.

10.2. Az adatbázis-elérést biztosító osztályok

Az eddigiekben az alkalmazáson kívülről, csak a fejlesztőeszköz lehetőségeit felhasználva szerkesztettük adatbázisunkat. Most megvizsgáljuk, hogy a .NET osztálykönyvtár mely osztályai támogatják az adatok elérését alkalmazásainkból.



10.5. ábra Az adatok kezelését biztosító osztályok UML diagramja

Az ADO.NET osztályokat úgy tervezték, hogy többrétegű környezetben támogassák az adatokhoz történő hozzáférést úgy is, ha a távoli adatbázis folyamatosan nem

érhető el. Alkalmazásunk fejlesztésekor eldönthetjük, hogy folyamatosan, vagy csak a szükséges ideig kapcsolódunk az adatbázis szerverhez. Rövid idejű kapcsolódás esetén letöltjük az adatokat, és azokon a továbbiakban az adatbázistól függetlenül dolgozhatunk (disconnected data).

Az ADO.NET osztályok a **System.Data** névtérben és alnévtereiben találhatók.

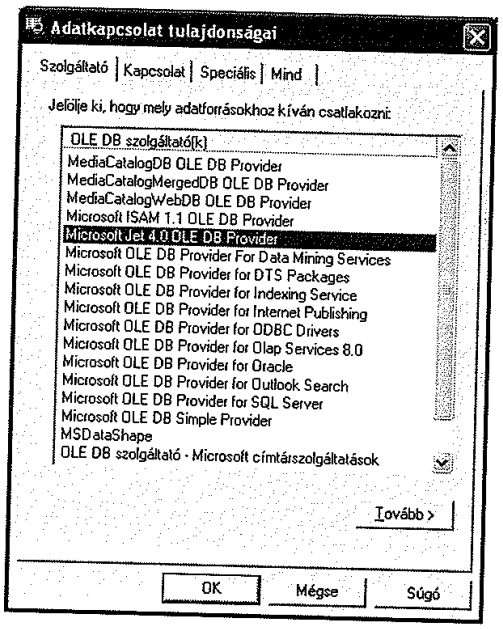
10.2.1. Az adatkapcsolat

Az adatbázis elérését a Connection osztályok biztosítják. SQL adatbázis esetén az **SqlConnection** osztály, ODBC (Open Database Connectivity) kapcsolat esetén **OdbcConnection**, Oracle adatbázis esetén **OracleConnection**, míg OLE DB kapcsolat esetén **OleDbConnection** osztály. Mindegyik osztály legfontosabb tulajdonsága a **ConnectionString**, mely meghatározza az adatbázis helyét.

Ha egy táblát vagy tárolt lekérdezést a Form felületére húzunk, a fejlesztőeszköz automatikusan létrehozza hozzá a Connection objektumot. Vagy a Toolbox Data csoportjában megtaláljuk az objektumokat, kiválasztva a megfelelőt, a Properties ablakban beállíthatjuk a ConnectionString tulajdonságát.

A OleDbConnection objektumának ConnectionString tulajdonságon a nyilat választva, New Connection, és megjelenik az adatkapcsolat tulajdonságai ablak.

Szolgáltatóként Access esetén célszerű a Microsoft Jet OLE DB Providert választani, majd megkeresni az adatbázist.



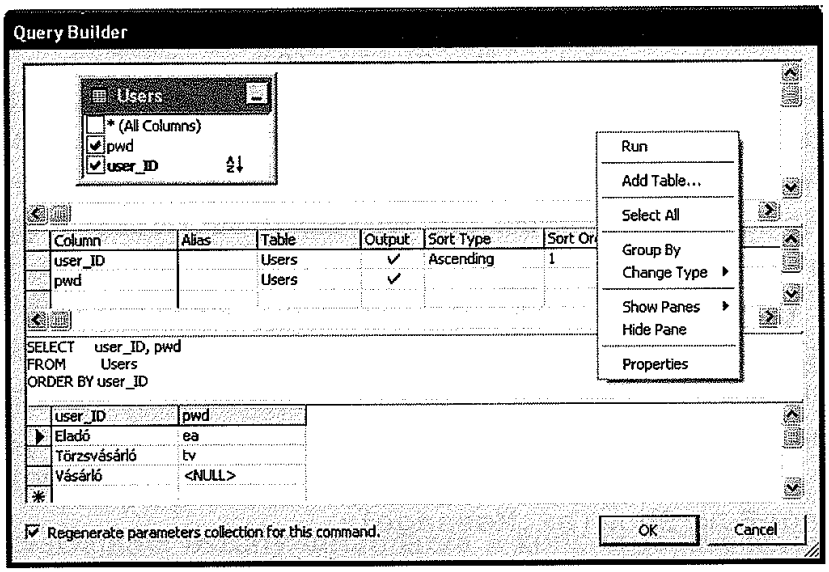
10.6. ábra Az adatkapcsolat tulajdonságait beállító ablak

Mint írható tulajdonság, a ConnectionString futásidőben a kódból is beállítható.

A gyakorlat 10.3. szakaszában látunk erre példát.

10.2.2. A Command osztályok

A Command objektumok az adatbázis közvetlen elérésére használhatóak. Az eszközsor Command objektumból is négyfélét kínál: OleDbCommand, SqlCommand, OracleCommand, OdbcCommand. Segítségükkel SQL parancsokat adhatunk. A Connection tulajdonságukba kiválaszthatjuk a már létező kapcsolatok valamelyikét, vagy létrehozhatunk új kapcsolatot. Ha már meghatároztuk a kapcsolatot, a CommandText tulajdonságban megadhatjuk a lekérdezést. A lekérdezés létrehozásához az SQL Serverből vagy az Accessből már ismert lekérdezésszerkesztő (Query Builder) segíti munkánkat.



10.7. ábra Az SQL parancs szerkesztése a gyorsmenüvel

Az ablak gyorsmenüjéből a Run menüpontot választva a lekérdezés eredményét is megnézhetjük. A parancsot szükség esetén paraméterezhetjük, így futásidőben is meghatározhatjuk a lekérdezés paramétereit.

10.2.3. A DataReader

Ha folyamatos adatbázis-hozzáféréssel rendelkezünk, akkor a leggyorsabban az adatokhoz **DataReader** objektum segítségével juthatunk. A DataReader segítségével egymás után (csak előre, forward only) kiolvashatjuk az adatokat az adatbázisból. Bármely SELECT parancs vagy SELECT-et tartalmazó tárolt eljárás végrehajtható így. A DataReader objektumhoz a Command objektum **ExecuteReader** tagfüggvényének visszatérési értékeként juthatunk hozzá. Ne feledjük azonban, amíg a DataReadert be nem zárjuk, addig él a kapcsolat az

alkalmazás és az adatbázis között! Így a DataReadert olyan gyorsan zárjuk be a **Close** metódussal, amilyen gyorsan csak lehetséges.

A következő kódrészlet végigolvassa a gyakorlaton szereplő pw Access adatbázist, és hiányzó jelszó esetén üres sztringet tesz a password lokális változóba.

```
using System.Data.OleDb;
…
string user;
string password;
try
{
 oleDbConnection1.Open();
 OleDbDataReader reader = oleDbCommand1.ExecuteReader();
 while (reader.Read())
 {
       user = reader.GetString(0);
       if(reader.IsDBNull(1))  // Üres jelszó.
       {
             password = "";
       }
       else
             password = reader.GetString(1);
 }
 reader.Close();
}
catch (InvalidOperationException ex) // Már nyitva!
{
 MessageBox.Show(ex.Message);
}
catch (OleDbException ex)       // Nem sikerült megnyitni!
{
 MessageBox.Show(ex.Message);
}

finally
{
 // A kapcsolatot bezárjuk akár volt kivétel akár nem.
 oleDbConnection1.Close();
}
```

10.2.4. A DataSet

A **DataSet** osztály a lekérdezés eredményeként visszaadott adatokat tárolja a memóriában. Amíg a DataReader egyszerre egy sort ad vissza a memóriában, addig a DataSet a lekérdezés teljes eredményét szolgáltatja. A DataSet szerkezete az adatbáziséhoz hasonló. Az adatokat táblákban (**DataTable**) tárolja, a táblák sorokra

(**DataRow**) (rekordokra) és oszlopokra (**DataColumn**) (mezőkre) oszthatók. A kapcsolt táblák közti kapcsolatokat **DataRelation** objektumokban tárolja.

A táblák gyűjteménye a DataSet.**Tables** tulajdonságán keresztül érhető el. A konkrét tábla sorait a **Rows** gyűjtemény, míg oszlopait a **Columns** gyűjtemény tartalmazza. A sorok és oszlopok is indexelhetők, az oszlopok a nevükkel is azonosíthatók.

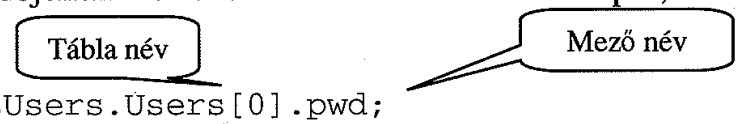
Az oszlopoknál lekérdezhetjük a nevet (ColumnName), a típusát (DataType), az alapértelmezett értéket (DefaultValue), hogy engedélyezett-e az üres érték (AllowDBNull), egyedi-e (Unique), csak olvasható-e (ReadOnly), vagyis amit a tervezéskor megadhatunk.

Egy konkrét sor megőrzi a módosítás előtti állapotát is, amíg **AcceptChanges**() függvényt nem hívunk rá. A **RowState** tulajdonságban lekérdezhetjük, lett-e módosítva a sor, és a **RejectChanges**() hívásával visszaállíthatjuk az utolsó AcceptChanges() híváskor megadott értékét.

Mivel az alkalmazásunk által felhasznált DataSet osztály adatbázis-specifikus, ezért az alkalmazásban egy speciális utódosztályt kell létrehoznunk, mely a konkrét adatbázis konkrét tábláit, konkrét oszlopait tartalmazza. A Visual Studio automatikusan egy DataSet1 nevű osztályt tud létrehozni, melyet célszerű egyedi beszédes névvel ellátnunk (UsersDataSet). A DataSet1 osztály objektumát a fejlesztőeszköz dataSet11-nek nevezi. Bár az osztály nagy kezdőbetűje megkülönböztetné a kis kezdőbetűs objektumtól C# környezetben, de a komponensfejlesztésnél említettük, hogy a case sensitivitást nem célszerű általánosan kihasználni. Ez magyarázza az objektum dupla indexelését. Mi adjunk beszédes azonosítót objektumainknak is! (dsUsers)

A létrehozott UsersDataSet osztály tartalmazza a konkrét táblák leírását tartalmazó DataTable utódosztályát és a konkrét sorok leírását tartalmazó DataRow utódosztályát.

Ha egy sor konkrét mezőjének értékét szeretnénk visszakapni, azt a következőképpen tehetjük:




```
textBox1.Text = dsUsers.Users[0].pwd;
```

A DataSet objektum tehát egy helyi másolata az adatbázis, vagy más adatforrás adatainak. Ezért használata lehetővé teszi, hogy alkalmazásunk ne kapcsolódjon folyamatosan az adatbázishoz, hisz a helyi adatokon a kapcsolat fenntartása nélkül is lehet dolgozni. Ez nagymértékben növeli az adathozzáférés skálázhatóságát (scalability), vagyis azt, hogy adatbázisunkhoz minél több felhasználó férhessen hozzá. Hisz a felhasználók csak az adatok letöltése és frissítése idején foglalják az adatbázis erőforrásait.

Megjegyzés:

A DataSet osztályok lehetnek típusosak és típus nélküliek. Mi itt a típusos DataSettel foglalkoztunk, melynek már fordításidőben ismerjük a tábláit és mezőit. Így kényelmesebb dolgozni vele, de jó, ha tudjuk, hogy a .NET támogatja a futásidőben érkező, az adatszerkezet információkat is futásidőben megadó adatok feldolgozását is.[1]

10.2.5. A DataAdapter osztály

A **DataAdapter** objektum köti össze az adatbázist a DataSet objektummal. Biztosítja az adatok kiolvasását az adatbázisból és mentését az adatbázisba. DataAdapterből is az adatkapcsolat típusának megfelelően különböző osztályok állnak rendelkezésünkre: OleDbDataAdapter, SqlDataAdapter, OdbcDataAdapter, OracleDataAdapter. A DataAdapter négy parancs tulajdonsággal rendelkezik: SelectCommand, DeleteCommand, InsertCommand és UpdateCommand. Mindegyik parancs esetén megadjuk a Connection objektumot és a CommandText tulajdonságban a megfelelő SQL lekérdezés szövegét.

A **DataSet** objektumot a legegyszerűbben a formra húzott DataAdapter objektumból gyorsmenü segítségével hozhatjuk létre. A Generate DataSet menüpontot választva a New melletti szövegmezőben megadhatjuk a DataSet osztály nevét, vagy az Existing választógombot kijelölve a listából kiválaszthatjuk a módosítani kívánt DataSet osztályt. Kiválasztjuk a tartalmazott táblák nevét. A Visual Studio a DataSet osztályon kívül az osztály egy objektumát is létrehozza. Az objektum nevét a Properties ablak Name tulajdonságában írhatjuk át.

Vizsgáljuk meg a Class View ablakban a létrehozott DataSet utódosztályát, a DataTable utódosztályát és a DataRow utódosztályát!

A Solution View ablakban megjelent a UsersDataSet szerkezetét leíró UsersDataSet.xsd, melyet táblázatosan vagy XML nézetben mutat a fejlesztőeszköz. Az XML leírás az adatok továbbítását platform függetlenné teszi, vagyis adataink bármely böngészőben bemutathatók lesznek. A Solution View ablakban lehet törölni a feleslegessé vált DataSeteket.

[1] MSDN, Index, untyped datasets.

```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="UsersDataSet"
targetNamespace="http://www.tempuri.org/UsersDataSet.xsd"
xmlns:mstns= "http://www.tempuri.org/UsersDataSet.xsd"
xmlns= "http://www.tempuri.org/UsersDataSet.xsd"
xmlns:xs= "http://www.w3.org/2001/XMLSchema"
xmlns:msdata= "urn:schemas-microsoft-com:xml-msdata"
attributeFormDefault = "qualified"
elementFormDefault="qualified">
 <xs:element name="UsersDataSet" msdata:IsDataSet="true"
                                 msdata:Locale="hu-HU">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Users">
        <xs:complexType>
          <xs:sequence>
           <xs:element name="user_ID"type="xs:string"/>
            <xs:element name="pwd" type="xs:string"
                                 minOccurs="0" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
  <xs:unique name="Constraint1" msdata:PrimaryKey="true">
    <xs:selector xpath=".//mstns:Users" />
    <xs:field xpath="mstns:user_ID" />
  </xs:unique>
 </xs:element>
</xs:schema>
```

A DataSet objektum feltöltése a DataAdapter.**Fill** metódusával történik, melynek paraméterként kell megadnunk a DataSet objektumot. Az adatok mentése a DataAdapter objektumra meghívott **Update** függvényhívással valósul meg, a mentésre váró DataSet, tábla vagy sorok tömbje paraméter megadásával. A Fill metódus a SelectCommand, az Update függvény pedig a DataSet sorain bekövetkező változástól függően az InsertCommand, DeleteCommand vagy UpdateCommand parancsot használja az adatok módosításához. Az adatbázishoz csak e két metódus futása alatt kapcsolódik alkalmazásunk. Ezt jelöli a DataAdapter ikonja, mely egy szürke mezőben mutatja a kapcsolatot az adatbázissal, így hangsúlyozva annak ideiglenességét.

Ha az adatforrás adatai változnak, a változásoknak a DataSet-ben is meg kell jelenniük. Erre szolgál a DataSet osztály **AcceptChanges** metódusa. A DataAdapter Fill és Update metódusai meghívják az AcceptChaneges-t csak ha közvetlenül Command paranccsal módosítjuk az adatbázist, akkor kell meghívnunk a DataSet módosításához.

10.3. Az adatok megjelenítése

A legegyszerűbb mód egy tábla vagy egy lekérdezés eredményének megjelenítésére a Data Form Wizard.

File menü

Add New Item

Templates: Data Form Wizard

Open. Next

DataSet létrehozása, vagy kiválasztása

Crate a new DataSet named: MyDataSet. **Next**

Kapcsolat kiválasztása, vagy létrehozása. **Next**

Táblák, nézetek kiválasztása. Next

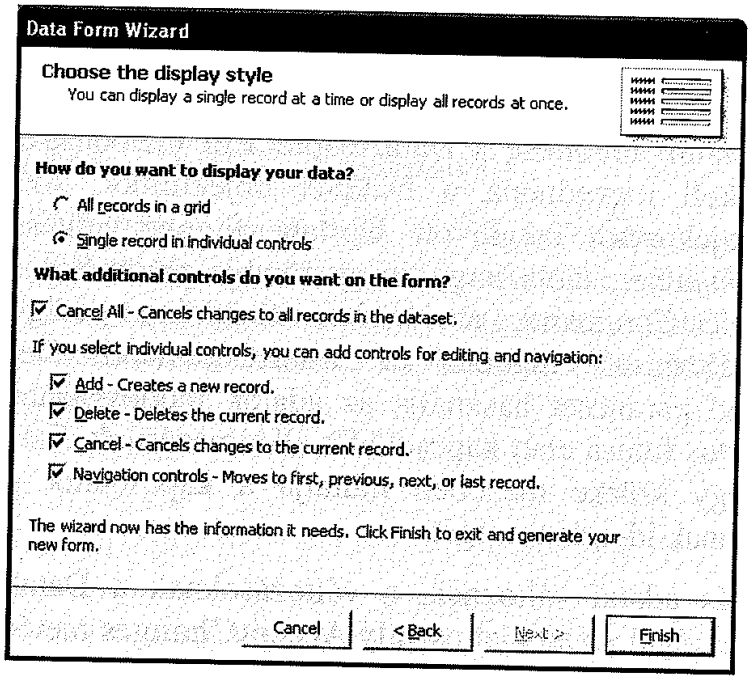
Oszlopok meghatározása. Next

Táblázatos, vagy egy oldalon egy sor önálló vezérlőkben

Kiválaszthatjuk az engedélyezett műveleteket:
Add, Delete, Cancel, navigációs vezérlők

Finish

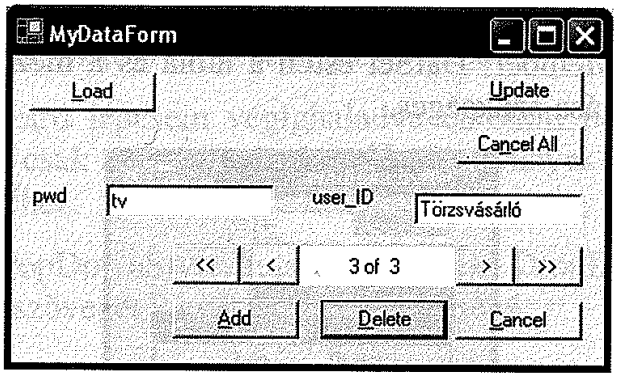
Eldönthetjük, kérünk-e jelszót a Form megjelenítéséhez.



10.8. ábra A Data Form Wizard adatmegjelenítési beállításai

A létrehozott formot szerkeszthetjük, és a kódból hívjuk meg!

```
MyDataForm myDataForm = new MyDataForm();
myDataForm.Show();
```



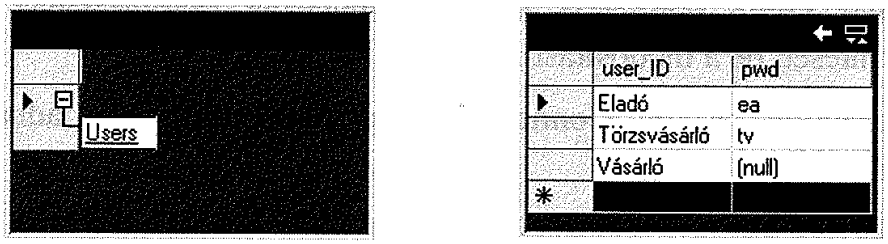
10.9. ábra A Data Form Wizard által létrehozott működő form

10.3.1. A vezérlők

Az előzőekben generált formot is továbbfejleszthetjük, módosíthatjuk, a Visual Studio által generált kódot felhasználhatjuk saját fejlesztéseinkhez.

10.3.1.1. A DataGrid

A DataGrid vezérlő kényelmessé teszi az adatbázis adatainak megjelenítését táblázatos formában. A vezérlő DataSource tulajdonságához DataSet objektumot, vagy közvetlenül táblát rendelhetünk hozzá. DataSet esetén felhasználáskor ki kell választani a megjeleníteni kívánt táblát, viszont több tábla megjelenítésére is módunk van. Ha a táblát adjuk meg közvetlenül, akkor az adataival együtt látszik futáskor.



10.10. ábra A DataGrid két állapota, ha DataSet objektum az adatforrás

Megjegyzés:

Több táblás adatbázis esetén a csatolt táblák hasonló módon, a + ikont kinyitva mutatják meg adataikat.

10.3.1.2. A ComboBox

Ha ComboBox segítségével szeretnénk egy mező különböző értékeit felsorolni, akkor a ComboBox tulajdonságlapján két tulajdonságot kell beállítanunk.

- DataSource: Egy DataSet, vagy egy tábla választható ki
- DisplayMember: DatSet estén a tábla és a mező, tábla esetén a mező kiválasztása szükséges.

10.11. ábra A kiválasztott user_ID sora lesz az aktuális a táblát mutató táblázatban is

A ComboBox még egy szolgáltatást kínál, ha DropDown Style-ját ComboBoxStyle.DropdownList típusúra állítjuk, akkor a begépelt karakterekre automatikusan rááll. Emiatt nem kell végiggépelni hosszú neveket. Ilyen beállítás esetén a ComboBoxba nem gépelhető más érték, csak azokat az elemeket tudjuk kiválasztani, melyek az adatbázisban szerepelnek. Ez a szolgáltatás különösen előnyös nagyméretű adatbázisok esetén, ahol nem a felhasználónak kell görgetéssel megkeresni a kívánt sort.

10.4. Teszt

1. A Visual Sudio támogatja létező adatbázisok szerkezetének és adatainak megtekintését, de a módosítást nem teszi lehetővé.
2. A Connection osztály ConnectionString tulajdonságában adhatjuk meg az adatbázis elérését.
3. A Microsoft SQL Server adatbázisokon kívül a többi elérését csak ODBC-n keresztül teszi lehetővé az osztálykönyvtár.
4. A Command objektumok közvetlenül az adatbázison dolgoznak, ezért működésükhöz közvetlen adatelérés szükséges.
5. A DataSet objektum a memóriában tárolja a DataAdapter által visszaadott adatokat.

6. Ha egy adat értékét esetleg vissza kívánjuk állítani a DataSetben, mentsük el egy segédváltozóban, mielőtt megváltoztatjuk!

7. Típusos DataSet a leírását XML Schemában tárolja, mely egy xsd kiterjesztésű XML fájl.

8. A DataAdapter objektum Fill metódusával fel kell töltenünk a DataGrid vezérlőt, hogy az adatokat ki tudja tenni az ablakba.

9. A Data Form Wizard lépésein végighaladva elkészíti nekünk a kívánt Form felületét, nekünk csak az eseménykezelő függvényeket kell megírnunk a vezérlők működéséhez.

10. A ComboBox DropDownStyle tulajdonságának DropDown-ra állításával a vezérlő a begépelt szöveget kiegészíti a lista elemévé.

Javítókulcs:

H, I, H, I, I, H, I, H, H, I.

11. Web alkalmazás fejlesztése

11.1. Előfeltételek

Ahhoz, hogy Web alkalmazást (ASP.NET Web Application), hálózati szolgáltatót (ASP.NET Web Service) és hálózati alkalmazások vezérlőit (Web Control Library) készíthessük el a Visual Studioban, operációs rendszerünk telepítése után telepítetnünk kell az Internet Information Servicest (**IIS**).

11.1. ábra Az IIS ikonja a Vezérlőpult Felügyeleti eszközei között

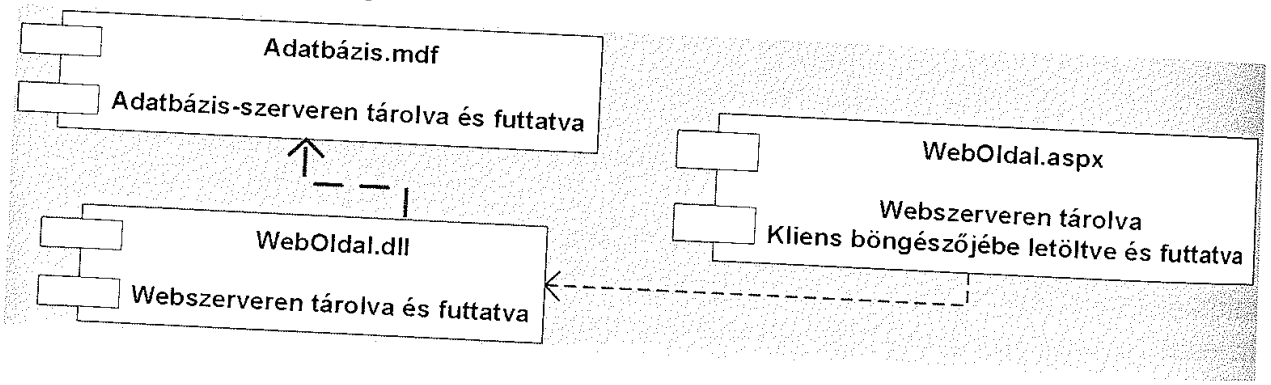
Bizonyos telepítő csomagok esetén a Visual Studio telepítése is csak az IIS működésekor lehetséges. Az alkalmazások böngészőben futtatásához pedig a helyi hálózat proxykiszolgálójának beállítása szükséges. Az előkészítésről részletes információk a fejezethez tartozó gyakorlat elején olvashatók.

11.2. A Web Form

A web form olyan dinamikus web oldal, amit a felhasználó a böngészőben lát, és amelyet a szerver állít elő. A web form HTML, kód, és olyan vezérlők

kombinációja, melyek eseményeit a Web szerveren (IIS) **futó kód kezeli.** A web form a szerkesztőben megadott felhasználói felületből (Design) HTML kódot generál, és ezt küldi el a böngészőnek. A felülethez tartozó háttérkód a szerveren fut. Tehát a kliens oldalon a felhasználói felület látszik, míg a kód a szerver oldalon fut. Ez merőben más megoldás, mint a hagyományos web oldalak, melyeknél a teljes kód letöltése és végrehajtása a kliens böngészőjében történt. Web form esetén csak a felület vezérlőit kell elküldenünk a böngészőnek, a kód végrehajtása a szerveren marad. A felhasználói felület és a kód szétválasztása növeli a támogatható böngészők számát, növeli a funkcionalitást és a biztonságot. **Mivel a web form a kliens oldali scriptekre csak a kliens szolgáltatásai esetén épít – a szerver oldali kód ilyen szolgáltatást nem biztosító kliens esetén is lefut –, nem függ a kliens böngészőjétől és operációs rendszerétől.**

Természetesen, mint az életben és a programozásban mindennek, így ennek a függetlenségnek is ára van. Ha a kód a szerveren hajtódik végre, akkor használatához állandó internetkapcsolatra van szükség, melyet minden esemény bekövetkeztekor igénybe is veszünk. Miközben az oldal letöltésekor kevesebb adat szükséges – hisz nincs szükség a scriptekre –, addig az eseményekhez kapcsolt kód végrehajtásához a további adatátvitel elengedhetetlen. Bár kisméretű adatok esetén ez a felhasználó számára lehet, hogy nem érzékelhető. Az állandó internetkapcsolat költséges, bár a költségek csökkenek. Sok esetben lassú az adatátvitel, ez is javul. Ezen kívül pedig folyamatosan terheli a szervert, ezért a hatékonyság érdekében általában vegyes megoldásokat szoktak alkalmazni. Az egyszerűbb, nem erőforrás-igényes tevékenységeket – pl. hogy a bevitt érték adott határok közt van-e – rábízhatjuk a kliensoldali scriptre, ha támogatja azt a böngésző. Így nem terheljük szerverünket, és még gyorsabb is helyben a végrehajtás. Ha böngésző nem támogatja a kívánt script végrehajtását, elvégezheti azt a szerver. Tehát a web formok új, eddig akadályt jelentő tevékenységeket kínálnak számunkra, de élni velük akkor érdemes, ha valóban hatékonyabbá és akadálymentesebbé teszik alkalmazásunk végrehajtását!



11.2. ábra Egy web alkalmazás főbb komponensei

A Web Formokat gyakran ASP.NET (Active Server Page .NET) oldalaknak vagy ASPX oldalaknak is nevezik az aspx kiterjesztésük miatt. **ASPX oldalak** és ASP oldalak (.asp kiterjesztés) futhatnak azonos szerveren. A Web Formokhoz többnyire

két külön fájl tartozik, az aspx fájl tartalmazza a felhasználói felületet (UI User Interface), míg az aspx.cs fájl a C# kódot, melyet az oldal mögötti kódnak is hívnak (**code-behind page**).

A Web Form html nézetében láthatjuk:

```
<form id="Form1" method="post" runat="server">
```

A Form1 **runat** attribútuma server értéket vett fel. Ez mutatja, hogy az eseménykezelés kódja a szerver oldalon fut. Egy HTML oldalon akárhány form is megjelenhet, de csak pontosan egy server-side form lehet egy .aspx oldalon! A runat="server" attribútum beállítás következtében a form a vezérlés adatait annak a szervernek küldi el, melyen a kód fut. Ha a runat attribútum nem server, akkor hagyományos HTML oldallal dolgozunk.

A HTML első sora mutatja a forráskód nyelvét és a forrásfájl nevét:

```
%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
```

A forráskódból megtudhatjuk az ősosztály nevét, és az eddig tanultak alapján értelmezhetjük, írhatjuk a forráskódot.

```
public class WebForm1 : System.Web.UI.Page
```

11.2.1. A Web Form szerkesztése

A Web oldal elrendezése kétféle lehet:

- Táblázatos elrendezés – **GridLayout**: A Form a Windows Formhoz hasonlóan kényelmesen szerkeszthető. Futtatáskor megőrzi a vezérlők egymáshoz és a formhoz viszonyított helyzetét. Hátránya, hogy a különböző felbontású és ablakméretű gépeken futtatva a vezérlők egy része az ablak szélén kívül kerülhet, így kényelmetlenné válhat a használat. Ki nem járt már olyan Web oldalon, ahol a szöveg egy része a keret mögé csúszott, és soronként oda-vissza kellett görgetni az elolvasáshoz?
- Sorfolytonos elrendezés – **FlowLayout**: A vezérlők egymás utáni sorrendje meghatározott. Az ablak határán nem nyúlnak túl, a következő sorba kerül az a vezérlő, amelyik nem fér el az előző sor végén. Előnye, hogy tetszőleges felbontású és ablakméretű böngészőben megmarad a vezérlők sorrendje és láthatósága. Hátránya, hogy nehéz elegáns elrendezést biztosítani az ablaknak. Sorfolytonos elrendezés esetén táblázat segítségével lehet pozícionálni.

Az ablak elrendezését a **pageLayout** tulajdonságban állíthatjuk be. A Visual Studio GridLayout formot generál, melyet a Properties ablak pageLayout tulajdonságában módosíthatunk.

A **Toolbox Web Forms** vezérlői közt nagyon sok eddig megismert vezérlővel találkozhatunk. Ezek azonban, bár funkcióik jelentős részében azonosak, más osztályok. Míg a **Windows Forms** vezérlőink osztályai a `System.Windows.Forms` névtérben, addig a Web Forms vezérlőink mögött a `System.Web.UI.WebControls` névtér osztályai állnak. Web alkalmazások esetén a Toolbox még tartalmaz egy **HTML** csoportot is, melynek vezérlőihez alapértelmezésben a felhasználó böngészőjében futtatott kód tartozik, vagyis a kliens oldalon scriptek segítségével kezelhetjük eseményeiket.

Megjegyzés:

A HTML vezérlők gyorsmenüjében a **Run As Sever Control** menüpontot választva a `System.Web.UI.HtmlControls` névtér osztályait kapjuk meg a háttérben, melyek szintén lehetővé teszik a szerver oldali eseménykezelést.

A következő HTML részlet három nyomógomb leírását mutatja, melyek azonos módon jelennek meg a böngészőben.

HTML button, csak kliens oldali eseménykezelés:

```
<INPUT type="button" value="HTML Button">
```

HTML server control button:

```
<INPUT type="button" value="HTML Server Button"
id="button1" runat="server">
```

Web server control button ASP.NET HTML:

```
<asp:button id="Button" runat="server" Text=
"Web Button">
```

11.3. A Server Control

Az ASP.NET szerver vezérlők esetén elválik a felhasználói felület és a futtatott kód. Az UI a böngésző ablakában jelenik meg, míg a kód a szerveren fut. A **server control** a WebForm.aspx Design nézetében grafikusan szerkeszthető. A felhasználói felület leírása a HTML kódban olvasható. A szerver vezérlő runat attribútuma is – a web formhoz hasonlóan – „server" értékű. A felhasználói felületen azonosítója (id) van, ami a C# forráskódban az objektum nevének felel meg.

HTML kontrol eseménykezelését kliens oldali script végrehajtásával oldhatjuk meg, míg a szerver vezérlő eseménykezelése a hagyományos Windows vezérlőkéhez hasonlóan történik.

Pl. egy web form Button vezérlőn duplán kattintva az aspx.cs forrásfájlban automatikusan létrejön a gomb alapértelmezett eseménykezelője – a Button_Click metódus. A különbség csak az, ami a formnál is, hogy a gomb a böngészőben jelenik meg, amely egy távoli gépen is futhat, s az eseménykezelő metódus a szerveren hajtódik végre. A látható eredmény pedig újra a távoli gép böngészőjében lesz megjelenítve.

11.3.1. Az állapot megőrzése

A Web oldalak egyik fő problémája, hogyan őrizzék meg állapotukat – beállításaikat és a felhasználói bevitelt. A HTML oldalakat a szerverről töltjük le, és a következő oldal letöltésekor megszűnik a form objektum ami létrehozta az oldalt. Ha visszalépünk rájuk, újra létrejön a form objektum alapértelmezett értékekkel és ez jelenik meg a böngészőben. A felhasználó által kitöltött adatok és beállítások elvesznek. Azt mondjuk erre, hogy a Web oldalak állapot nélküliek (**stateless**).

Az ASP.NET Web Form úgy kezeli ezt a problémát, hogy tárolja a vezérlő állapotát. Ehhez egy rejtett vezérlőt használ amit **_VIEWSTATE**-nek nevezünk és ha igazra állítjuk a vezérlő vagy az oldal **EnableViewState** attribútumát, akkor megjegyzi a vezérlő vagy az oldal adatait. Amikor az oldalt újra letöltjük a szerverről az állapot a vezérlővel együtt letöltődik.

Alapértelmezésben a Web Form megőrzi a vezérlők állapotát. Több vezérlős oldal esetén az oldal _VIEWSTATE tulajdonságának mérete nagyon nagy lehet. Ezért ilyenkor tiltsuk le az oldal @Page direktívájában az oldal állapotmegőrzését, és vezérlőnként állítsuk be azt!

```
<%@ Page EnableViewState = "False" %>

<asp:ListBox id="Users" EnableViewState = "true"
                        runat = "server"> </asp:ListBox>
```

Az ASP.NET oldalnak van még egy hatékonyságot növelő eszköze, ez a **SmartNavigation**. Ha ezt a direktívát igazra állítjuk, frissítéskor csak a módosított adatokat tölti le az oldal.

```
<%@ Page SmartNavigation = "True" %>
```

11.3.2. A bevitt értékek ellenőrzése

Lehetőségünk van még egy hasznos vezérlőtípus alkalmazására, ez a **Validation control**. A Validation control feladata az adatbevitel ellenőrzése. Szerver oldali és kliens oldali ellenőrzést egyaránt megvalósíthatunk velük.

Az ellenőrző vezérlőket a Toolboxból húzhatjuk az ellenőrizni kívánt vezérlő mellé. A **ControlToValidate** tulajdonsághoz kiválasztott azonosító (id) határozza meg az ellenőrzött vezérlőt. Egy vezérlőhöz több ellenőrző vezérlőt is csatolhatunk, egyik például a kötelező kitöltés, míg a másik a bevitt érték ellenőrzése, megfelel-e az előírásoknak.

Az **EnableClientScript** tulajdonság True értékre állításával (alapértelmezés) engedélyezhetjük, hogy az ellenőrzés a kliens oldalon történjen meg először. Így elérhetjük, hogy bizonyos hibák esetén az oldal nem küld adatokat a szervernek, a kliens oldalon végrehajtott kód küld hibaüzenetet. A kliens oldali ellenőrzést a DHTML-t támogató böngészők (pl. Internet Explorer 4.0 vagy későbbi verziók) tudják biztosítani, más böngészők esetén az ellenőrzés a szerver oldalon történik meg.

Néhány ellenőrző vezérlő: **RequiredInputValidator** a kötelező kitöltést ellenőrzi, a **CompareValidator**ral összehasonlíthatunk, míg a **RangeValidator** ellenőrzi, hogy a vezérlő tartalma a megadott értékhatárok közt van-e. Ne feledjük Type tulajdonságának beállítását, mert annak hiánya hibás működést eredményezhet! A 1999 mint sztring kisebb az 5-nél, hisz első karaktere kisebb, de mint Integer már nem. A **RegularExpressionValidator** segítségével speciális mezőket ellenőrizhetünk, pl. e-mail címet, vagy URL-t.

A **CustomValidator** a fejlesztő által definiált ellenőrzést biztosít. A szerveroldali ellenőrző függvényt a **ServerValidate** esemény kezelésével tudjuk létrehozni. A függvény **ServerValidateEventArgs** paraméterének **Value**tulajdonságával érhetjük el az ellenőrizni kívánt sztring értékét, és **IsValid** tulajdonságának beállításával adhatjuk meg a sztring érvényességét.

Ha kliens oldali ellenőrzést akarunk, a szerver oldali ellenőrző függvény elkészülte után készítsük el az aspx oldal ellenőrző függvényét, melynek formátuma a következő:

VBScript:

```
Sub ValidationFunctionName(source, arguments)
```

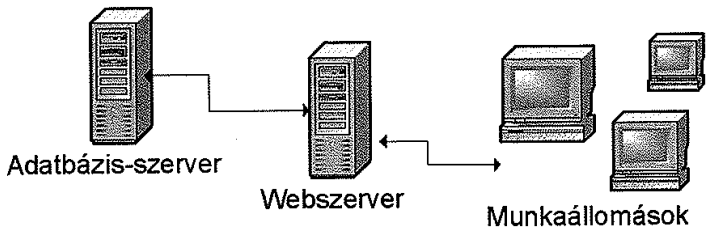
JScript:

```
function ValidationFunctionName(source, arguments)
```

Ezután a **ClientValidationFunction** tulajdonságba írjuk be az ellenőrző függvény nevét!

Minden ellenőrző objektum esetén adjuk meg az **ErrorMessage** szövegét, mely érvénytelen bevitel esetén a felhasználó számára nyújt tájékoztatást a hiba okáról!

11.4. Az adatbázis adatainak megjelenítése



11.3. ábra Számítógépek kommunikációja web alkalmazások futtatása során

Az adatbázis adataihoz a Windows alkalmazásokhoz hasonló módon az adatbázis hozzáférést támogató osztályok segítségével juthatunk el Web alkalmazások esetén is. A **Connection** osztályok biztosítják az egyszerű hozzáférést, a **DataAdapter** az adatátvitelt az adatbázis és az általunk létrehozott **DataSet** között. A DataSet biztosítja a **csatlakozásfüggetlen** (**disconnected**) adatkezelést, ahol az adapter csak a letöltés vagy a módosítás idejére áll kapcsolatban az adatbázissal. Az adatok írását-olvasását a **Command** osztályok végzik.

Mielőtt egy DataSet objektumból az adatot kitennénk a felhasználói felületre, előbb fel kell tölteni azt értékekkel. Ezt teszi a **DataAdapter Fill** metódusa. Ezután WindowForm esetén csak a **DataGrid DataSource** tulajdonságát kellett a DataSetre állítani, és megjelentek a táblázatban a DataSet adatai.

Web alkalmazás esetén meg kell hívnunk a **DataGrid DataBind** metódusát, melynek feladata, hogy a táblázatot a DataSource tulajdonságában megadott adatforráshoz kösse. Ennek hatására töltődik csak fel a táblázat a DataSet adataival.

Ahhoz, hogy hozzáférjünk az adatbázishoz, megfelelő jogosultsággal is rendelkeznünk kell. Az SQL Server esetén beállítandó jogosultság részletes leírása a gyakorlaton szerepel.

11.5. Teszt

1. Ahhoz, hogy Web alkalmazásokat tudjunk fejleszteni, szükség van az IIS telepítésére.
2. A Web Form a Form egy speciális fajtája, mely a szerverhez kapcsolódik, és futtatásához nem kell más szoftver, elég ha közvetlen hálózati kapcsolat van a szerver és a kliens gép között.
3. Web Form és Web Control esetén a felhasználói felület a böngészőben jelenik meg, míg a háttérkód a szerveren fut.
4. A Web Form nem minden vezérlőjének eseményeit kezeljük a szerveren, csak amelyikre be van állítva a runat="server" attribútum.

5. A Web Form FlowLayout beállítása azt jelenti, hogy folyamatosan a felrakott vezérlők sorrendjében történik az eseményfeldolgozás.
6. A PageLayout elrendezés esetén az oldal jobb szélére táblázat segítségével tehetünk vezérlőt.
7. A Toolbox Web Control kategóriájában felsorolt vezérlők nem tudnak szerver oldali kódot futtatni.
8. A Server Control eseményéhez rendelt kódot a Form.aspx.cs forrásfájlban találjuk.
9. Alapértelmezésben a Web Form megőrzi az állapotát, míg a HTML oldal stateless.
10. A vezérlőkbe beolvasott adatokat Validator objektumokkal ellenőrizhetjük az adatbázisba írás előtt. A kliens oldali ellenőrzést nem érdemes bekapcsolni, mert ha a böngésző nem támogatja a DHTML használatát, akkor az ellenőrzés elmarad.
11. Egy vezérlőhöz csak egy Validator tartozhat, de egy Validator több különböző ellenőrzést tud elvégezni.

Javítókulcs:

I, H, I, I, H, H, H, I, I, H, H.

"Informatics is like love, you cannot learn it from books."[1]

Graffiti

[1] Az informatika olyan, mint a szerelem, könyvekből nem tanulható!

GYAKORLATOK

1. Gyakorlat. Bevezető feladat

A könyv gyakorlatok része többnyire egymásra nem épülő feladatokat tartalmaz.

1.1. A háttér beállító alkalmazás

Ezen a gyakorlaton elkészítjük a Visual C# fejlesztőeszközt használó első alkalmazásunkat. Ismerkedünk a fejlesztői környezet nyújtotta lehetőségekkel. Kezeljük a felhasználó számára felkínált interfész, a form felület beállítási lehetőségeit. A nyomógombokhoz eseménykezelő függvényt készítünk. Módosítjuk a program ikonját, lehetővé tesszük az ablak minimalizálását, maximalizálását. Megtanuljuk, hogyan nyithatunk meg egy régebben írt projektet.

1.1.1. Előkészítés

Hozzon létre egy könyvtárat, melyben a tanulás során a Visual Studio C#-ban (ejtsd: szí sárpban) írt programjait tárolni fogja! Ha nem teszi, az alkalmazásvarázsló felkínálja a MyProjects alkönyvtárat.

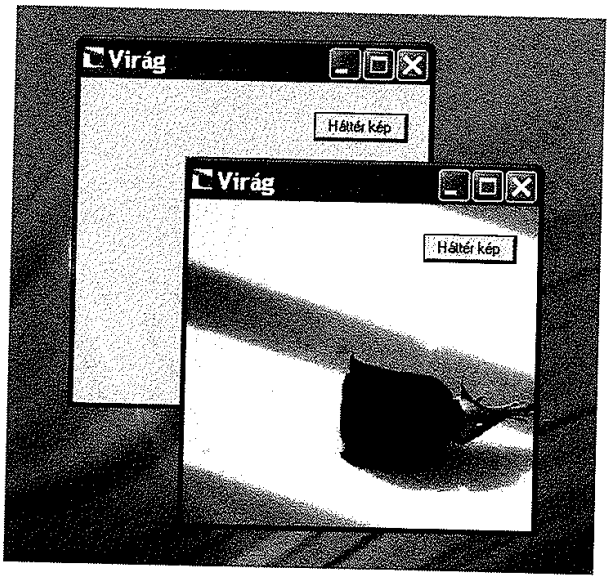
- **Indítsuk el a Microsoft Visual Studio .NET fejlesztőeszközt!**

  Start

  Programs

  Microsoft Visual Studio .NET

Az 1. Gyakorlat. 1. ábra a kész futó alkalmazás képét mutatja. Ha a 'Háttér kép' feliratú gombot leütjük, megjelenik a háttér kép.



1. Gyakorlat. 1. ábra A háttér beállító alkalmazás futás közben

Megjegyzés:

Aki már ismeri a Visual Studio korábbi verzióit, a kezdőlapon (Start Page) beállíthatja a profile ablakban az általa megszokott környezetet. Mi a következőkben Visual C# Developer beállítással dolgozunk.

1.1.2. Új projekt

File menü

New menüpont

Project

Project Types: Visual C# Projects

Templates: Windows Application

Name: Elso

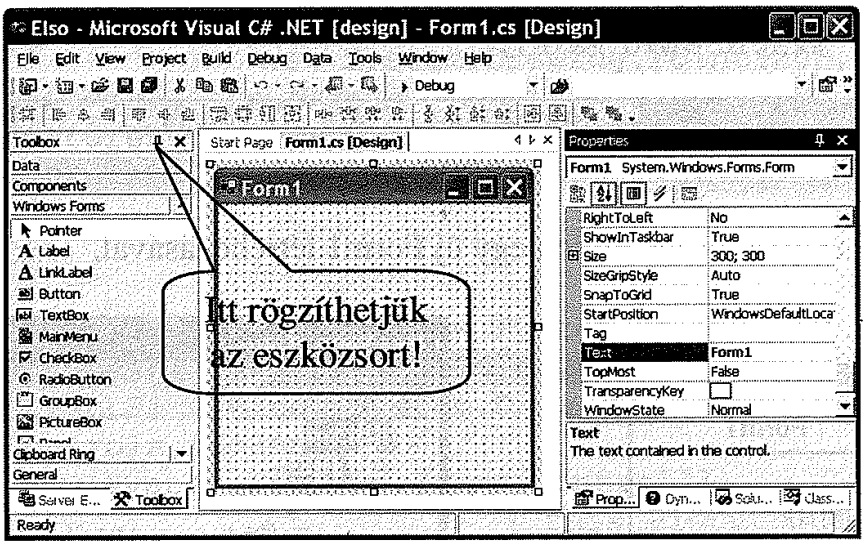
Location: A kívánt könyvtár

OK.

Ha a jobboldali Properties ablakot a fölötte elhelyezkedő Solution Explorert és Class Viewt tartalmazó ablak mellé húzzuk (drag & drop), jobban látjuk a tulajdonságokat. A címsorát megfogva a balegérgombot lenyomva mozgatjuk az

egeret, a keret mutatja az ablak helyét az egérgomb elengedése után. Ha az egér épp a fülekre mutat úgy engedjük el, akkor a Properties is a választható fülek egyike lesz.

Az ablak baloldalán a ToolBox (szerszámok ikon) fölé mozgatva az egérgombot az eszköztár megnyílik. A kitűzővel rögzíthetjük, hogy ne csukódjon vissza.

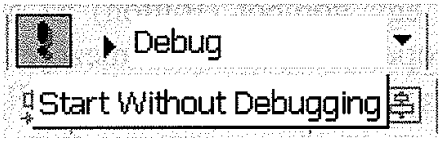


1. Gyakorlat. 2. ábra Az új alkalmazás a környezet beállítása után

Fordítás/Futtatás

Készítsük el az alkalmazást! A Build menü Build Solution vagy Build Elso utasítására elkészül az alkalmazás. Egy solution több projectből is állhat, de jelen esetben csak az Elso alkalmazás tartozik hozzá.

A Debug menü Start vagy Start Without Debugging menüpontjai segítségével futtathatjuk az alkalmazást. A nyomkövetéses futtatás ikonja az eszközsoron egy ▸ ikon. Javaslom, a nyomkövetés nélküli futtatás ikonját is tegyük mellé, hisz ez gyorsabban végrehajtódik. (1. Gyakorlat. 3. ábra) Az eszközsoron jobbegérrel kattintva, a gyorsmenü Customize parancsára a Commands fül Debug kategóriában megtalálható a parancs. Ikonját drag & drop technikával az eszközsoron a kívánt helyre húzhatjuk.

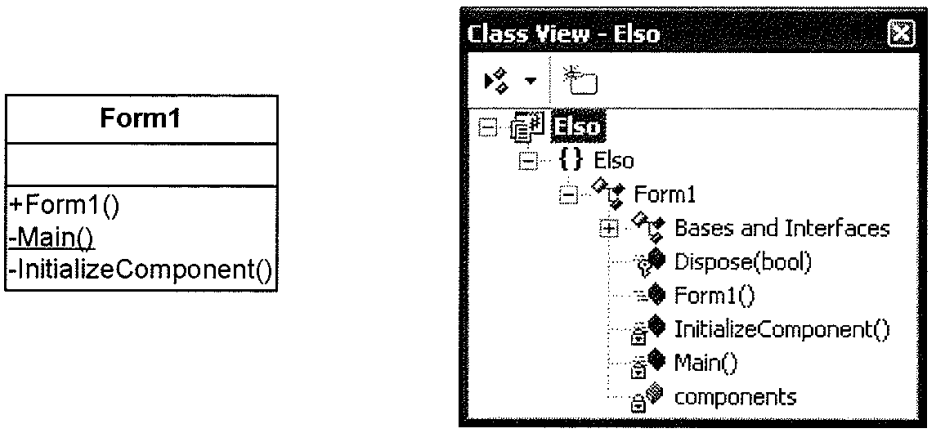


1. Gyakorlat. 3. ábra Az nyomkövetés nélküli futtatás eszközgombja az eszközsoron

A kódot majd lépésenként ismerjük meg. Most csak azt nézzük meg, hogyan indul el az alkalmazás! (A későbbiekben a projekt és a névtér fogalma is tisztázódik.) A Class View fül ablakában kinyitva az Elso projektet, majd az Elso névteret (namespace) a Form1 osztályban megtaláljuk a Main() függvényt, mely az alkalmazás indításáért felelős. Duplán kattintva a Main() nevén kinyílik a forráskódot tartalmazó ablak, és megnézhetjük a Main() kódját.

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

Ez azt jelenti, az alkalmazás fusson egy új Form1 létrehozásával.



1. Gyakorlat. 4. ábra A Form1 osztály UML diagramja balra, Class View nézet jobbra. Jól látható a konstruktor, a statikus Main és az InitializeComponent.

1.1.3. Kezdeti beállítások

Legyen a Form1.cs [Design] fül az aktuális, állítsuk be a háttérszínt és az ablak címsorát!

A Properties ablakban a BackColor mezőnél a nyílra kattintva válasszuk ki a színt (Web fül / LightBlue)! A Text mezőbe pedig írjuk a Virág szöveget!

Az eszköztárról húzzunk be egy gombot (Button) az ablakba! A gomb szövege legyen 'Háttér kép'! Ezt a Properties ablak Text mezőjében állíthatjuk be. A gomb kapjon 'beszédes ' nevet. A Name mezője legyen pictureButton-ra állítva!

Vizsgáljuk meg, mi történt a kódban!

A Class View fülben láthatjuk, hogy megjelent a Form1 osztályban a pictureButton tag. A kezdő állapot beállítása az inicializálás során történik (InitializeComponent).

```
private void InitializeComponent()
{
 this.pictureButton = new System.Windows.Forms.Button();
 this.SuspendLayout();
 //
 // PictureButton
 //
 this.pictureButton.Location =
                    new System.Drawing.Point(208, 32);
 this.pictureButton.Name = "pictureButton";
 this.pictureButton.TabIndex = 0;
 this.pictureButton.Text = "Háttér kép";
 //
 // Form1
 //
 this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
 this.BackColor = m.Drawing.Color.LightBlue;
 this.ClientSize = new System.Drawing.Size(292, 262);
 this.Controls.AddRange(
       new System.Windows.Forms.Control[]
                                 { this.PictureButton});
 this.Name = "Form1";
 this.Text = "Form1";
 this.ResumeLayout(false);
}
```

Az inicializálás során történik a háttérszín beállítása, mely vonatkozik a gomb színére is. A this a Form1 objektum.

Megjegyzés:

Ha az InitializeComponent metódust az osztálynézet helyett a kódban keressük, akkor a **Windows Form Designer Generated Code** felirat alatt találjuk meg. Ez a régió (kódrészlet) többnyire összecsukott állapotban jelenik meg, mert kódját jellemzően nem kézzel írjuk, hanem a tulajdonságlap beállításai alapján a fejlesztőeszköz generálja.

Fordítás/Futtatás

1.1.3.1. A this és a névtérhivatkozások elhagyhatóak

Próbáljuk ki, hogy ha a kódból kitöröljük a this. szövegrészeket, a fordító ugyanúgy megtalálja a metódusokat és tulajdonságokat, és a program hibátlanul lefut! Ha azonban a using direktívával hivatkozott névterek kiírását megjegyzésjelek mögé tesszük, a fordító azonnal hiányolja a névtér hivatkozást.

Fordítás/Futtatás

1.1.4. A gombon kattintás kezelése

Azt szeretnénk, hogy a 'Háttér kép' nevű gomb választása esetén megjelenne egy kép az ablak hátttereként! Ehhez két dolgot kell ismernünk.

1. Hogyan lehet képet a háttérbe betenni?
2. Hogyan lehet egy gombon a kattintás eseményt kezelni?

Próbáljuk meg először a kép megjelenítését az eseménykezelés nélkül, tehát kezdeti beállításokkal megvalósítani! Ez a megoldás segíthet abban is, hogy megtudjuk, milyen kód segítségével tehetjük ki a képet az ablakba.

Mielőtt nekilátnánk azonban, nézzük meg az alkalmazás könyvtárában elhelyezett két fájl méretét! Az Elso alkönyvtárban találunk egy Form1.resx nevű fájlt, mely a Form kirajzolásához szükséges információkat tárolja. Mérete 6 KB körül van. Ha a bin\Debug alkönyvtárban található exe fájlt nézzük, annak a mérete is hasonló.

Megjegyzés:

Ha a resx kiterjesztésű fájl tartalmát is meg akarjuk nézni, azt a Solution Explorer / Show All Files ikonját választva, és a Form1.cs foldert kinyitva az erőforrás adatait leíró XML fájlt nézhetjük meg Data és XML megjelenítésben.

1.1.5. A háttérkép beállítása

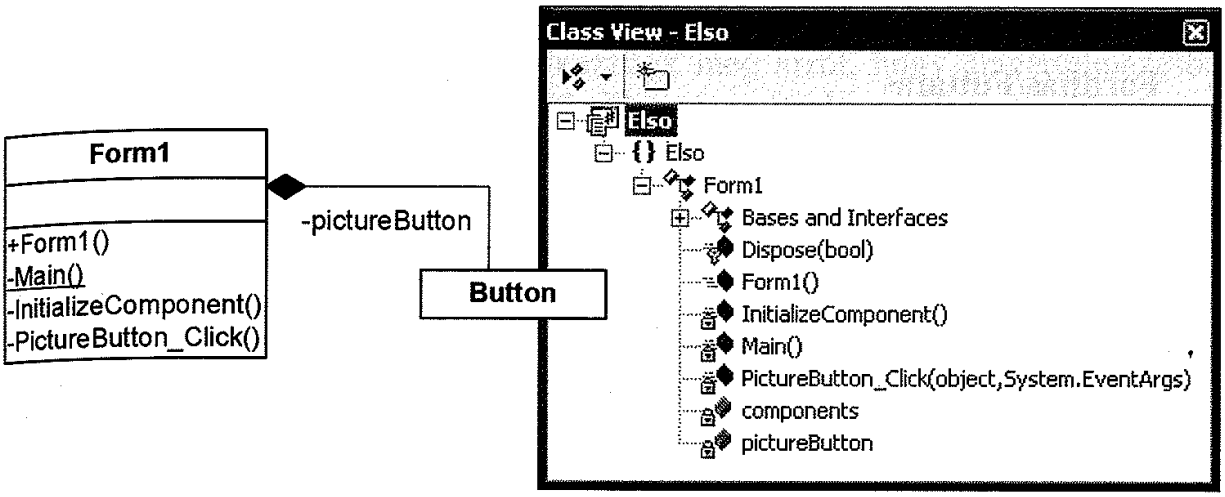
Nézzük meg, van-e a formnak olyan tulajdonsága, ami a háttérképet állítja! Legyen a Form1.cs [Design] aktuális! A Properties ablakban a Properties megjelenítésével megnézhetjük a form tulajdonságait, és a megjelenés (Appearance) kategóriában rögtön a BackColor alatt meg is találjuk a BackgroundImage tulajdonságot. A keresőgomb segítségével csak rá kell keresnünk a megjelenítendő képre, s azt már a szerkesztőablakban is láthatjuk. A kép természetesen nem állította át a háttér színét, amint azt a gomb színéből láthatjuk.

Ha elmentjük fájljainkat, nézzük meg a resx fájl méretét, jelentős változást érzékelhetünk. A fordítás és futtatás után az exe mérete is megnövekszik. A méretnövekedés arra utal, hogy a kép adatait mindkét fájl tartalmazza, vagyis a képfájlt nem kell az alkalmazás telepítésekor magunkkal vinni.

Megjegyzés:

A resx fájl tartalmát meg is nézhetjük, és megtaláljuk benne a képre vonatkozó adattömeget.

Ezután kezeljük a kattintás eseményt! Legyen a Form1.cs [Design] aktuális! Kattintsunk duplán az egérrel a gombon! Hatására elkészül a `pictureButton_Click` tagfüggvény, és már írhatjuk is a kódját.



1. Gyakorlat. 5. ábra Az Elso osztálydiagramja balra, Class View nézet jobbra. Jól látható az új pictureButton tag és a PictureButton_Click eseménykezelő metódus.

Azt szeretnénk, ha a gomb hatására megjelenne egy kép az ablak hátterében! Nézzük, hogyan valósította ezt meg a Form Designer! Nyissuk ki a `Windows Form Designer generated code` régiót, és az `InitializeComponent()` metódusban megtaláljuk a háttérkép beállítását.

```
this.BackgroundImage = ((System.Drawing.Bitmap)(
        resources.GetObject("$this.BackgroundImage")));
```

Írjuk ezt be az eseménykezelőnkbe!

```
private void PictureButton_Click(object sender,
                                 System.EventArgs e)
{
  this.BackgroundImage = ((System.Drawing.Bitmap)(
        resources.GetObject("$this.BackgroundImage")));
}
```

`The type or namespace name 'resources' could not be found` hibaüzenetet kapunk. Tehát a fordító nem ismeri fel a resources objektumot. Azért nem, mert az `InitializeComponent()` metódus első sorában létrehoztuk, itt pedig nem. A lokális objektumot nem látjuk a többi függvényből. Hozzuk létre a kód másolásával!

```
private void PictureButton_Click(object sender,
                                 System.EventArgs e)
{
 System.Resources.ResourceManager resources =
 new System.Resources.ResourceManager(typeof(Form1));
 this.BackgroundImage = ((System.Drawing.Bitmap)(
       resources.GetObject("$this.BackgroundImage")));
}
```

Fordítás/Futtatás

A kód most már lefordul, ahhoz hogy lássuk is a kép betöltődését, a két sort töröljük az `InitializeComponent()` kódjából.

Meg lehet-e mindezt egyszerűbben is valósítani? A System. Drawing névtérnek van egy Image osztálya is, amit a FromFile metódussal tölthetünk fel. A this-szel kezdjük a kód írását, mert a form tulajdonságát szeretnénk megváltoztatni. A pont leütése után megjelenik a hívható függvények listája. A BackgroundImage tulajdonságot szeretnénk állítani. A képet a futó alkalmazás könyvtárába Elso\bin\Debug-ba tegyük, vagy adjuk meg az elérési útját is?

```
private void PictureButton_Click(object sender,
                                 System.EventArgs e)
{
 this.BackgroundImage = Image.FromFile("Rozsa.jpg");
}
```

Fordítás/Futtatás

Megjegyzés:

Ha közvetlenül fájlból olvassuk be a képet, akkor a resx fájlból kitörölhetjük annak adatait. Legegyszerűbben a Data nézetben a sor kijelölésével. Ekkor visszacsökken a resx és fordítás után az exe mérete is, de a kép csak akkor tud betöltődni, ha az elérési útban megadott helyen a fájl elérhető az exe számára.

Változtassuk meg az ablak méretét! Azt tapasztaljuk, ha az ablak mérete a kép méreténél nagyobb, akkor újra kirajzolódik a kép. Azt is láthatjuk, hogy a Háttérkép feliratú gomb nem módosítja helyét az ablak szélességének változtatásakor.

Ha azt akarjuk, hogy a gomb az ablak jobb szélétől mindig azonos távolságra legyen, akkor rögzítsük azt a jobb szélhez! Ezt az Anchor tulajdonság

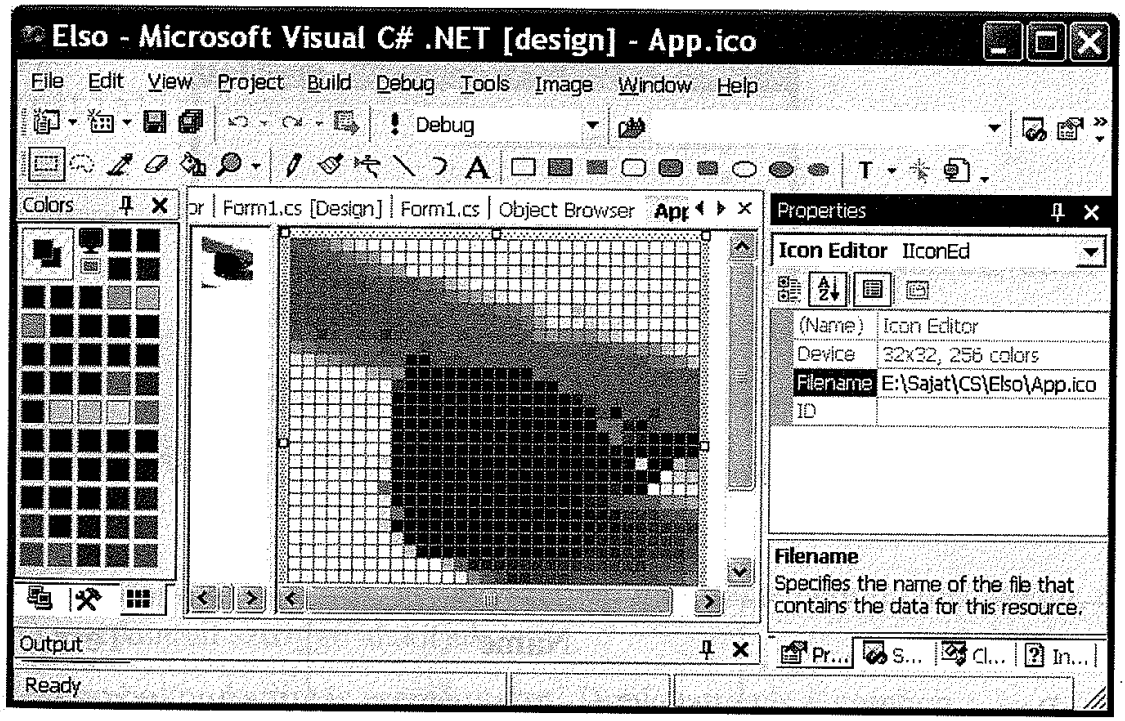
kiválasztásával tehetjük. (Design fül / Properties ablak / Anchor) A felső és a jobb szélhez rögzítsük a gombot!

1.1.6. Az alkalmazás ikonjának módosítása

A Solution Explorerben válasszuk ki az alkalmazás ikonját tartalmazó fájlt (App.ico)! A rajz eszköztárral megszerkeszthetjük az ikon képét! A vágólapon keresztül képet is másolhatunk bele.

A fordítás után azt láthatjuk, hogy a Debug könyvtár nem minden nézetében változott meg az ikon képe. Ne feledkezzünk meg arról, hogy alapértelmezésben minden ikonnak két változata van, a 32x32-es és a 16x16-os 16 színű. Ha csak az egyiket módosítjuk, akkor azokban a nézetekben, amikor a másikat látjuk, nem tapasztalható a változás. Az Image menü / Current Icon Image Types hatására választhatunk a szerkesztett képek között. Az Image menü / New Image Type segítségével vehetünk föl újabb ikontípusokat tárolásra. Minél több típust valósítunk meg, alkalmazásunk annál több környezetben lesz képes a környezet beállításainak megfelelő minőségű ikont alkalmazni.

A futáskor a címsorban megjelenő ikont a Form tulajdonságai közt, az Icon tulajdonságnál állíthatjuk be az App.ico fájlra.



1. Gyakorlat. 6. ábra Az ikonszerkesztő

1.2. Az érték típusú és a referencia típusú változók összehasonlítása

Készítsünk egy új alkalmazást az érték és referencia változó közti különbség bemutatására!

File

New

Project

Visual C# projects

Windows Application

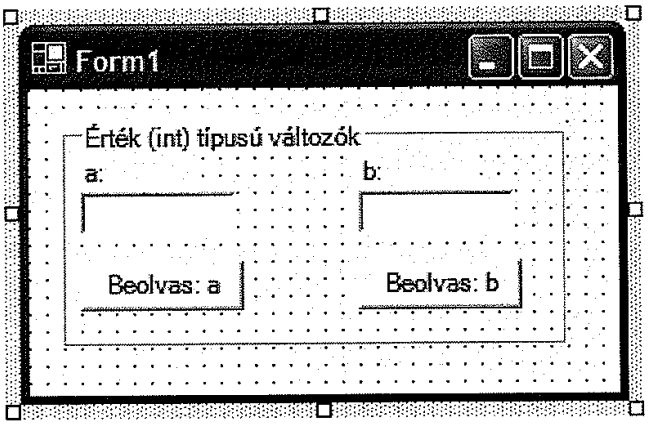
Name: ValRef

Location: beállítva

OK

1.2.1. Az érték típusú változók

Tegyünk a Formra egy GroupBox, két Label, két TextBox és két Button vezérlőt!



1. Gyakorlat. 7. ábra Az ablak a szerkesztőben

	Name	Text
GroupBox	valueGroupBox	Érték (int) típusú változók
Label		a: / b:
TextBox	aTextBox / bTextBox	
Button	aButton / bButton	Beolvas: a / Beolvas: b

Rendezzük őket a Layout eszközsor segítségével! Ha a GroupBox eltakarná a többi vezérlőt, a Send to Back eszközgomb segítségével „küldhető" hátra.

Vegyünk fel két int típusú tagváltozót a Form1 osztályba 'a' és 'b' néven!

Class View

Form1 jobbegér

Add

Add Field

Field sccess: private

Field Type: int

Field name: a

Finish

Állítsuk be a két változó kezdőértékét! Ez a konstruktor feladata, melyet úgy ismerünk fel, hogy neve megegyezik az osztály nevével.

```
public Form1()
{
 //
 // Required for Windows Form Designer support
 //
 InitializeComponent();

 //
 //TODO:Add any constructor code after
                                InitializeComponent call
 //
 a=5;
 b=a;
 aTextBox.Text=a.ToString();
 bTextBox.Text=b.ToString();
}
```

A Text tulajdonság sztringet vár, így egyszerűen int értéket nem tudunk átadni, de a ToString() metódussal sztriggé alakítjuk.

Kezeljük a gombokon kattintás eseményeket! Duplakattintás a gombon.

```
private void aButton_Click(object sender,
                                    System.EventArgs e)
{
 a=Convert.ToInt32(aTextBox.Text);
 aTextBox.Text=a.ToString();
 bTextBox.Text=b.ToString();
}
```

Az első sor az aTextBox vezérlőbe bevitt szöveget átalakítja int típusú változóvá, majd kiírja az a és a b értékét a képernyőre.

Fordítás/Futtatás

A futtatáskor azt látjuk amit vártunk, az aButton_Click az 'a' változóba olvassa be az értéket, a bButton_Click a 'b'-be. Az 'a' és a 'b' változó is rendelkezik saját memóriacímmel, így különböző értékeket tartalmazhatnak annak ellenére, hogy kezdetben ugyanazt az értéket kapták. (b=a;)

Ha a szövegmezőbe nem számot írunk, a konverzió során kezeletlen kivétel hibaüzenetet kapunk. A kivételkezelésről a következő fejezetben lesz szó.

1.2.2. A hivatkozás típusú változók

Valósítsuk meg ugyanezt a feladatot egyelemű tömbökre! Ne feledjük, az első elem sorszáma: 0!

```
private int[] ta;
private int[] tb;

public Form1()
{...
 a=5;
 b=a;
 aTextBox.Text=a.ToString();
 bTextBox.Text=b.ToString();
 ta=new int[1];
 tb=ta;
 ta[0]=3;
 taTextBox.Text=ta[0].ToString();
 tbTextBox.Text=tb[0].ToString();
}
```

Fordítás/Futtatás

A tb=ta; értékadás után a tb tömb ugyanazon a memóriaterületen dolgozik, mint a ta, hisz a tömbök referencia típusú változók, vagyis a memóriaterület címét

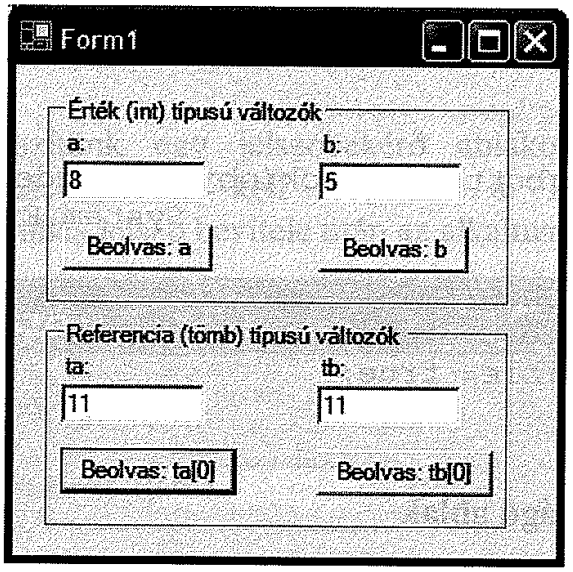
tartalmazzák. Tehát a két referencia ugyanarra a területre hivatkozik. Így láthatóan tb[0] értékét nem is kell megadnunk, az megegyezik ta[0] értékével.

Nézzük, mi történik, ha beolvasunk egy értéket bármelyik tömb 0. elemébe!

```
private void taButton_Click(object sender,

 System.EventArgs e)
{
 ta[0]=Convert.ToInt32(taTextBox.Text);
 taTextBox.Text=ta[0].ToString();
 tbTextBox.Text=tb[0].ToString();
}
```

Fordítás/Futtatás

Láthatóan csak a ta[0] értékét olvastuk be, mégis vele együtt változott tb[0] értéke is. Ha tb[0] értékét olvassuk be (tbButton_Click), vele módosul ta[0] értéke is.



1. Gyakorlat. 8. ábra Tömb esetén az értékek egyszerre módosulnak

Miért nem mutatható be ez sztring változó esetén, hisz a sztring is referencia típusú? Mert a sztring típusú változónak valójában nem módosítható az értéke. Vagyis a módosítás során új helyen foglalódik le a szöveg számára a hely. Így két azonos címre mutató sztring esetén, ha az egyik új értéket kap, akkor már nem mutatnak azonos címre.

1.3. Feladatok

1. Készítsünk egy alkalmazást, mely egy gomb választásának hatására egy üzenetablakot tesz ki a képernyőre **'Másik gomb'** felirattal! Az üzenetablak

bezárása után az első gomb eltűnik és megjelenik egy második. A második gombot választva ugyanez történik. Ez így folytatódik, amíg egy Nem unod még? Én már igen! feliratú szöveg után nem jelenik meg újabb gomb.*

2. Készítsünk születésnapi köszöntőkártyát! A kártyán nyomógombokkal választhassuk ki, milyen nyelven szeretnénk az üdvözlő szöveget látni! Tegyünk képe(ke)t is a szöveg mellé!

3. Készítsünk egy utazási iroda kínálatáról egy programot! Az utazás célpontjául felkínált városokat nyomógombok segítségével választhassuk ki! A kiválasztott város háttérképe legyen a város egy jellegzetessége, írjuk ki az ablakba az utazás kínálatával kapcsolatos információkat!

4. Készítsünk egy **állítható tulajdonságú ablak**ot! Az ablak tulajdonságait gombokkal lehessen állítani! Háttérszín, betűszín, mindig felül (TopMost), kéz alakú kurzor, fix méret.*

1.4. Megoldásötletek

1. 'Másik gomb'

```
private void button1_Click(object sender,
                                    System.EventArgs e)
{
 MessageBox.Show("Másik gomb");
 button1.Visible= false;
 button2.Visible= true;
}
```

4. Állítható tulajdonságú ablak

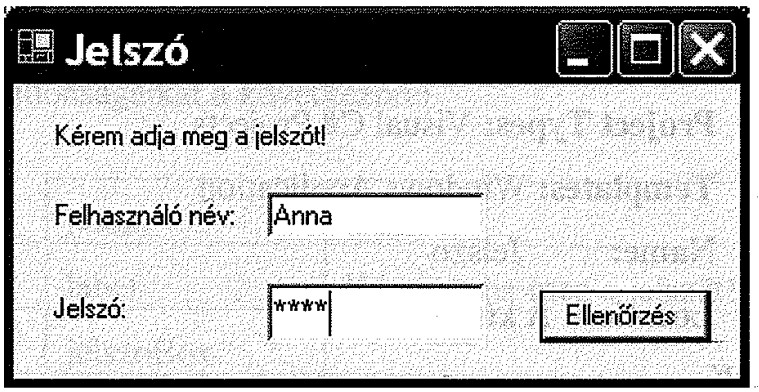
```
this.ForeColor=Color.Red;
this.TopMost=true;
this.Cursor=Cursors.Hand;
this.FormBorderStyle=FormBorderStyle.FixedDialog;
```

Kicsit összetettebb, de elegánsabb megoldás, ha színt beállító párbeszédablak objektumot használunk.

```
ColorDialog colorDlg = new ColorDialog();
colorDlg.ShowDialog();
this.BackColor=colorDlg.Color;
```

2. Gyakorlat. Jelszóbekérő ablak

A gyakorlat során készítünk egy jelszóbekérő ablakot, melybe különböző felhasználóknak más-más jelszót kell beírniuk a Helyes jelszó felirat megjelenéséhez. Három hibás jelszó bevitele után az alkalmazás leáll.

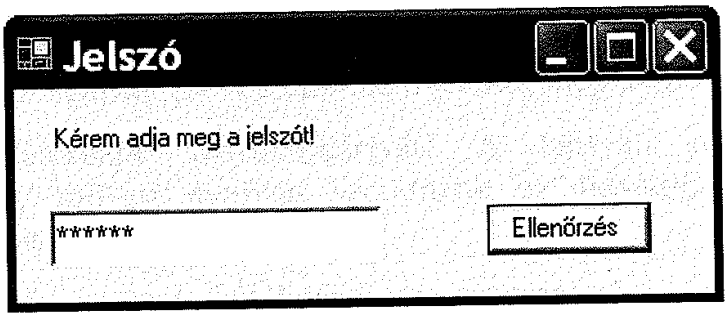


2. Gyakorlat. 1. ábra Az alkalmazás futás közben

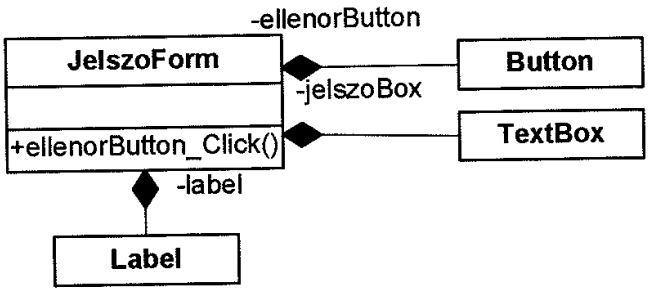
Egyszerű jelszóbekérő ablak. Jó jelszó esetén kiírja, hogy helyes a jelszó, hibás jelszó esetén leáll.

2.1. Az egyszerű jelszóbekérő ablak

Kérjük be a jelszót, és helyes jelszó esetén írjuk ki: helyes jelszó, s a program fusson tovább, míg hibás jelszó esetén figyelmeztetés után álljon le!



2. Gyakorlat. 2. ábra Egyszerű jelszóbekérő alkalmazás



2. Gyakorlat. 3. ábra az alkalmazás osztálydiagramja

- **Készítsünk egy új projektet!**

File menü

New menüpont

Project

Project Types: Visual C# Projects

Templates: Windows Application

Name: Jelszo

Location: A kívánt könyvtár

OK.

- **Nevezzük át a Form1 formot JelszoFormra!**

Form1.cs [Design]

Properties fül

Name mező: JelszoForm // Az osztály neve.

Text mező: Jelszó // A cím felirata.

Solution Explorer fül

Form1.cs-t átnevezni: JelszoForm.cs-re // A fájl neve.

JelszoForm.cs

Main: Application.Run(new **JelszoForm**()); // Már nincs Form1!

summary: A namespace deklaráció utáni megjegyzések közt is van egy Form 1.

Megjegyzés:

A /// -el írt és <summary> tegek közé zárt megjegyzéssorok a fordító számára valóban megjegyzések, de a kódszerkesztő a tulajdonságleíró súgó ablakban — (property description ToolTips), mely pl. az osztály objektuma fölé víve a kurzort jelenik meg —, innét olvassa a megjelenített adatokat. Ha tehát nem javítjuk itt a nevet, akkor később egy több osztályt, komponenst használó alkalmazásban lesz sok egymástól különböző osztályú, a 'súgó szerint Form1 típusú ablakobjektum, melyek osztályát már rég nem így hívjuk, csak a súgót nem tájékoztattuk erről.

Ezek a megjegyzések bekerülnek fejlesztőeszköz által generált XML dokumentációba is.

➢ Készítsük el a felületet!

A Toolbox-ból válasszunk egy Label, egy TextBox és egy Button vezérlőt! Módosítsuk tulajdonságaikat a következőre:

	Name	**Text**	**PasswordChar**
Label	label	Kérem adja meg a jelszót!	
TextBox	jelszoBox		*
Button	ellenorButton	Ellenőrzés	

Az ablakot ne engedjük méretezni, hisz a további területek úgyis üresek: A JelszoForm (Design) / Properties / **FormBorderStyle** legyen **Sizable** helyett **FixedDialog!**

Fordítás/Futtatás

Láthatjuk, hogy a szövegmezőben karakterek helyett csillagok jelennek meg. Ha túl kicsinek találjuk a csillagokat, a jelszoBox Font tulajdonságának Size értékét állíthatjuk nagyobbra.

➢ Kezeljük az ellenőrzés gombot!

Duplán kattintva a gombon generálódik a kezelőmetódus.

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 if (jelszoBox.Text != "AAA") Application.Exit();
 label.Text = "Helyes jelszó!";
}
```

Ez egy egyágú elágazás, ha nem „AAA" a jelszó, az alkalmazás leáll.

Ellenőrizze az **Enter** gomb leütése után is a jelszót! Ehhez a JelszoForm **AcceptButton** tulajdonságát az ellenorButton-ra kell állítani!

Ha azt akarjuk, hogy a kilépés előtt még írja ki, hogy hibás jelszó, majd várjon 2 másodpercig, s csak azután lépjen ki, akkor az igaz ág kódjába több utasítás kerül, ezeket egy blokkba kell elhelyezni!

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 if (jelszoBox.Text != "AAA")
 {
    label.Text = "Hibás jelszó!";
    System.Threading.Thread.Sleep(2000);   // Vár.
    Application.Exit();
 }
 label.Text = "Helyes jelszó!";
}
```

A várakoztatást a Sleep függvénnyel szokás megoldani. A súgó Indexben rákeresve találunk is egy Sleep method kulcsot. A .NET Framework Class Library szolgáltatja. A leírásnál láthatjuk, hogy C# metódus, hogy a Thread osztály tagfüggvénye, és a Thread a System.Threading névtérben van leírva. Paraméterét milliszekundumban kell megadnunk.

Ez egy osztályszintű metódus, tehát nem kell hozzá objektum, hogy meghívjuk. Vagyis nem hozunk létre új szálat, csak a szál osztály statikus metódusát hívjuk meg.

Megjegyzés:

A Sleep egy publikus, statikus metódus, és emiatt, bár a Thread osztály tagfüggvénye, nem kell a végrehajtásához egy új szál objektumot létrehozni.

Fordítás/Futtatás

A futtatás során, bár a program a leállás előtt várakozik, mégsem látjuk a 'Hibás jelszó' feliratot! Az Enter működik.

Biztosan van a Label osztálynak egy frissítő metódusa pl. Refresh? Ha a változó nevét begépelve kitesszük a '.' karaktert, a megjelenő súgóban próbálkozhatunk az R betűvel, s lám van Refresh metódusa!

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 if (jelszoBox.Text != "AAA")
 {
    label.Text = "Hibás jelszó!";
        label.Refresh(); // Frissítésre kényszeríti a
                                  // vezérlőt.
    System.Threading.Thread.Sleep(2000);   // Vár.
    Application.Exit();
 }
 label.Text = "Helyes jelszó!";
}
```

Fordítás/Futtatás

A jelenlegi kódnál mindegy, hogy a 'Helyes jelszó!' szöveget az elágazáson kívül, vagy az elágazás else ágában helyezzük el, mert hibás jelszó esetén az alkalmazás leáll, vagyis a kód semmiképp nem jut el az elágazás utáni kódsorokra. A szöveg most még arra is utalhat, hogy itt folytatódik az alkalmazás.

2.2. Három próbálkozás

A program csak a 3. hibás jelszó megadása után álljon le!

Ehhez szükségünk van egy számlálóra, melynek kezdőértéke 0, és értékét a hibás próbálkozások növelik. Nézzük meg, milyen egész típusok állnak rendelkezésünkre! A súgó index szerinti keresőjében (Index fül) adjuk meg az int kulcsszót! Az int keyword alcím alatt a többi egész típusról is információkhoz juthatunk. A byte a legkisebb nem negatív egész.

Class View fül

JelszoForm osztály, jobbegérgomb

Add

Add Field

Field access private

Field type byte

Field name missed

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 if (jelszoBox.Text != "AAA")
 {
    label.Text = "Hibás jelszó!";
    label.Refresh(); // Frissítésre kényszeríti a
                     // vezérlőt.
    System.Threading.Thread.Sleep(2000); // Vár.
    if (++missed == 3) Application.Exit();
 }
 label.Text = "Helyes jelszó!";
}
```

```
private byte missed=0;
```

A ++ még az összehasonlítás előtt növeli a missed értékét eggyel.

Fordítás/Futtatás

Azt látjuk, hogy a két másodpercig látszó 'Hibás jelszó!' felirat 'Helyes jelszó!'-ra vált. Igen, hisz most már nem áll le minden esetben az alkalmazás: hibás jelszó esetén, vagyis ha még nem értük el a 3 próbálkozást, akkor végrehajtódik az elágazást követő sor, mely átírja a szöveget. Most már tehát csak az else ágban helyes a 'Helyes jelszó!' felirat. Vegyük észre, hogy a megoldáshoz egymásba ágyazott elágazást használtunk! Mivel az igaz ág utasítássorozata egy blokkba van betéve, teljesen egyértelműen az else a külső if-hez tartozik.

```
private void ellenorButton_Click(object sender,
                                 System.EventArgs e)
{
 if (jelszoBox.Text != "AAA")
 { ...
    if (++missed == 3) Application.Exit();
 }
```

```
 else
    label.Text = "Helyes jelszó!";
}
```

Fordítás/Futtatás

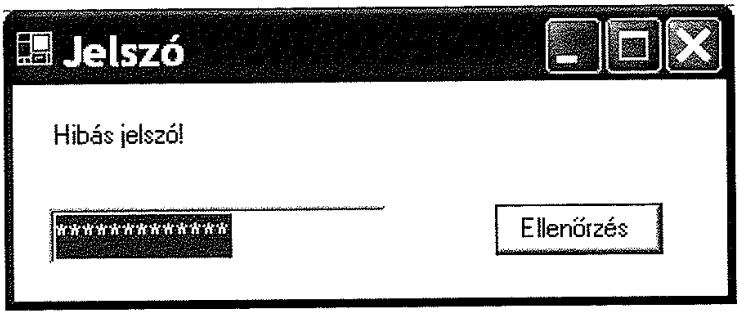
Szépítésként még hibás jelszó esetén kerüljön a kurzor a TextBox mezőbe, legyen kijelölve a tartalom, hogy gyorsabban lehessen az új jelszót megadni, és csak a kilépés előtt várakozzon, az új jelszót lehessen azonnal bevinni!

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 if (jelszoBox.Text != "AAA")
 {
    label.Text = "Hibás jelszó!";
    if (++missed == 3)
    {
        label.Refresh(); // Frissítésre kényszeríti a
                         // vezérlőt.
        System.Threading.Thread.Sleep(2000); // Vár.
        Application.Exit();
    }
    else
    {
        jelszoBox.Focus();
        jelszoBox.SelectAll();
    }
 }
 else
    label.Text = "Helyes jelszó!";
}
```

Megjegyzés:

Ha a blokk kezdetét és végét új sorba tesszük, a szerkesztő automatikusan tabulálja a záró blokkot a nyitó alá. Egy rövid ideig még vastagon ki is emeli a párt. A blokk záró kapocs kiírásakor a blokkban megírt kódot is tabulálja.

Fordítás/Futtatás



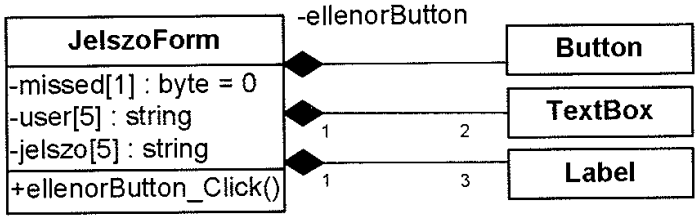
2. Gyakorlat. 4. ábra A hibás jelszó esete

2.3. Több felhasználó egyedi jelszava

Legyen több felhasználónk, akik különböző jelszóval rendelkeznek!

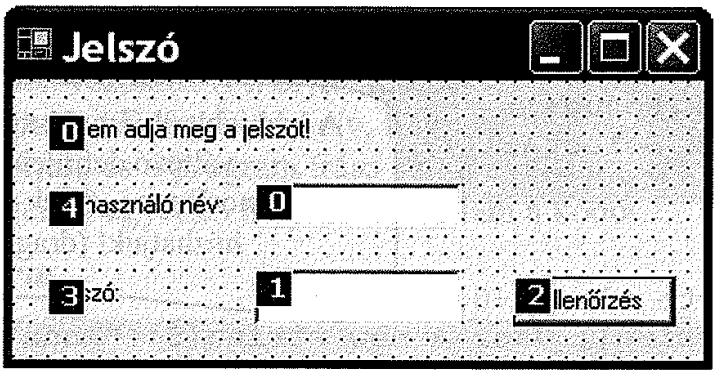
Az ablakban két szövegmező kell, egy a felhasználó nevének, egy pedig a jelszónak a bekéréséhez! Szükségünk van a felhasználókat és jelszavukat tároló tömbre! Egyszerűen írjuk be az osztályba a többi adattag mögé:

```
private string[] user = new string[5]{"Anna", "Pali",
                                      "Józsi","Mari","Kati"};
private string[] jelszo = new string[5]{"Anna", "Pali",
                                      "Józsi","Mari","Kati"};
```



2. Gyakorlat. 5. ábra az adatok tömbökben

A felhasználó azonosítóját bekérő TextBox neve userBox. A Layout eszközsor eszközgombjaival rendezzük a felületet! A View menü **Tab Order**. menüpontjával (vagy a tulajdonságlap Tab Index mezőjével) beállíthatjuk a Tab billentyű által meghatározott sorrendet, az úgynevezett **tabulátor sorrend**et. Egyszerűen az egérrel a kívánt sorrendbe kell a vezérlőkre kattintanunk. A legkisebb érték: 0, kezdetben a 0 indexű vezérlő az aktuális. A feliratok nem kerülnek fókuszba. A Tab Order a menüpont újraválasztásával kapcsolható ki.



2. Gyakorlat. 6. ábra tabulátor sorrend

```
private void ellenorButton_Click(object sender,
System.EventArgs e)
{
 // Megkeresi a user indexét.
 int i = 0;
 while ((i<5) && (userBox.Text != user[i]))
    i++;

 if (i<5)    // Ha megtalálta a usert.
    if (jelszoBox.Text == jelszo[i]) // Helyes jelszó.
       label.Text = "Helyes jelszó!";
    else     // Hibás jelszó.
    {
        label.Text = "Hibás jelszó!";
        if (++missed == 3)
        {
              label.Refresh(); // Frissítésre kényszeríti
                               // a vezérlőt.
              System.Threading.Thread.Sleep(2000); // Vár.
              Application.Exit();
        }
        else // Még nem hibázott 3x.
        {
              jelszoBox.Focus();
              jelszoBox.SelectAll();
        }
    }
```

```
    else  // Nincs ilyen user.
    {
       label.Text = "Hibás felhasználó név";
       userBox.Focus();
       userBox.SelectAll();
    }
}

private byte missed=0;
private string[] user = new string[5]{"Anna", "Pali",
                                      "Józsi","Mari","Kati"};
private string[] jelszo = new string[5]{"Anna", "Pali",
                                        "Józsi","Mari","Kati"};
```

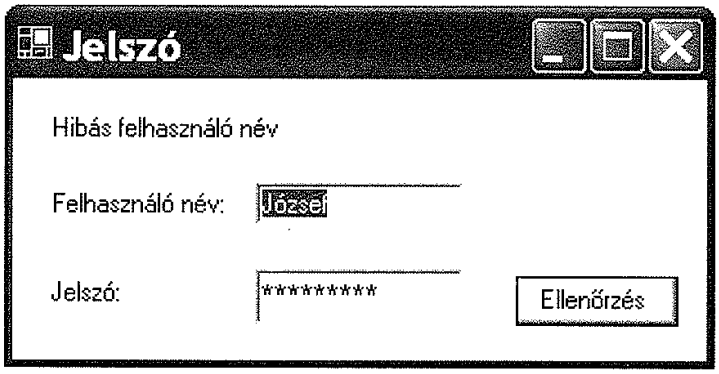
A kód jobb olvashatósága érdekében a változó definíciókat az ellenorButton_Click metódus elé húzhatjuk! (drag&drop)

A függvény elején egy while ciklussal megkerestük a beírt felhasználó indexét a user tömbben. Ha megtaláltuk, a találat helyének indexével lép ki az i, ha nem találtuk, akkor i = 5 értékkel lép ki. Ez alapján tesztelhető, hogy volt-e találat.

Megjegyzés:

A felhasználót a **Lineáris Keresés programozási tétel** segítségével kerestük meg. A programozási tételekkel a későbbiekben részletesen foglalkozunk.

Ha a hibás felhasználói nevet is a három hibába beszámoljuk, akkor a 'Nincs ilyen user' ágat is a 'Hibás jelszó' ágnak megfelelően ki kell egészíteni a missed növelő és vizsgáló, az alkalmazást leállító kóddal.



2. Gyakorlat. 7. ábra A hibás felhasználó esete

Megjegyzés:

A hibás jelszó (felhasználónév) után úgy is beállíthatjuk az ablakot, hogy kitöröljük a hibás szót, és a törlés után a kurzor az üres szövegmezőben várakozik az új bevitelre (jelszoBox.Clear();).

A felhasználók száma növekedhet vagy csökkenhet. Ekkor mindenütt a jelenlegi '5' értéket át kell írnunk az új értékre. Ha valahol elfelejtjük a módosítást, az súlyos programozási hibához vezethet. (Pl. olyan tömbindexre hivatkozunk, amilyen eleme már nincs is a tömbnek.) Ezért fontos elv, hogy az 'ilyen' értékeket a programban egy változóban tároljuk, és a program írása során mindenütt erre hivatkozzunk! Így változás esetén elegendő egyetlen helyen módosítani a kódot.

JelszoForm

Add

Add Field

Field access: private

Field type: int

Field name: numOfUsers

Filed modifiers: Constant

Field value: 5

Finish

Mivel a változó értékének változtatását a futás során nem engedélyezzük, legyen konstans! Ekkor kötelező az érték kitöltése is.

Edit menü

Find & replace

Replace

Find what: 5

Replace with: numOfUsers

Find Next és ha cserélni kívánjuk a számot: **Replace**

```
private byte missed=0;
private string[] user = new string[numOfUsers]{"Anna",
                         "Pali", "Józsi","Mari","Kati"};
private string[] jelszo = new string[numOfUsers]{"Anna",
                         "Pali", "Józsi","Mari","Kati"};

private void ellenorButton_Click(object sender,
                                      System.EventArgs e)
{
 int i = 0;
 while ((i<numOfUsers) && (userBox.Text != user[i]))
   i++;
```

```
    if (i<numOfUsers) // Ha megtalálta a usert.
...
}

private const int numOfUsers = 5;
```

A tömbök megfelelő számú elemének beállítását a fordító ki fogja kényszeríteni.

A felhasználónevek legördülő listából történő kiválasztására, és az alkalmazás főablakának megnyitására a 3. gyakorlat, új felhasználó felvételére a 4. gyakorlat, míg a jelszó fájlba mentésére az 5. gyakorlat feladatai közt látunk példát.

2.4. Feladatok

1. Módosítsuk alkalmazásunkat úgy, hogy minden felhasználó három jelszóbeviteli lehetőséggel rendelkezzen! Pontosabban, ha változik a felhasználó, akkor kezdjük újra a hibás jelszavak számlálását!
2. **Kép ki-be**. Az 'Elso' alkalmazást módosítsuk úgy, hogy ha nincs kép a háttérben, akkor legyen, de ha van, akkor tűnjön el! A gomb felirata is változzon a feladatnak megfelelően!*
3. **Automatikus szemétgyűjtő demo**. Hozzunk létre egy osztályt, mely egy statikus adattagban számolja a példányai számát! Egy alkalmazásban gombnyomásra hozzunk létre *egy változóban* sok példányt! Nézzük meg, hogyan szabadítja fel az automatikus szemétgyűjtő az elengedett példányokat!*
4. **Sztringek összehasonlítása**. Készítsünk alkalmazást, mely sztringeket hasonlít össze! Legyen egy az == operátort, egy az Equal metódust és egy a CompareTo metódust használó gombunk! Az eredményt írjuk ki a két beviteli mező közé! Ha valamelyik sztring üres, azt külön írjuk ki!*
5. Készítsünk egy alkalmazást, mely a szünet előtt kiírja a VAKÁCIÓ szöveget! Először csak az utolsó Ó-betűt lássuk, az egérrel kattintva rajta jelenjen meg az I, és minden további betűn kattintva az előtte álló betű. Esetleg a végén egy Végre itt a vakáció!!! párbeszédablak is előbukkanhat!
6. **Happy 15th birthday!** Adja meg, hogy hányadik születésnap (15), és megjelenik a 'Happy 15th birthday!' felirat valami jópofa képpel a háttérben.*

2.5. Megoldásötletek

2. Kép ki-be

```
private void PictureButton_Click(object sender,
                                 System.EventArgs e)
{
 if (this.BackgroundImage == null)
 {
    this.BackgroundImage = Image.FromFile("Rozsa.jpg");
    PictureButton.Text="Nincs kép";
 }
 else
 {
    this.BackgroundImage=null;
    PictureButton.Text="Háttér kép";
 }
}
```

3. Automatikus szemétgyűjtő demo

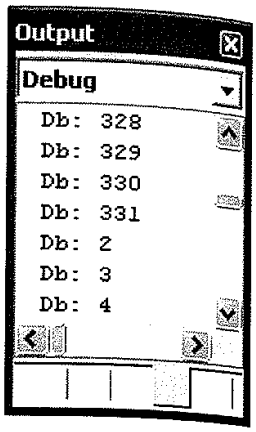
Hozzunk létre egy GC nevű alkalmazást! A Solution Explorerben a projekten jobbegérrel kattintva: Add / Add class / MyClass.

```
public class MyClass
{
 public static int db=0;
 public MyClass()
 {
    db++;     // Létrehozáskor növel.
 }
 ~MyClass()
 {
    db--;     // Felszámoláskor csökkent.
 }
}
```

Az alkalmazásunk formján egy gomb a következő kódot hívja:

```
private void button1_Click(object sender,
                                 System.EventArgs e)
{
 for (int i=0;i<1000;i++)
 {
    MyClass m = new MyClass();
    Console.WriteLine("Db: " +MyClass.db);
 }
}
```

Nézzük futás közben az Output ablakban (View menü / Other Windows) a kiírt számokat, miközben többször megnyomjuk a gombot! Görgessük vissza az Output ablak tartalmát, keressük meg a pontokat, ahol automatikus szemétgyűjtés történt! Ne feledjük, mi nem szüntettünk meg egyetlen objektumot sem, csak elengedtük őket!



2. Gyakorlat. 8. ábra Az Output ablak az éppen létező objektumok számát mutatja.

4. Sztringek összehasonlítása

```
private void compareButton_Click(object sender,
                                 System.EventArgs e)
{
 compareLabel.Text=saTextBox.Text.CompareTo(
                          sbTextBox.Text).ToString();
}

private void equalButton_Click(object sender,
                                       System.EventArgs e)
{
 if (saTextBox.Text== "" || sbTextBox.Text == "")
    compareLabel.Text="üres";
 else if (saTextBox.Text == sbTextBox.Text)
        compareLabel.Text = "=";
    else
        compareLabel.Text = "<>";
}
```

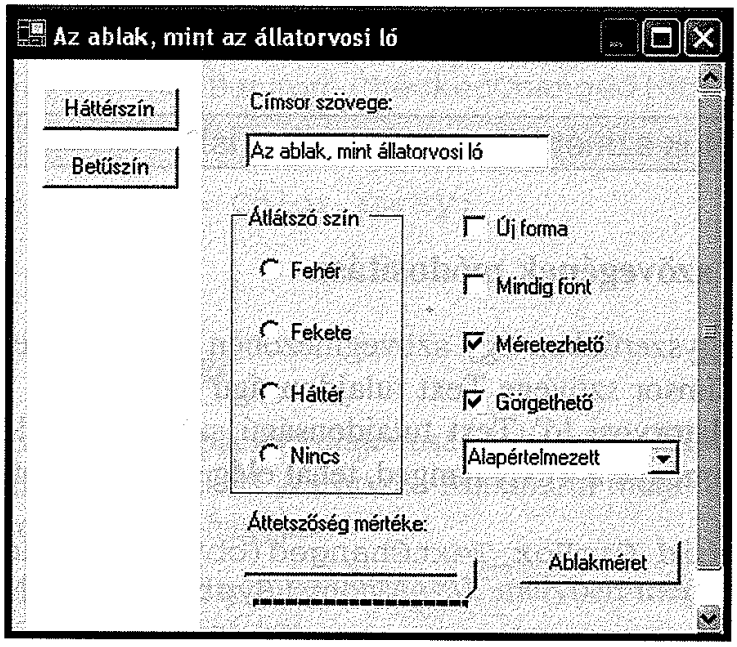
6. Happy 15th birthday!

```
switch (age % 10)
{
 case 1:
    suffix = (age / 10 == 1) ? "th": "st";
    break;
 case 2:
    suffix = (age / 10 == 1) ? "th": "nd";
    break;
 case 3:
    suffix = (age / 10 == 1) ? "th": "rd";
    break;
 default:
    suffix = "th";
    break;
}
```

3. Gyakorlat. A vezérlők használata

3.1. Az ablak mint állatorvosi ló

Készítsünk egy ablakot, amin megpróbáljuk bemutatni a Form és a vezérlők beállításainak minél több lehetőségét!



3. Gyakorlat. 1. ábra A kész ablak

File New Project Visual C# Projects Windows Application

Name: Lo

Nevezzük át a Form osztályunkat LoForm-ra!

3.1.1. A háttérszín beállítása futásidőben

Az ablak háttérképét már állítottuk az első gyakorlaton, most tegyünk egy Háttérszín feliratú gombot ablakunkba! A kattintás esemény a gombon alapértelmezett, tehát miután a Name:: HatterSzin_Box és Text: Háttérszín tulajdonságát beállítottuk, elegendő duplán kattintani az egérrel a gombon, és elkészül az eseménykezelő. A színt a kész ColorDialog ablak segítségével szokás kiválasztani. Tegyük ki azt modálisan a képernyőre!

```
private void HatterSzinButton_Click(object sender,
                                    System.EventArgs e)
{
 ColorDialog dlg= new ColorDialog();        // Új ablak.
 if (dlg.ShowDialog()==DialogResult.OK)  //Kiteszi modál.
   BackColor = dlg.Color;                   // Kiolvassa a színt.
}
```

A form háttérszíne legyen a párbeszédablakban kiválasztott színnel egyenlő!

Fordítás/Futtatás

Nézzük meg, valóban nem választható a főablak, amíg a színbeállító ablak nyitva van!

Készítsük el a betűk színét változtató gombot is!

3.1.2. A címsor szövegének módosítása

Egy szöveget legegyszerűbben egy szövegmezőben vihetünk be. Vegyünk fel egy CimLabel nevű, Címsor szövege Text tulajdonságú szöveget és egy CimBox nevű 'Az ablak mint állatorvosi ló' Text tulajdonságú szövegmezőt! Mivel a TextBox alapértelmezett eseménye a TextChanged, tehát elég kettőt kattintanunk rajta:

```
private void CimBox_TextChanged(object sender,
                                System.EventArgs e)
{
 Text = CimBox.Text;
}
```

Fordítás/Futtatás

Láthatóan amint változik a szövegmezőbe írt szöveg, úgy módosul a cím.

3.1.3. Az ablak bizonyos színek mögött legyen átlátszó!

Először is tegyünk fel egy panelt az ablakra úgy, hogy a két háttérszínt állító gomb fölötte legyen! A panel háttérszínét állítsuk fehérre!

Vegyünk fel egy GroupBox vezérlőt 'Átlátszó szín' felirattal! Tegyünk bele egymás után 4 választógombot (RadioButton)! Feliratuk és nevük legyen rendre: Fehér, Fekete, Control, Nincs. Kezeljük a CheckedChanged eseményüket!

```
private void WhiteButton_CheckedChanged(object sender,
                                        System.EventArgs e)
{
 TransparencyKey = System.Drawing.Color.White;
}

private void ControlButton_CheckedChanged(object sender,
                                        System.EventArgs e)
{
 TransparencyKey = System.Drawing.SystemColors.Control;
}
```

Annak visszaállításához, hogy ne legyen egyik szín sem átlátszó, állítsuk üres színre az átlátszó tulajdonságot!

```
private void NincsButton_CheckedChanged(object sender,
System.EventArgs e)
{
 TransparencyKey = Color.Empty;
}
```

3.1.4. Az ablak áttetszősége

Az áttetszőség mértékét az Opacity tulajdonság adja százalékban. Egy csúszkával (TrackBar) módosítsuk az értékét! A csúszka Maximum tulajdonságát állítsuk 100-ra, míg Minimum értékét 0-án hagyjuk.

A Form rendelkezik egy Opacity tulajdonsággal, melyben megadhatjuk, hogy hány százalékban kívánjuk látni az ablakot.

Készítsünk egy csúszkát, mellyel az ablak áttetszősége szabályozható! Mivel az Opacity tulajdonságot %-ban adjuk meg, célszerű a csúszka (TrackBar) értékeit 0 és

100 közé venni. Húzzunk be a formra egy TrackBar vezérlőt! Neve legyen opacityTrackBar, a **Maximum** tulajdonságát állítsuk **100**-ra (Minimum marad 0), és az értékét (**Value**) is **100**-ra állítsuk kezdetben! Majd kezeljük a Scroll eseményét!

```
private void opacityTrackBar_Scroll(object sender,
                                    System.EventArgs e)
{
 Opacity = ((float)opacityTrackBar.Value)/100;
}
```

A TrackBar értéke int típusú, ha egyszerűen 100-zal osztjuk, az egész rész 0 lesz, tehát azonnal eltűnik az ablak. Ha előbb float-tá konvertáljuk, akkor az osztás eredményébe is lehet törtrész.

Fordítás/Futtatás

Ne feledjük az egeret a csúszkán tartani, mert a teljesen átlátszó ablakot értelemszerűen nem látjuk. Ilyenkor a futtatókörnyezet Stop debugging gombjával vagy Ctrl + Alt + Del után Task List-ből állíthatjuk le.

Ha kipróbáljuk, azt tapasztaljuk, hogy az első pillanatban egy kicsit elfeketedik az ablak, de utána szépen működik az áttetszőség. A feketedést az okozza, hogy mivel az ablak nem volt áttetsző, így nem volt szüksége a háttér információk tárolására, ezt az első beállítás során tölti be. Ha kezdetben 99%-ra állítjuk az áttetszőséget (Opacity), akkor bár ez még nem érzékelhető, mégis már kezdetben szükség lesz a háttérképre. Ekkor a program indulásakor látjuk a fekete villanást, de később nem.

Teljes átlátszóság beállításakor, bár az ablakot eltüntettük a képernyőről, a tálcán látható, és a gyorsmenüvel be is zárható. Ha azt akarjuk, hogy innét is tűnjön el, a ShowInTaskbar tulajdonságát állítsuk hamisra!

```
private void opacityTrackBar_Scroll(object sender,
                                    System.EventArgs e)
{
 Opacity = ((float)opacityTrackBar.Value)/100;
 ShowInTaskbar=Opacity!=0;
}
```

3.1.5. A tulajdonságok ki-be kapcsolása jelölőnégyzettel

Legyen néhány olyan tulajdonsága ablakunknak, ami egymástól függetlenül állítható! Ezek a tulajdonságok legyenek ki-be kapcsolhatóak!

A tulajdonságok a következők:

- Az ablak formájának módosítása.

- Az ablakunk legyen mindig fönt, akkor se tűnjön el, ha egy másik ablakot helyezünk fókuszba!
- Méretezhetőség állítása.
- Görgethetőség állítása.

3.1.5.1. Az ablak formájának módosítása

Tegyünk ki egy jelölőnégyzetet a Formra, és kezeljük a választás (változtatás) eseményt!

LoForm.cs(Design)

CheckBox vezérlő a ToolBoxról

Properties

Name: regionCheckBox

Text: Új forma

Events: CheckedChanged duplaklikk

Próbáljuk meg hagyományos gondolkodással megoldani a feladatot! Van az ablaknak egy kezdő régiója, legyen ez a firstRegion, és van egy másik, amit választhatunk (region). A jelölőnégyzet ki-be kapcsolása ezek közötti váltogatást jelent.

A firstRegion megkaphatja a kezdőállapot értékét, a region pedig a konstruktorban kap kezdőértéket. Ügyeljünk rá, hogy a jelölőnégyzet a választott területre essen!

```
private Region firstRegion;
private Region region;

public LoForm()
{
 InitializeComponent();

 //
 // TODO: Add any constructor code after
 //                          InitializeComponent call
 //
 firstRegion = this.Region;
 GraphicsPath gp = new GraphicsPath();
 gp.AddRectangle(new Rectangle(0,0,Width,60));
 gp.AddEllipse(10, 62, Width-20,100);
 region = new Region(gp);
}
```

```
private void regionCheckBox_CheckedChanged(object sender,
                                           System.EventArgs e)
{
 if (this.Region == firstRegion)
    this.Region = region;
 else
    this.Region = firstRegion;
}
```

Megjegyzés:

A Form osztály Region tulajdonsága mögött, feltehetően egy region privát adattag tárolja a form aktuálisan beállított régióját. Nem probléma, ha az utódosztályban azonos nevű adattagot hozunk létre, mert az a tagfüggvényekhez hasonlóan eltakarja az ősosztály azonos nevű adattagját. Ez fontos, hisz a súgó nem mutatja meg a privát tagok nevét, mert azokhoz a fejlesztő nem férhet hozzá.

Fordítás/Futtatás

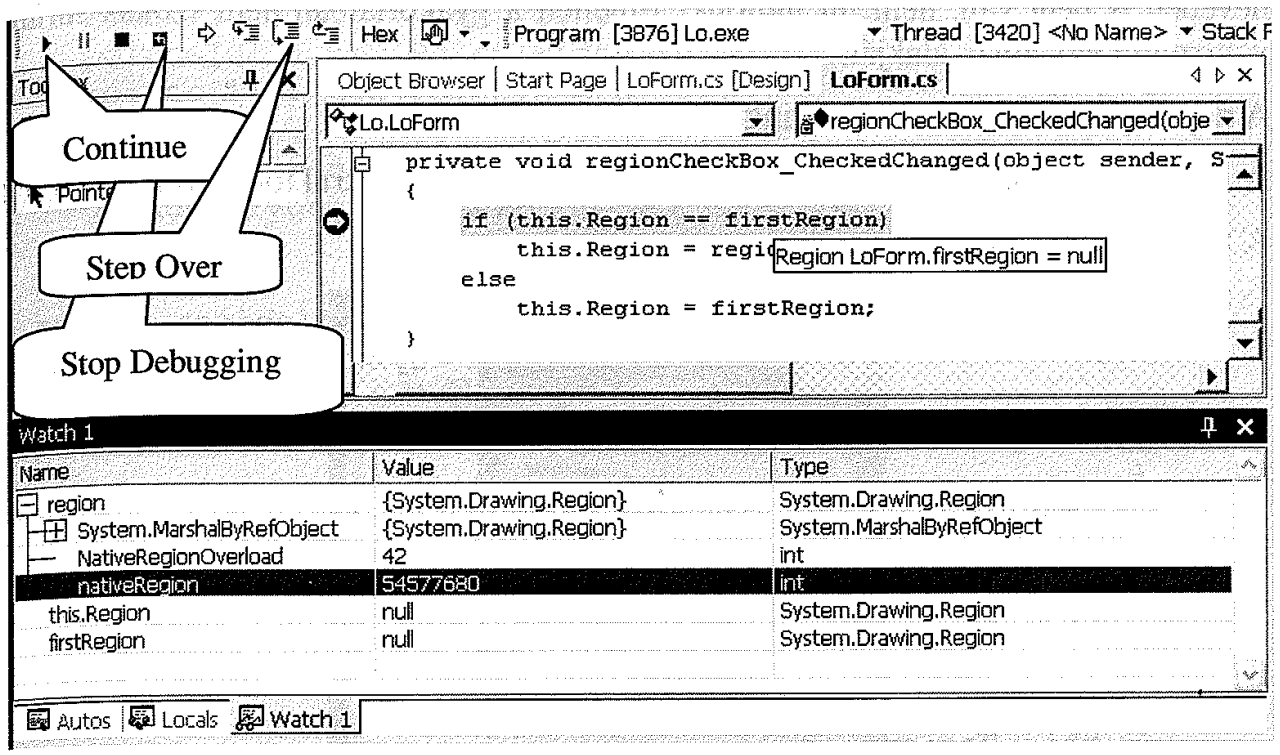
Már az írás során is fel kellett tűnnie, hogy nem működik a súgó, de a fordítás során a GraphicsPath-t nem ismeri fel. Ha a szóra kattintva az F1 gombot megnyomjuk, a súgó About GraphicsPath class leírásának végén a Reqiurementsben megtaláljuk, hogy az osztály a System.Drawing.Drawing2D névtérben szerepel, mely a System.Drawing.dll-ben található. Ezt a dll-t már használjuk, így csak a névteret kell a using direktívával lehivatkozni.

Fordítás/Futtatás

Azt tapasztaljuk, hogy egyszeri ki-be kapcsolás működik, de másodszorra hibaüzenetet kapunk.

Vizsgáljuk meg nyomkövetővel a hiba okát! Tegyünk töréspontot a `regionCheckBox_CheckedChanged` függvény első sorához úgy, hogy a kód melletti szürke csíkra kattintunk az egérrel! Ott egy bordó pont jelenik meg, amit ugyanígy vehetünk ki. Ennek hatására, ha nyomkövetéssel futtatjuk a programot (Start), akkor le fog állni a végrehajtás, amikor ehhez a sorhoz ér.

Kezdetben még a form Region tulajdonsága azonos a firstRegion-nal. A képernyő alján (3. Gyakorlat. 2. ábra) látható Watch ablakba írjuk be a vizsgálni kívánt változók értékét! Egyébként a kódban az egérmutatót bármely változó fölé mozgatva a megjelenő ablakban megnézhetjük annak pillanatnyi értékét.



3. Gyakorlat. 2. ábra Nyomkövetés közben a kód sárga nyíllal jelzett sora következik a végrehajtásban

Az első, amit észreveszünk, hogy a firstRegion értéke null, ezt pedig fölösleges külön változóban tárolni. Mivel a gondot a harmadik választás okozta, addig menjünk tovább (continue)! A második választás után láthatjuk, hogy a this.Region az első választás során felvette a region értékét. A kód mostani végrehajtása során (StepOver) pedig újra null értéket kap. Continue. A harmadik megállás során azt tapasztaljuk, hogy miközben a this.Region értéke null lett, a region nativeRegion tagja is 0 értéket kapott. Ez okozza aztán a hibát.

Vagyis a kód írása során nem vettük figyelembe, hogy a C# objektumok referenciák. Kezdetben a form Region tulajdonsága nem mutat sehova (null). Később ráállítjuk a konstruktorban elkészített régióra, ami azt jelenti, hogy a region és a this.Region azonos memóriaterületre mutatnak. Amikor a this.Region null értéket kap – a nativeRegion adattagnak is el kell engednie hivatkozását, hogy az automatikus szemétgyűjtő felszabadíthassa –, vele együtt a region is elveszíti az általa mutatott címet. Tehát hiába terveztük el, ezzel a módszerrel nem lehet visszaállítani a regionban kezdetben meghatározott állapotot.

Vagyis, ha megőrizni nem tudjuk, akkor létre kell hozni minden alkalommal az új régiót, amit aztán az automata-szemétgyűjtő szabadít fel, amikor már nem használjuk. Azért, ha futás közben megnézzük a Windows feladatkezelő (Ctrl + Alt + Del) folyamatok ablakában az alkalmazás által felhasznált memóriát, az a váltásokkal egyre növekszik. Ne feledjük, az automatikus szemétgyűjtőt, mint a

nevében is bent van, nem mi indítjuk! Emellett a gyakorlatban nem szokás sűrűn változtatni egy ablak alakját.

A nyomkövetést a Stop Debugging választásával állíthatjuk le.

A javítás után nincs szükség a két Region adattagra és a konstruktorba írt kódra.

A Kód a javítás után:

```
private void regionCheckBox_CheckedChanged(object sender,
                                           System.EventArgs e)
{
 if (regionCheckBox.Checked)   //(this.Region== firstRegion)
 {
    GraphicsPath gp = new GraphicsPath();
    gp.AddRectangle(new Rectangle(0,0,Width,60));
    gp.AddEllipse(10, 62, Width-20,100);
    this.Region = new Region(gp);
 }
 else
    this.Region = null;
}
```

Ügyeljünk rá, hogy a régiót állító jelölőnégyzet, de legalább a főmenü ikonja vagy a bezáró gomb elérhető legyen, mert különben a tálcán jobbegérrel, vagy a feladatkezelőt hívva a Task List-ben kell a futó alkalmazást leállítanunk!

Megjegyzés:

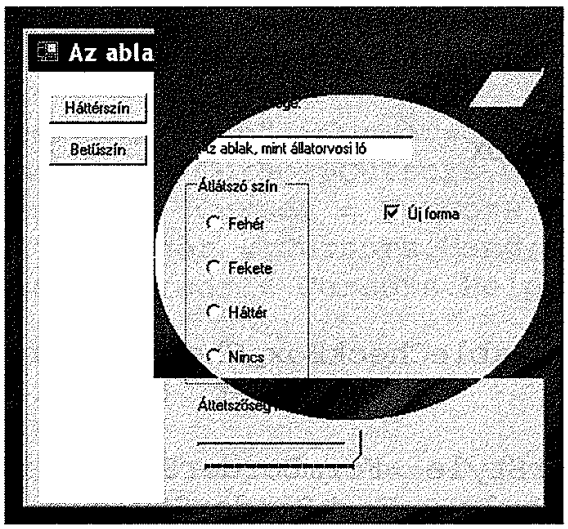
A Region osztálynak van egy Clone nevű metódusa, mely egy másolatot készít az objektumról. Ha a Region tulajdonságot e másolattal tesszük egyenlővé, akkor megőrizhetjük az eredeti változó értékét. Ez azonban minden alkalommal újra létrehoz egy új változót, tehát a kód nem lesz hatékonyabb a fent leírtnál.

Másrészről az ablakméret változtatása esetén, ha azt akarjuk, hogy a felirat töltse ki az egész ablakot, akkor úgyis újra kell számolni a régiót.

Alakítsuk át a régiót LÓ feliratra!

```
GraphicsPath gp = new GraphicsPath();
gp.AddRectangle(new Rectangle(0,0,100,Height));
gp.AddRectangle(new Rectangle(100,Height-100,
                                    Width-100,100));
gp.AddEllipse(100, 50, Width-100,Height-100);
```

```
Point[] points = new Point[4];
points[0] = new Point(Width-40,30);
points[1] = new Point(Width,30);
points[2] = new Point(Width-20,60);
points[3] = new Point(Width-60,60);
gp.AddPolygon(points);
```



3. Gyakorlat. 3. ábra Az ablak formája a Ló szó

Ha azt akarjuk, hogy a méretezés számolja újra a régiót, akkor a LoForm SizeChanged eseményében kell ezt elvégeztetnünk!

LoForm[Design] / Properties ablak / Events fül / SizeChanged duplaklikk

```
private void LoForm_SizeChanged(object sender,
                                System.EventArgs e)
{
 regionCheckBox_CheckedChanged(sender,e);
}
```

3.1.5.2. A TopMost tulajdonság

Új jelölőnégyzet:

Name: topCheckBox

Text: Mindig fönt

```
private void topCheckBox_CheckedChanged(object sender,
                                        System.EventArgs e)
    {
        TopMost=!TopMost;
    }
```

3.1.5.3. A méretezhetőség állítása

Új jelölőnégyzet:

Name: sizableCheckBox

Text: Méretezhető

Checked: True

```
private void sizableCheckBox_CheckedChanged(object
                            sender, System.EventArgs e)
{
 if (FormBorderStyle==FormBorderStyle.Sizable)
    FormBorderStyle = FormBorderStyle.FixedSingle;
 else
    FormBorderStyle = FormBorderStyle.Sizable;
}
```

3.1.5.4. A görgethetőség állítása

```
private void scrollCheckBox_CheckedChanged(object sender,
                                        System.EventArgs e)
{
 AutoScroll = !AutoScroll;
}
```

3.1.6. A kurzorváltás

Combo boksz:

Name: cursorComboBox

Text: Alapértelmezett

Items / Collection:

Alapértelmezett

Homokóra

Kereszt

Kéz

Súgó

Az alapértelmezett kezelőmetódus:

```
private void cursorComboBox_SelectedIndexChanged(object
                              sender, System.EventArgs e)
{
 switch(cursorComboBox.Text)
 {
    case"Alapértelmezett":{Cursor=Cursors.Default;break;}
    case "Homokóra":{Cursor=Cursors.WaitCursor; break;}
    case "Kereszt": {Cursor = Cursors.Cross;break;}
    case "Kéz":{Cursor = Cursors.Hand;break;}
    case "Súgó":{Cursor = Cursors.Help;break;}
 }
}
```


Az elégedett Ön a fizetésével feladatban további kurzorkezelési lehetőségekkel is találkozhatunk.

3.1.7. Az ablak mérete beolvasással

Nyomógomb:

Name: sizeButton

Text: Ablakméret

Solution View

Lo project jobbegér

Add

Add Windows Form

Templates: Windows Form

Name: SizeForm.cs

Open

Properties, Text: Ablakméret beállítása

3. Gyakorlat. 4. ábra A méretet beolvasó ablak a szerkesztőben

A szövegmezők nevei a kiírt szövegben olvashatók. A vezérlők legyenek public hozzáférésűek, hogy kívülről állíthassuk értéküket!

Tegyünk egy OK és egy Mégse gombot is az ablakba!

Az OK gomb DialogResult értéke legyen OK, a mégsem gombé CANCEL! A SizeForm AcceptButton tulajdonsága legyen az OKButton, így Enter hatására az OK gomb DialogResult értéke állítódik be. A CancelButton tulajdonságot állítsuk a mégsem gombhoz tartozó névre, így Esc hatására a Mégsem gombhoz rendelt eseménykezelő hívódik meg.

```
private void sizeButton_Click(object sender,
                                    System.EventArgs e)
{
 SizeForm form = new SizeForm();
 form.xTextBox.Text=Left.ToString();
 form.yTextBox.Text=Top.ToString();
 form.widthTextBox.Text=Width.ToString();
 form.heightTextBox.Text=Height.ToString();
 form.ShowDialog();
 if (form.DialogResult==DialogResult.OK)
 {
   Left=Convert.ToInt32(form.xTextBox.Text);
   Top=Convert.ToInt32(form.yTextBox.Text);
   Width=Convert.ToInt32(form.widthTextBox.Text);
   Height=Convert.ToInt32(form.heightTextBox.Text);
 }
}
```

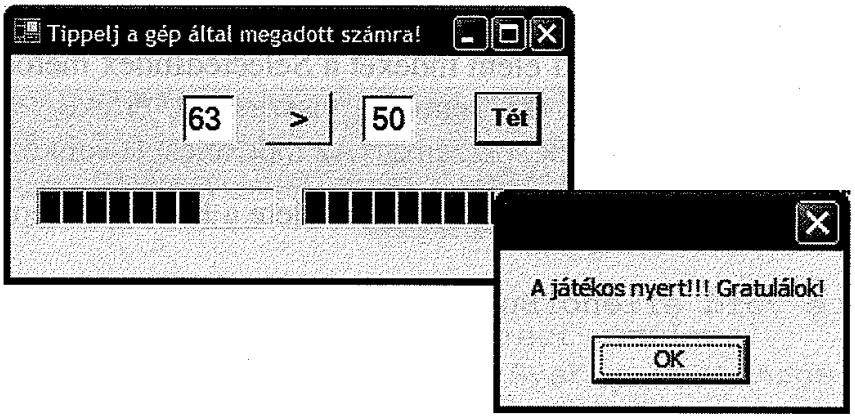
3.2. Feladatok

1. A jelszóbekérő alkalmazásunkat oldjuk meg úgy, hogy a felhasználók nevét egy comboboxból lehessen kiválasztani! Helyes jelszó esetén nyíljon ki a főablak, három hibás jelszó esetén lépjünk ki az alkalmazásból!*

2. **Csúszka és folyamatkijelző.** Tegyünk az ablakra egy csúszkát és egy folyamatkijelzőt! Mindkettő azonos értékhatárokat mutasson! A csúszkát mozgatva a folyamatkijelző is mutassa a beállított értéket! Együtt mozogjanak!*

3. Egy utazási iroda az utazásra vonatkozó adatokat kér. Ha repülőgéppel utazunk, kérdezze meg, hogy business vagy turista osztályon utazik-e, ha vonattal, akkor kér-e hálókocsit és helyjegyet, busz és saját személygépkocsi esetén nem kérdezünk semmit.

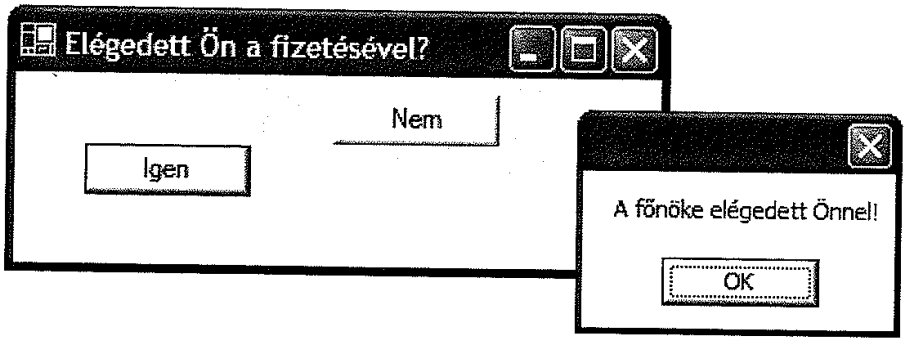
4. **Tipp.** A feladat 1 és 100 közötti véletlen számra megtippelni, hogy 50-nél nagyobb lesz-e. A két szám közti gombra kattintva az egérrel változik a (<; =, >) felirat. Helyes tipp esetén a játékos folyamatkijelzője, hibás tipp esetén a gép folyamatkijelzője növekszik. Aki előbb eléri a 10 pontot, az nyerte a játékot.*



5. **Kő, papír olló játék.** A játékos kijelöl, a gép véletlenszám-generátorral választ. A játék állását folyamatkijelző mutatja.

6. **Elégedett Ön a fizetésével?** A kérdés az ablak címsora, és az ablak két gombot tartalmaz, az egyiken 'Igen', a másikon 'Nem' felirat. Ha az egeret a nem gomb fölé visszük, hogy kiválasszuk, az elugrik egy véletlen helyre. Így csak az 'Igen' gomb lesz választható, ami után egy párbeszédablak a következő szöveget írja ki: A főnöke elégedett Önnel!*

   A feladat megoldható 'Szereted a csokit?' kérdéssel, és itt csak a 'Nem' válasz választható, és a 'A fogorvosod elégedett Veled!' felirat a befejezés.



Más megvalósítás: Ne engedjük a kurzorunkat az 'Igen' gombról elmozdulni!
Más megvalósítás: Ne a gombot, az egeret dobjuk véletlen helyre!

7. **Számológép.** Készítsünk egy számológépet, először csak a négy alapművelet elvégzésére, aztán lehet továbbfejleszteni!

3.3. Megoldásötletek

1. **Válasszuk combo-ból a jelszóbekérő ablak felhasználóját!**

   A combo neve: userComboBox, Text tulajdonsága üres, DropDownStyle tulajdonsága DropDownList (lehet DropDown is, ha az új user bevitelét is itt akarjuk később megvalósítani). Az items collection ablakába vigyük fel a felhasználókat! A kiválasztott elem indexét a SelectedIndex metódus adja vissza. Ha még nem választottunk ki semmit, visszaadott értéke -1, egyébként a collection megfelelő elemének sorszáma. (Az indexelést 0-val kezdjük!)

   Az ellenorButtonClick függvény fenti kódrészlete a következőre módosul:

```
private void ellenorButton_Click(object sender,
                                 System.EventArgs e)
{
    if (comboBox1.SelectedIndex>-1 &&
    jelszoBox.Text == jelszo[comboBox1.SelectedIndex])
        // Helyes jelszó.
        label.Text = "Helyes jelszó!";
    else      // Hibás jelszó...
}
```

   Helyes jelszó estén jelenjen meg a főablak! Ehhez először létre kell hoznunk a főablakot!

Solution Explorer

Jelszo namespace jobbegér

Add

Add Windows Form

Windows Form

Name: MainForm

Open

Itt már szerencsére nincs szükség az átnevezésekre.

A Main metódust kell módosítanunk, hogy mielőtt elindul az alkalmazás, kérje be a jelszót! A jelszóbekérő ablakot modálisan nyissuk ki (ShowDialog)! Ez azt jelenti, hogy amíg be nem zártuk, addig az alkalmazás többi ablaka nem kap fókuszt.

A modális ablaknak van visszatérési értéke. Ez a DialogResult tulajdonságban tárolódik. Ezt az értéket be kell állítanunk helyes jelszó és 3 rossz jelszó esetén.

```
static void Main()
{
 JelszoForm jFrm = new JelszoForm();
 jFrm.ShowDialog();
 if (jFrm.DialogResult==DialogResult.OK)
    Application.Run(new MainForm());
}

private void OKButton_Click(object sender,
                                System.EventArgs e)
{
 ...
 int i= userComboBox.SelectedIndex;
 if (i!=-1)
 {
    if (jelszoBox.Text== jelszo[i])
    {
        label.Text="Helyes jelszó";
            DialogResult= DialogResult.OK;
            this.Close();
    }
```

```
        else
        {
            label.Text="Hibás jelszó";
            if (++missed==3)
            {
                  label.Refresh();
                  System.Threading.Thread.Sleep(1000);
                  //Application.Exit();
                  DialogResult= DialogResult.Cancel;
                  Close();
            }
    ...
}
```

2. Csúszka és folyamatkijelző

```
private void trackBar1_Scroll(object sender,
                                        System.EventArgs e)
{
 progressBar.Value=trackBar.Value;
}
```

4. Tipp

Minden kattintásra változzon a gomb felirata és a tipp tagváltozó értéke!

```
private int tipp=1;

private void tipButton_Click(object sender,
                                        System.EventArgs e)
{
 switch (tipp)
 {
    case 1:  tipp--; tipButton.Text="="; break;
    case 0:  tipp--; tipButton.Text="<"; break;
    case -1: tipp=1; tipButton.Text=">"; break;
 }
}

private Random rand;
private int randNum;

private void tetButton_Click(object sender,
                                        System.EventArgs e)
{
 randNum = rand.Next(100);
 gepTextBox.Text=randNum.ToString();
```

```
if ((randNum < Convert.ToInt32(textBox50.Text))&&
                                (tipButton.Text == "<")
   ||(randNum == Convert.ToInt32(textBox50.Text))&&
                                (tipButton.Text == "=")
   ||(randNum > Convert.ToInt32(textBox50.Text))&&
                                (tipButton.Text == ">"))
   userProgressBar.Value++;
else
   gepProgressBar.Value++;
if (userProgressBar.Value==10 )
{
   MessageBox.Show("A játékos nyert!!! Gratulálok!");
   userProgressBar.Value = 0;
   gepProgressBar.Value = 0;
}
else if (gepProgressBar.Value==10)
{
   MessageBox.Show("A gép nyert. Legközelebb sikerülni
                                                    fog!");
   userProgressBar.Value = 0;
   gepProgressBar.Value = 0;
}
}
```

6. **Elégedett Ön a fizetésével?**

Ügyeljünk rá, hogy a Nem gomb legyen felül!

```
public FizForm()
{
 InitializeComponent();
 rand = new Random();
}

private void igenButton_Click(object sender,
                                    System.EventArgs e)
{
 MessageBox.Show("A főnöke elégedett Önnel!");
}

private Random rand;
```

```
private void nemButton_MouseMove(object sender,
                                 System.EventArgs e)
{
 nemButton.Left=rand.Next(
                  ClientSize.Width-nemButton.Width);
 nemButton.Top=rand.Next(
                  ClientSize.Height-nemButton.Height);
}
```

Megjegyzés:

A MouseEnter is jó lenne, csak ha véletlenül úgy indul az alkalmazás, hogy az egér épp a 'Nem' gomb fölött van, akkor nem ugrik el. Ha azt akarjuk, hogy a Tab gombbal se lehessen fölé mozdulni, állítsuk hamisra a TabStop tulajdonságot!

Más megvalósítás: Ne engedjük a kurzort az 'Igen' gombról elmozdítani!

A **Cursor Clip** tulajdonsága adja meg a téglalapot amibe a kurzort bezártuk. A téglalapot képernyő koordinátákban kell megadni (**RectangleToScreen**)! Mivel csak olyan ablakra lehet beállítani, amelynek a Cursor üzenetek mennek, az ablak aktiválásakor tudjuk beállítani.

Megjegyzés:

A tesztelés során, Alt F4-gyel bezárható az ablak, ha a kurzort nem tudjuk kimozdítani.

```
privateRectangle fullScreenRect;

private void FizForm_Closing(object sender,
             System.ComponentModel.CancelEventArgs e)
{
 Cursor.Clip = fullScreenRect;
}

private void FizForm_Load(object sender,
                                  System.EventArgs e)
{
 fullScreenRect = Cursor.Clip;
}
```

```
private void FizForm_Activated(object sender,
                                        System.EventArgs e)
{
 Cursor.Clip = igenButton.RectangleToScreen(
             new Rectangle( 0,0, igenButton.Size.Width,
                                  igenButton.Size.Height));
}

private void igenButton_Click(object sender,
                                        System.EventArgs e)
{
 MessageBox.Show("A főnöke elégedett Önnel!");
 Cursor.Clip=fullScreenRect;
}
```

Más megvalósítás: Ne a gombot, az egeret dobjuk véletlen helyre!

```
private void nemButton_MouseMove(object sender,
             System.Windows.Forms.MouseEventArgs e)
{
//  nemButton.Left=rand.Next(ClientSize.Width-
//nemButton.Width);
//  nemButton.Top=rand.Next(ClientSize.Height-
//nemButton.Height);
 Cursor.Position = new Point(
                   rand.Next(fullScreenRect.Width),
                   rand.Next(fullScreenRect.Height));
}
```

4. Gyakorlat. Programozási tételek

Generáljunk egy új Windows Applicationt Tetelek néven! A Formját nevezzük TetelekFormnak! Az egyes tételcsoportok bemutatásához használjunk TabControlt! A tabControl TabPages tulajdonságát töltsük föl elemekkel!

4.1. A csere algoritmusa

TabPage Collection Editor

Add

Name: csereTabPage

Text: Csere

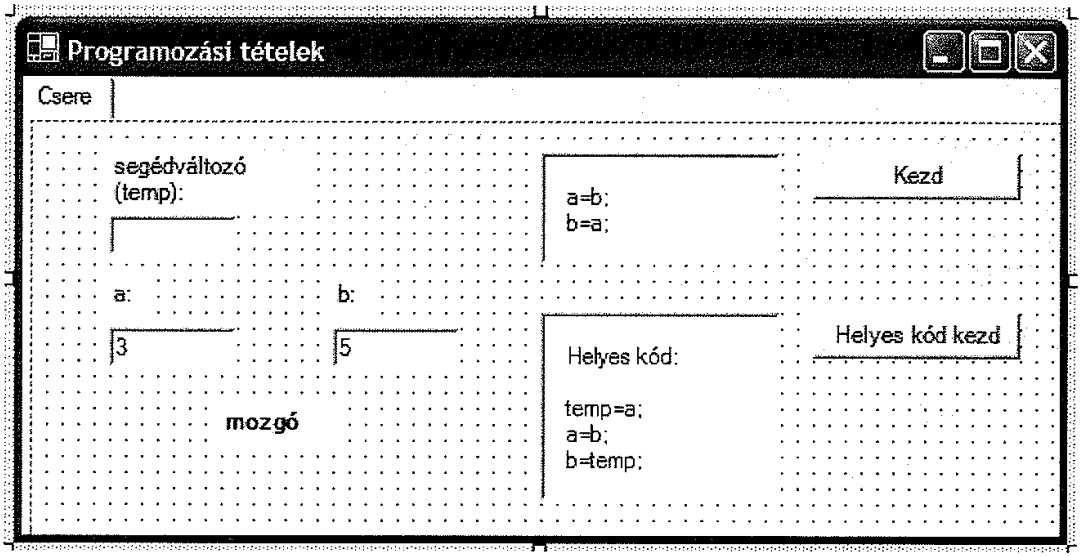
A csere, mint nevéből is kiderül, két változó értékét hivatott kicserélni.

A feladatban be kívánjuk mutatni, hogy a csere során szükségünk van segédváltozóra. A hibás és helyes kódot egy-egy listadobozban tároljuk. A mozgó feladata az adatmozgás bemutatása. Készítsük el a következő felületet a tervezőben!

	Name	Text	Visible	Items
TextBox	aTextBox, bTextBox	3, 5,		
Label	aLabel, bLabel,	a:, b:		

	Name	Text	Visible	Items
TextBox	tempTextBox		False	
Label	tempLabel	segédváltozó (temp):	False	
Label	textLabel	mozgó	False	
ListBox	failedCodeListBox		False	a=b; b=a;
ListBox	validCodeListBox		False	Helyes kód: temp=a; a=b; b=temp;
Button	startButton	Kezd		
Button	helyesButton	Helyes kód kezd	False	

A textLabel BackColor tulajdonságát állítsuk White-ra, ForeColor tulajdonságát pirosra és a fontját félkövérre (Bold).



4. Gyakorlat. 1. ábra A csere fül felhasználói felülete a tervezőben

Ügyeljünk a vezérlők egymás fölötti elhelyezésére! (A textLabelt hozzuk legfelülre!)

Futásidőben csak az a, b változók és a Kezd gomb látható induláskor. Írjuk meg a kezd választás kódját! Ne végezzen valóságos cserét, mert akkor elveszítjük 'a' értékét!

4.1.1. Hibás csere

```
private void startButton_Click(object sender,
                                        System.EventArgs e)
{
 failedCodeListBox.Visible=true;
 failedCodeListBox.SelectedIndex=1;

 //Kiteszi a textLabelt.
 textLabel.Left=bTextBox.Left;
 textLabel.Top=bTextBox.Top;
 textLabel.Text=bTextBox.Text;
 textLabel.Visible=true;
 System.Threading.Thread.Sleep(300);
 //Mozgatja a textLabelt a-ra.
 while(textLabel.Left!=aTextBox.Left)
 {
    textLabel.Left--;
    System.Threading.Thread.Sleep(20);
 }
 failedCodeListBox.SelectedIndex++;
 System.Threading.Thread.Sleep(500);
 failedCodeListBox.Items.Insert(0,"Hibás kód!");
 failedCodeListBox.BackColor=Color.Red;
 helyesButton.Visible=true;
}
```

Fordítás/Futtatás/Tesztelés

Amikor a textLabel leér a bTextBoxról – annak ablaka üres.

Ha újraindítom: az 'a' üres, a 'Hibás kód' szöveget újra és újra beírja a listadobozba. Célszerű lenne a gomb 'Kezd' szövegét 'Hibás kód kezd'-re változtatni, ha már tudjuk, hogy ez így hibás.

Az ablak felületének frissítése egy windows üzenet kezelését jelenti (méghozzá pihenőidős tevékenységként). Amíg egy függvényünk kódjának végrehajtását végezzük, nem lenne jó, ha kevésbé fontos (pihenőidős) tevékenységek vennék el a végrehajtástól az időt. Itt viszont mi azonnal szeretnénk látni a megjelenített labelt, vagy a mögötte immár látszó szövegmezőt, ezért magunknak kell a Refresh() függvényhívással kikényszeríteni a frissítést.

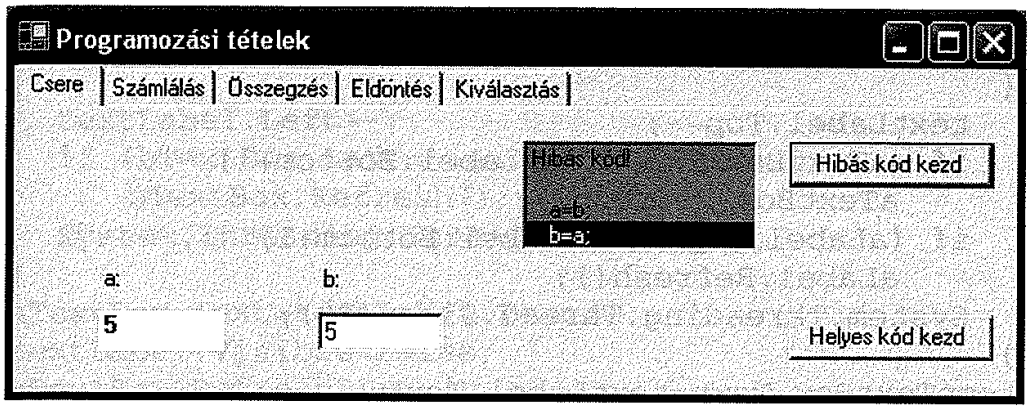
A hibás kódot tartalmazó listadoboz nem baj, ha piros hátterű marad, és most már választhatunk a hibás vagy helyes kód végrehajtása között, hogy összehasonlíthatók legyenek.

A kód a módosítások után:

```
private void startButton_Click(object sender,
                                    System.EventArgs e)
{
 textLabel.Visible=false;      //Az újraindítás miatt.
 aTextBox.Refresh();           //Az újraindítás miatt.
 failedCodeListBox.Visible=true;
 if (!failedCodeListBox.Items.Contains("Hibás kód!"))
    failedCodeListBox.SelectedIndex=1;
 else
    failedCodeListBox.SelectedIndex=2;

 //Kiteszi a textLabelt.
 textLabel.Left=bTextBox.Left;
 textLabel.Top=bTextBox.Top;
 textLabel.Text=bTextBox.Text;
 textLabel.Visible=true;
 textLabel.Refresh();
 System.Threading.Thread.Sleep(300);
 //Mozgatja a textLabelt a-ra.
 while(textLabel.Left!=aTextBox.Left)
 {
    textLabel.Left--;
    if (bTextBox.Left<textLabel.Right)
       bTextBox.Refresh();
    System.Threading.Thread.Sleep(20);
 }
 failedCodeListBox.SelectedIndex++;
 System.Threading.Thread.Sleep(500);
 // Eldöntés.
 if (!failedCodeListBox.Items.Contains("Hibás kód!"))
    failedCodeListBox.Items.Insert(0,"Hibás kód!");
 failedCodeListBox.BackColor=Color.Red;
 helyesButton.Visible=true;
 startButton.Text="Hibás kód kezd";
}
```

Fordítás/Futtatás/Tesztelés

Mivel a függvény végén nem cseréljük ki az aTextBox és a fölé mozgatott textLabel szövegét, hanem fölötte látni hagyjuk pirossal a hibás értéket, ezért minden újrakezdéskor az aTextBox tartalmát frissíteni kell, hogy kezdetben látható legyen.



4. Gyakorlat. 2. ábra A csere hibás kódja futás után

4.1.2. A helyes csere

Kezeljük a 'Helyes kód kezd' gombon kattintás eseményt! Előbb csak a temp=a-t!

```
private void helyesButton_Click(object sender,
                                System.EventArgs e)
{
 aTextBox.Refresh();
 validCodeListBox.Visible=true;
 validCodeListBox.Refresh();
 validCodeListBox.SelectedIndex=3;

 //temp=a;
 tempLabel.Visible=true;
 tempLabel.Refresh();
 tempTextBox.Visible=true;
 tempTextBox.Refresh();

 //Kiteszi a textLabelt.
 textLabel.Left=aTextBox.Left;
 textLabel.Top=aTextBox.Top;
 textLabel.Text=aTextBox.Text;
 textLabel.Visible=true;
 textLabel.Refresh();
 System.Threading.Thread.Sleep(300);
```

```
    //Mozgatja a textLabelt tempre.
    while(textLabel.Top!=tempTextBox.Top)
    {
       textLabel.Top--;
       if (aTextBox.Top==textLabel.Bottom+1)
          aTextBox.Refresh();
       if (aLabel.Top==textLabel.Bottom+1)
          aLabel.Refresh();
       System.Threading.Thread.Sleep(20);
    }
    tempTextBox.Text=textLabel.Text;
    textLabel.Visible=false;
    tempTextBox.Refresh();
}
```

Fordítás/Futtatás/Tesztelés

Itt nemcsak az aTextBox, hanem az aLabel fölött is elmegy a textLabel, így amikor elhagyja, annak megjelenését is frissítenünk kell!

A következő kódrészlet szinte a fenti hibás kóddal azonos, csak itt már megtörténhet az értékadás, mert 'a' értékét elmentettük temp-be. Írjuk a tempBox.Refresh() után:

```
private void helyesButton_Click(object sender,
                                System.EventArgs e)
{…
 tempTextBox.Refresh();

 //a=b
 validCodeListBox.SelectedIndex++;
 System.Threading.Thread.Sleep(500);

 //Kiteszi a textLabelt b-re.
 textLabel.Left=bTextBox.Left;
 textLabel.Top=bTextBox.Top;
 textLabel.Text=bTextBox.Text;
 textLabel.Visible=true;
 textLabel.Refresh();
 System.Threading.Thread.Sleep(300);
```

```
//Mozgatja b-ből a-ba.
while(textLabel.Left!=aTextBox.Left)
{
   textLabel.Left--;
   if (bTextBox.Left==textLabel.Right+1)
      bTextBox.Refresh();
   System.Threading.Thread.Sleep(20);
}
aTextBox.Text=textLabel.Text;
textLabel.Visible=false;
aTextBox.Refresh();
}
```

Fordítás/Futtatás/Tesztelés

Már csak a temp tartalmát kell b-be betenni.

```
private void helyesButton_Click(object sender,
                                        System.EventArgs e)
{…
 aTextBox.Refresh();

 //b=temp;
 validCodeListBox.SelectedIndex++;
 System.Threading.Thread.Sleep(500);
 //Kiteszi a textLabelt temp-re.
 textLabel.Left=tempTextBox.Left;
 textLabel.Top=tempTextBox.Top;
 textLabel.Text=tempTextBox.Text;
 textLabel.Visible=true;
 textLabel.Refresh();
 System.Threading.Thread.Sleep(300);

 //Mozgatja temp-ből b-be.
 while(textLabel.Left!=bTextBox.Left)
 {
    textLabel.Left+=2;
    textLabel.Top++;
    if (tempTextBox.Left==textLabel.Left-1)
       tempTextBox.Refresh();
    System.Threading.Thread.Sleep(20);
 }
 bTextBox.Text=textLabel.Text;
 textLabel.Visible=false;
 bTextBox.Refresh();
```

```
    //Eltüntetjük a segédváltozót.
    tempTextBox.Text=""; // Újraindítás miatt.
    tempLabel.Visible=false;
    tempTextBox.Visible=false;
  }
```

Fordítás/Futtatás/Tesztelés

Bár a három kód más-más adatokat cserélt ki, más irányú mozgást végez, és más vezérlők fölött halad el – az olvasó is nyilván másolta és módosította a meglevő kódrészletet –, a kódismétlés elkerülése végett jó lenne egy általános függvényt írni az adatmozgatás megjelenítésére, mely függvényt aztán paraméterezve hívhatnánk!

Class View

TetelekForm class jobbegér

Add

Add Method

Method name: Mozgat

Parameter type: Control

Parameter name: mit

Add

Parameter type: Control

Parameter name: hova

Add

Parameter type: Control

Parameter name: at

Add

Finish.

```
public void Mozgat(Control mit, Control hova, Control at)
{
  //Kiteszi a textLabelt mit-re.
  textLabel.Left=mit.Left;
  textLabel.Top=mit.Top;
  textLabel.Text=mit.Text;
  textLabel.Visible=true;
  textLabel.Refresh();
  System.Threading.Thread.Sleep(300);
```

```
    //Kiszámítja a mozgatás egységét
    int x=0;
    if (mit.Left!=hova.Left)
      x=(hova.Left-mit.Left)/50;
    int y=0;
    if (mit.Top!=hova.Top)
      y=(hova.Top-mit.Top)/50;

    //Mozgatja mit-ből hova-ba.
    while(textLabel.Left!=hova.Left ||
                                  textLabel.Top!=hova.Top)
    {
      textLabel.Left+=x;
      textLabel.Top+=y;
      //Egyszerűbb frissiteni, mint kiértékelni mikor kell.
      mit.Refresh();
      at.Refresh();
      System.Threading.Thread.Sleep(20);
    }

    // Hova megkapja az új értéket.
    hova.Text=textLabel.Text;
    textLabel.Visible=false;
    hova.Refresh();
  }
```

A Mozgat függvényt nem tudjuk meghívni a hibás kódból, mert a végén ott a 'hova' paraméternek átadott érték. De a jó kódrészletből háromszor is hívható. A tételek demonstrálása során újra használható lesz. A hívások kódja:

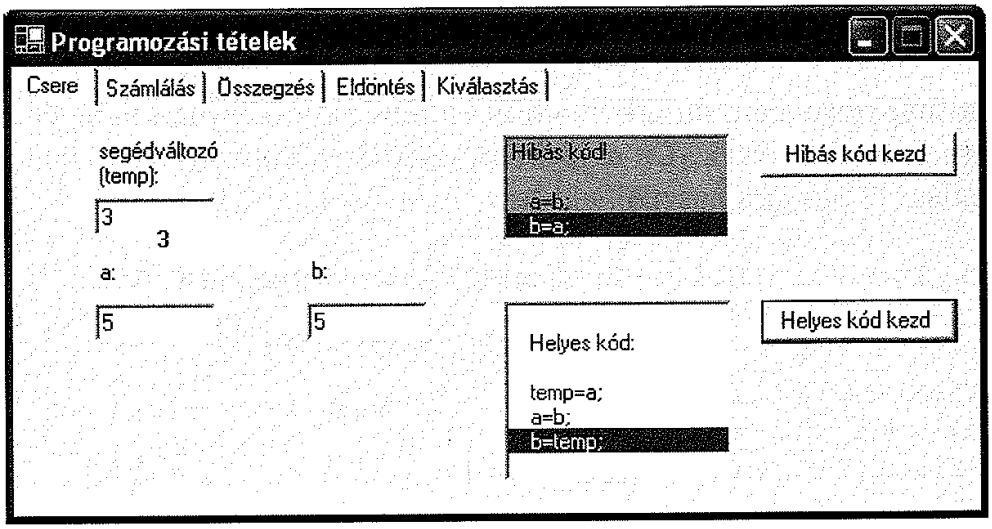
```
  private void helyesButton_Click(object sender,
                                        System.EventArgs e)
  {
    aTextBox.Refresh();
    validCodeListBox.Visible=true;
    validCodeListBox.Refresh();
    validCodeListBox.SelectedIndex=3;

    //temp=a;
    tempLabel.Visible=true;
    tempLabel.Refresh();
    tempTextBox.Visible=true;
    tempTextBox.Refresh();
    Mozgat(aTextBox, tempTextBox, aLabel);
```

```
  //a=b
  validCodeListBox.SelectedIndex++;
  System.Threading.Thread.Sleep(500);
  Mozgat(bTextBox,aTextBox,bTextBox);

  //b=temp;
  validCodeListBox.SelectedIndex++;
  System.Threading.Thread.Sleep(500);
  Mozgat(tempTextBox, bTextBox, bLabel);

  //Eltüntetjük a segédváltozót.
  tempTextBox.Text=""; // Újraindítás miatt.
  tempLabel.Visible=false;
  tempTextBox.Visible=false;
}
```

Fordítás/Futtatás/Tesztelés

Próbáljuk ki, hogy az 'a', 'b' változó értékének futásidejű változtatása (átgépelése) után is helyesen működik a program!

```
System.Threading.Thread.Sleep(1500);
```



4. Gyakorlat. 3. ábra A csere helyes kódja futás közben

4.1. Elemi programozási tételek

TabPage Collection Editor

Add

Name: countTabPage

Text: Számlálás

Add

Name: sumTabPage

Text: Összegzés

Add

Name: containsTabPage

Text: Eldöntés

Add

Name: findTabPage

Text: Kiválasztás

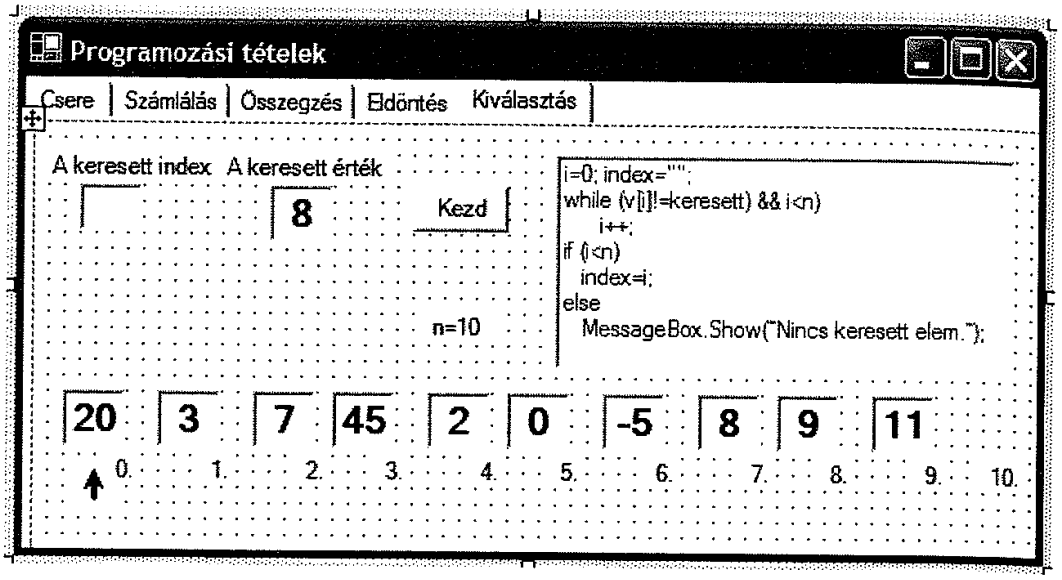
4.1.1. A kiválasztás tétele

Adjuk meg, hogy hányadik helyen található az adott sorozatban az első, adott feltételt teljesítő elem.

A feladatban a feltétel az adott értékkel egyenlő. Alapértelmezésben az adott érték legyen 8, mely a futtatás során módosítható!

A sorozat legyen 10 elemű, elemeinek van kezdőértéke, hogy ne kelljen mindig feltölteni értékekkel, de bármely elem módosítható.

A fenti 11 értéket egy-egy szövegmezőben mutatjuk meg, és a keresett index tárolására is felveszünk egy szövegmezőt. Az index kezdetben legyen üres!



4. Gyakorlat. 4. ábra A kiválasztás tétel a szerkesztőben

Ahhoz, hogy bemutassuk, hol tartunk a keresésben, egy felfelé nyíl ikont fogunk használni. A nyíl rámutat a tömb éppen aktuális elemére. Az ikont a Microsoft Visual Studio .NET / Common7 / Graphics / Icons / arrows könyvtárában találjuk.

A szövegmezők fontját nagyobb méretűre választjuk (14) és félkövérre. A mezők alsó sarkába felírjuk az adott érték tömbindexét. Az indexelést eggyel tovább írjuk ki, hogy azt az esetet is bemutathassuk, amikor nem találja az elemet a sorozatban. A demonstrált kód egy listadobozban található.

A TetelekForm osztályba vegyünk fel egy az elemszámot tartalmazó 'n' mezőt!

ClassView

TetelekForm

Add

Add Field

Field access: private

Field type: int

Field name: n

Finish

Előkészítés: A felvitel sorrendjében a findTabPage oldal vezérlői indexet kapnak, mely a findTabPage.Controls.IndexOf(vezérlő neve) függvénnyel lekérdezhető. Ahhoz, hogy ciklusban tudjunk végigmenni a sorozat elemeit tartalmazó szövegmezőkön, célszerű az indexüket egymás után állítani az előkészítés során. A konstruktorban jelöljük meg a listadoboz 0. sorát, az első végrehajtható kódsort!

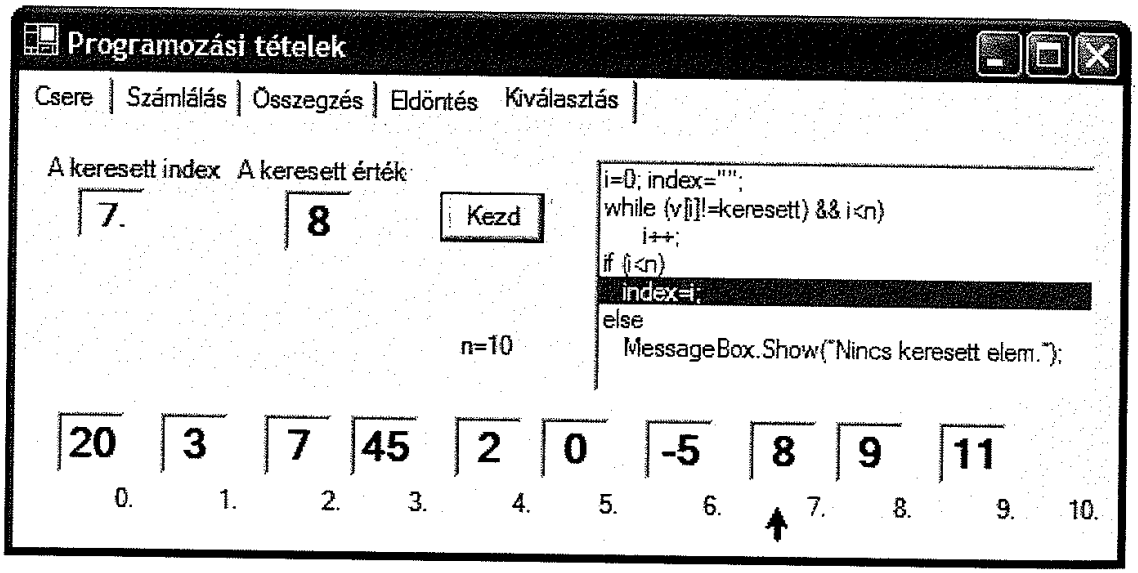
```
public TetelekForm()
{
 //
 // Required for Windows Form Designer support
 //
 InitializeComponent();

 //
 //TODO: Add any constructor code after
 //InitializeComponent call
 //
 // Adjuk meg a szövegmezőink indexét, hogy ciklusban
 //dolgozhassunk velük!
 findTabPage.Controls.SetChildIndex(textBox1,0);
 findTabPage.Controls.SetChildIndex(textBox2,1);
 findTabPage.Controls.SetChildIndex(textBox3,2);
 findTabPage.Controls.SetChildIndex(textBox4,3);
 findTabPage.Controls.SetChildIndex(textBox5,4);
 findTabPage.Controls.SetChildIndex(textBox6,5);
 findTabPage.Controls.SetChildIndex(textBox7,6);
 findTabPage.Controls.SetChildIndex(textBox8,7);
 findTabPage.Controls.SetChildIndex(textBox9,8);
 findTabPage.Controls.SetChildIndex(textBox10,9);

 //A kód első sora.
 kivalasztListBox.SelectedIndex=0;
}
```

Először oldjuk meg 10 elemre a feladatot! Az index mező kiürítése, és a nyíl első elemre állítása a függvény újrahívásakor fontos.

```
private void kezdButton_Click(object sender,
                                    System.EventArgs e)
{
 indexTextBox.Text="";
 indexTextBox.Refresh();
 n=10;
 int i=0;
 kivalasztListBox.SelectedIndex=1;
 arrowBox.Left=findTabPage.Controls[i].Left+5;
 Thread.Sleep(400);
```

```
    while (valueTextBox.Text != findTabPage.Controls[i].Text
                                                    && i<n)
    {
       kivalasztListBox.SelectedIndex=2;
       i++;
       arrowBox.Left=findTabPage.Controls[i].Left+5;
       Thread.Sleep(400);
       kivalasztListBox.SelectedIndex=1;
       Thread.Sleep(400);
    }
    if (i<n)
    {
       kivalasztListBox.SelectedIndex = 3;
       Thread.Sleep(1000);
       kivalasztListBox.SelectedIndex = 4;
       indexTextBox.Text = i.ToString()+".";
    }
    else
       MessageBox.Show("Nincs '"+valueTextBox.Text+"' értékű
                                    elem a sorozatban.");
}
```

Fordítás/Futtatás/Tesztelés.



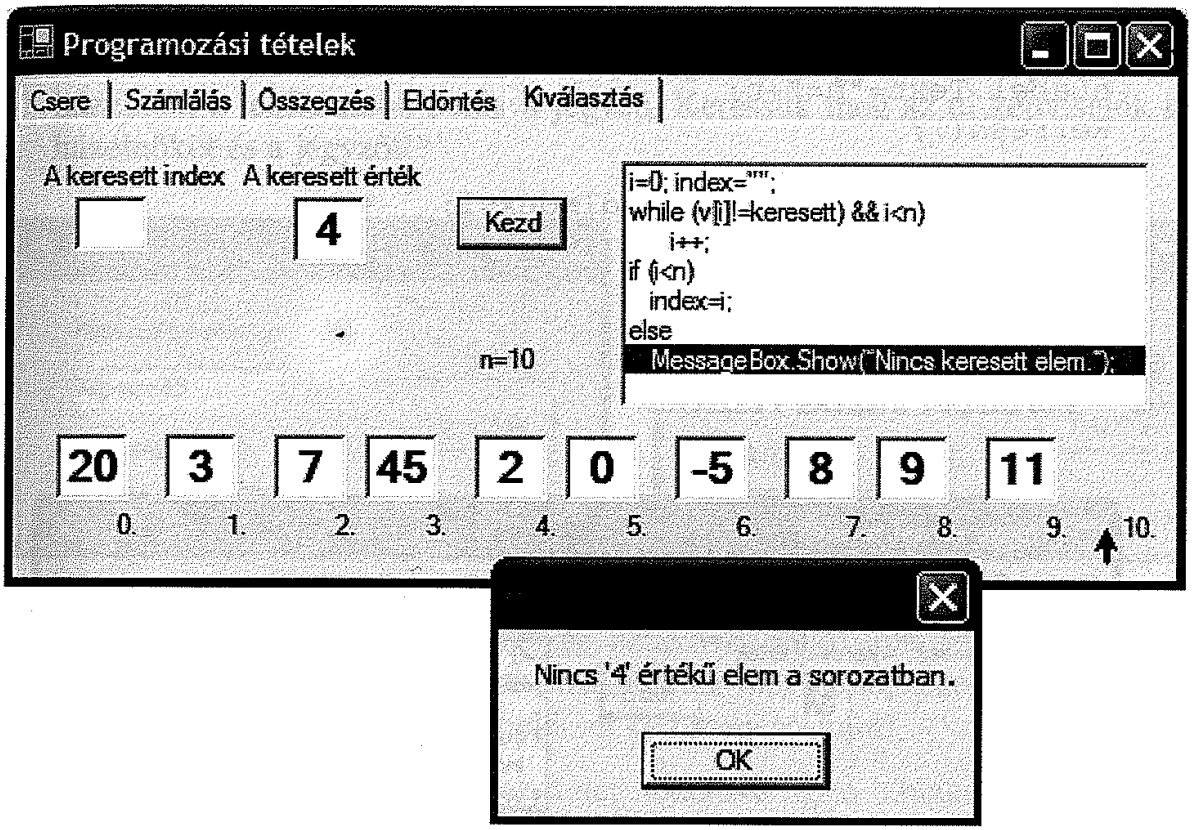
4. Gyakorlat. 5. ábra A keresett elem a sorozatban van

Ha a keresett elem megtalálható a sorozatban, a program jól működik. Módosítás és újraindítás után is. Ha a keresett elem nincs a sorozatban, az algoritmus akkor is helyes eredményt ad, azonban a nyíl nem a következő üres helyre mutat, hanem annak a vezérlőnek az oszlopába ugrik, amelyik a következő a vezérlők listáján. Tehát szükségünk van egy olyan szövegmezőre, mely a következő helyen áll a

sorban, csak nem látszik. Ha ennek indexét a konstruktorban beállítjuk a következő értékre, akkor erre a láthatatlan TextBoxra fog rámutatni a nyíl. És ez a cél.

```
public TetelekForm()
{...
 findTabPage.Controls.SetChildIndex(textBox10,9);
 findTabPage.Controls.SetChildIndex(textBox11,10);

 //A kód első sora.
 kivalasztListBox.SelectedIndex=0;
}
```

Fordítás/Futtatás/Tesztelés



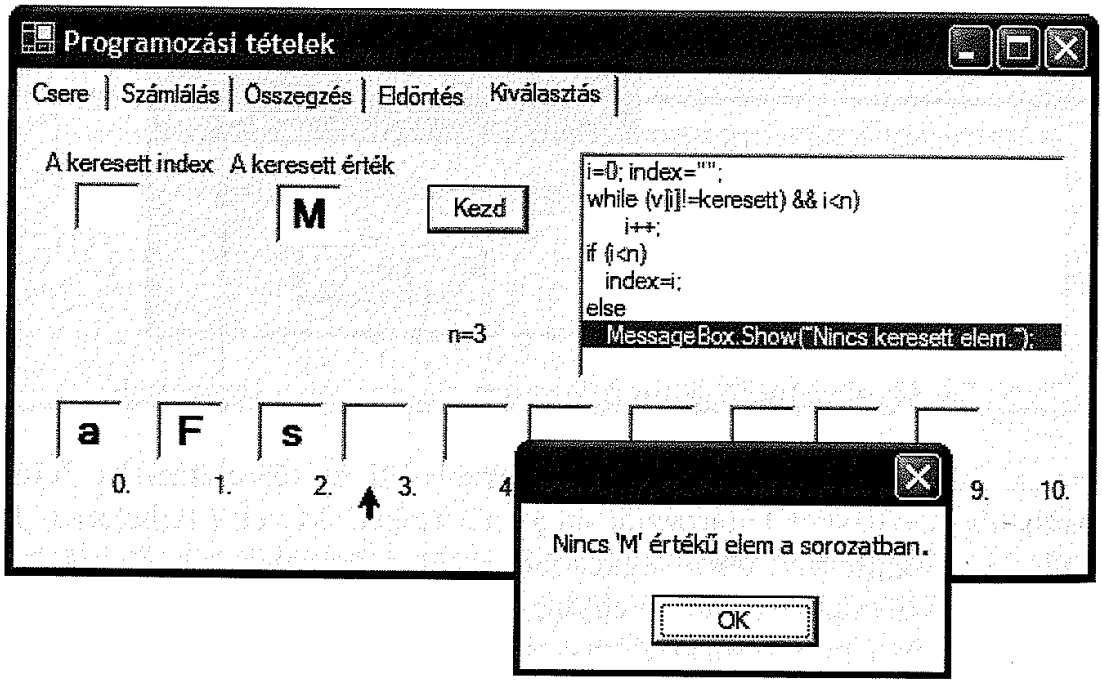
4. Gyakorlat. 6. ábra A keresett elem nincs a sorozatban

A sorozat végét az első üres mező jelezze! Mivel az elemek a futás során módosíthatók, kezdéskor határozzuk meg 'n' értékét! Mivel a láthatatlan TextBox szövegét üresre állítottuk, tehát legkésőbb annál, leáll a keresés. Ekkor 'n' értéke épp az üres értékű elemre mutat. Vegyük figyelembe, hogy 'n' nem az utolsó elem indexét jelenti, hanem a sorozat elemszámát, ami épp megegyezik az utolsó utáni elem indexével, mivel az indexelést mindig 0-val kezdjük!

Az üres mező utáni értékeket töröljük ki! Vegyük észre, hogy az elemszám meghatározása az első üres mező keresését jelenti! Tehát ez is egy lineáris keresés algoritmus.

```
private void kezdButton_Click(object sender,
                                System.EventArgs e)
{
 indexTextBox.Text="";
 indexTextBox.Refresh();

 //Elemszám az első üres elemig.
 n=0;
 while (findTabPage.Controls[n].Text!="")
   n++;
 //Az üres mező utáni elemeket kitöröljük.
 for (int j=n; j<10; j++)
   findTabPage.Controls[j].Text="";
 nLabel.Text="n="+n;
 Refresh();
 int i=0;
 kivalasztListBox.SelectedIndex=1;
 ...
}
```

Fordítás/Futtatás/Tesztelés



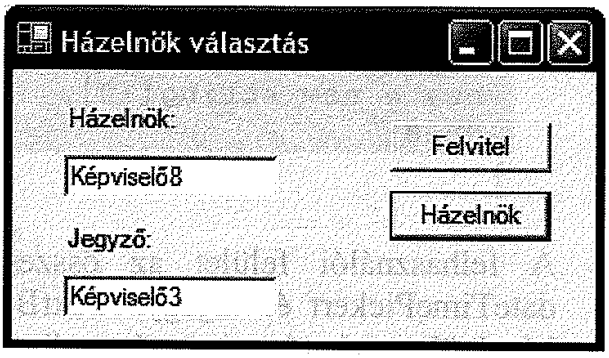
4. Gyakorlat. 7. ábra A keresett elem nincs a rövidebb sorozatban

Az üres mezőket, és az elsőt kivéve a hozzájuk tartozó indexet láthatatlanná is tehetnénk, azonban akkor, ha több elemmel kívánunk próbálkozni a következő tesztben, nem áll módunkban azokba értéket írni. Készíthetnénk egy 'új sorozat' gombot, ami láthatóvá tenné az összes mezőt, esetleg véletlen szám értékekkel fel is tölthetné azokat.

A feladat továbbfejlesztését az olvasóra bízzuk. Ez a feladat is jó példa arra, hogy minden programnál van általánosabb, felhasználóbarátabb, hatékonyabb, gyorsabb, esztétikusabb, testre szabhatóbb ... egyszóval még jobb program.

4.2. Feladatok

1. Demonstráljuk a többi tanult programozási tételt is!
2. **Parlament.** Az első parlamenti napon választják meg a házelnököt, így az újjáválasztott parlamentben nincs, aki a házelnöki teendőket ellássa e választás előtt. A házszabály szerint az első napon a legidősebb képviselő lesz a házelnök, a legfiatalabb pedig a jegyző. Keressük meg az új képviselők közt a házelnököt és a jegyzőt![1]



3. Készítsünk a jelszó módosítását lehetővé tevő ablakot, mely bekéri a felhasználó nevét, a régi jelszót, majd ha az megfelelő, lehetővé teszi új jelszó beírását, melyet ismételt begépeléssel kell megerősíteni!
4. Készítsünk új felhasználó felvitelét lehetővé tevő ablakot, mely bekéri a felhasználó nevét, ellenőrzi, hogy ilyen nevű felhasználónk még nincs, majd lehetővé teszi a jelszó beírását, melyet ismételt begépeléssel kell megerősíteni!

[1] Ötlet: Medzichradszky Dénes.

4.3. Megoldásötletek

4. Parlament

A feladatot maximum 10 képviselőre oldjuk meg, de a szám értelemszerűen a képviselők számához igazítható. Szükségünk lesz két tömbre, melyben a képviselők nevét és születési dátumát tároljuk, valamint egy db. változóra, mely megadja az eddig felvitt képviselők számát. A helyfoglalást és a kezdőértékadást a konstruktorban valósítjuk meg. A dátumot DateTimePicker vezérlővel visszük föl, ez biztosítja a valódi dátumfelvitelt, ellenőrzi, hogy az adott hónap hány napos. Az adatok begépelhetők (a hónapnál is számot vár), vagy kiválaszthatók a naptárból.

```
DateTime[] szuletes;
string[] nev;
int db;

public ParlamentForm()
{
 InitializeComponent();

 szuletes = new DateTime[10];
 nev = new string[10];
 db=0;
}
```

A felhasználói felület az összes vezérlőt tartalmazza, egymás fölött a dateTimePickert és a jegyzoTextBoxot is. Ez utóbbi kezdetben láthatatlan, a házelnök meghatározása után pedig a dateTimePicker a láthatatlan. A házelnököt feltöltés közben is meghatározhatjuk, de ha már minden képviselő adatát felvittük, akkor csak az eredmény megjelenítése lehetséges.

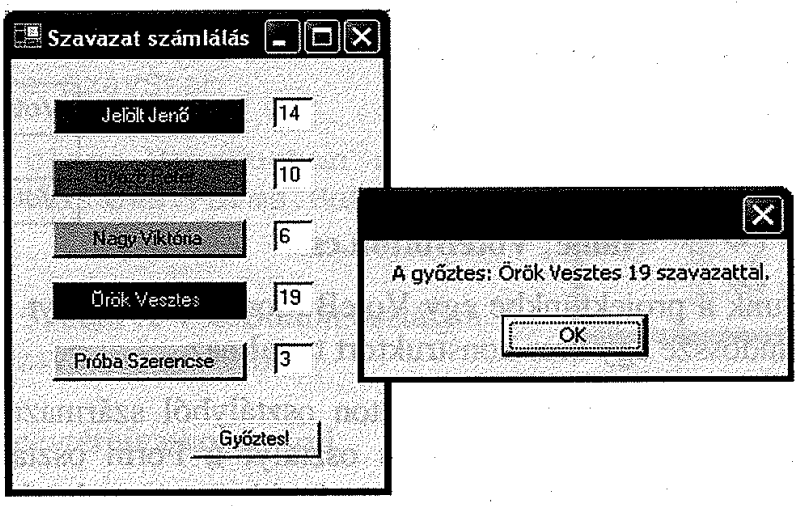
```
private void felvitelButton_Click(object sender,
                                  System.EventArgs e)
{
 // Ha még nem töltöttük fel,
 //de már megnéztük a házelnököt.
 jegyzoTextBox.Visible=false;
 dateTimePicker.Visible=true;
 label1.Text = "Név:";
 label2.Text= "Születési idő";
```

```
    //Felvitel.
    nev[db]=nevTextBox.Text;
    szuletes[db]=dateTimePicker.Value.Date;
    if (db<9)
      db++;
    else
    {
      felvitelButton.Visible = false;
      hazelnokButton.Visible = false;
      hazelnokButton_Click(sender,e);
    }
  }

  private void hazelnokButton_Click(object sender,
                                    System.EventArgs e)
  {
    if (db!=0)
    {
      int minInd=0;
      int maxInd=0;
      for (int i=1; i<db; i++)
      {
          if (szuletes[i]<szuletes[minInd])
                minInd=i;
          if (szuletes[i]>szuletes[maxInd])
                maxInd=i;
      }
      label1.Text = "Házelnök:";
      nevTextBox.Text = nev[minInd];
      label2.Text = "Jegyző:";
      dateTimePicker.Visible = false;
      jegyzoTextBox.Text = nev[maxInd];
      jegyzoTextBox.Visible = true;
    }
  }
```

5. Gyakorlat. Az öröklés és a .NET osztályai

5.1. Szavazatszámlálás

Készítsünk egy alkalmazást, mely a nyilvános szavazatszámlálást támogatja! A formon megjelenő gombok száma egyezzen meg a jelöltek számával, és feliratuk a jelöltek nevét tartalmazza! Minden név mellett egy szövegmező mutatja a kapott szavazatok számát. A végeredmény gomb választására egy üzenetablakban írjuk ki a győztes jelölt nevét és szavazatainak számát!



5. Gyakorlat. 1. ábra A kész alkalmazás

A szavazatszámlálás során a felolvasott szavazatot megkapó személy gombját megnyomjuk. A gomb egy árnyalattal sötétebb lesz, és a mellette levő számláló értéke eggyel növekszik.

5.1.1. A projekt felhasználói felülete

A feladat megoldásához generáljunk egy Windows alkalmazást SzavSzam néven! A form osztályunk neve legyen SzavSzamForm! Tegyük ki rá a vezérlőket!

Name	jejeButton	gyozoButton	nagyButton	vesztButton	probaButton
Text	Jelölt Jenő	Győző Péter	Nagy Viktória	Örök Vesztes	Próba Szerencse

Name	jejeTextBox	gyozoTextBox	nagyTextBox	vesztTextBox	probaTextBox
Text	0	0	0	0	0

Legyen még egy Győztes! feliratú eredmButton is!

5.1.2. A számlálás kijelzést színével támogató gomb

Nekünk olyan gombra lenne szükségünk, aminek a szavazatok növekedtével egyre sötétebb a színe, ezzel is jelezve, ki vezet a jelöltek közül.

Ezt a Button osztály nem szolgáltatja. Tehát szükségünk van egy olyan osztályra, amilyen a Button, de amelyik képes sötétíteni a színét. Tehát a Button osztályt szeretnénk a célnak megfelelően specializálni! A problémát örökléssel, a Button osztály VoteButton nevű utódosztályának létrehozásával oldhatjuk meg.

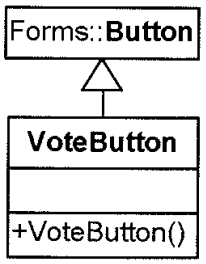
Solution View

SzavSzam projekt jobbegér

Add

Add Class

Name: VoteButton.cs



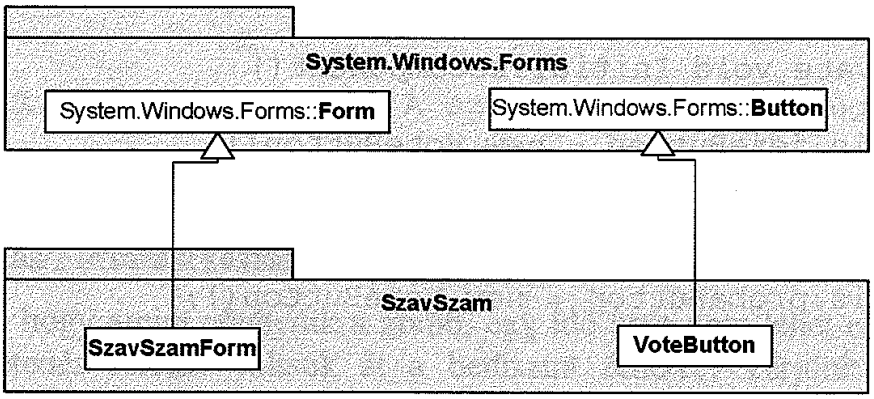
Ezzel létrehoztunk a projektünkbe egy VoteButton nevű osztályt. Ha megnézzük a forráskódot, mindössze egy üres konstruktort tartalmaz.

Azt szeretnénk, ha ez az osztály a Button osztályból származna! Nézzük meg, hogyan származtattuk a SzavSzamForm osztályt a Form osztályból! A Button osztály leírása is a System.Windows.Forms névtérben szerepel, tehát:

```
public class VoteButton : System.Windows.Forms.Button
```

A konstruktorban állítsuk a gomb színét fehérre! A fehér szín úgy jön létre, hogy a szín piros (red), zöld (green) és kék (blue) összetevőjét is maximálisra (255-re) állítjuk. A Color osztály FromArgb függvénye a három szín számértékéből összeállítja a színt.

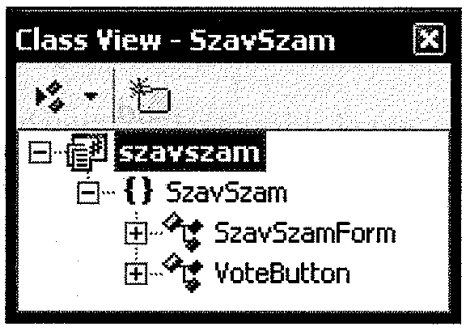
```
public VoteButton()
{
 this.BackColor =
            System.Drawing.Color.FromArgb(255,255,255);
}
```



5. Gyakorlat. 2. ábra Az alkalmazás névterei és osztályai

Használjuk ezt az osztályt, mint a gombot a SzavSzamForm felületén! Ezzel egyben tesztelhetjük a létrehozott osztályt. Az objektumok létrehozásakor Button helyett VoteButtont hozzunk létre! A VoteButton a SzavSzam névtérben lett definiálva, amiben épp most dolgozunk. Ez jól látható a kódból, de mutatja Class View ablak is.

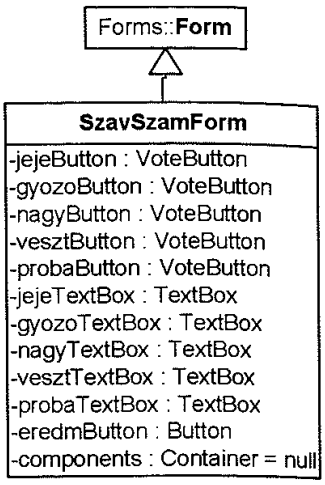
```
namespace SzavSzam
{
 public class VoteButton :
 System.Windows.Forms.Button
}
```



5. Gyakorlat. 3. ábra A VoteButton osztály a SzavSzam névtérben

Így nem kell elé névtérhivatkozást írni.

```
public class SzavSzamForm : System.Windows.Forms.Form
{
 private VoteButton jejeButton;
 private VoteButton gyozoButton;
 private VoteButton nagyButton;
 private VoteButton vesztButton;
 private VoteButton probaButton;
...
}
```



Természetesen az objektum osztályának megfelelő konstruktort kell hívnunk az InitializeComponent tagfüggvényében, ha nem ezt tesszük, a fordító figyelmeztet bennünket.

```
private void InitializeComponent()
{
 this.jejeButton = new VoteButton();
 this.gyozoButton = new VoteButton();
 this.nagyButton = new VoteButton();
 this.vesztButton = new VoteButton();
 this.probaButton = new VoteButton();
...
}
```

Fordítás/Futtatás

A gombok fehér színben jelennek meg az ablakban.

5.1.3. Új metódus a VoteButton osztályban

Class View

VoteButton jobbegér

Add

Add Method

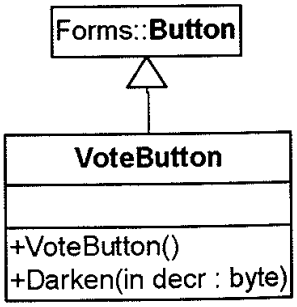
Method name: Darken

Parameter type: byte

Parameter name: decr

Add

Finish



A Color osztály R, G, B tulajdonságai visszaadják az aktuális szín összetevőinek értékét.

```
public void Darken(byte decr)
{
 if (BackColor.R>decr)
    BackColor = System.Drawing.Color.FromArgb(
    BackColor.R-decr,BackColor.G-decr,BackColor.B-decr);
}
```

A gombon való kattintás esemény kezelőjében nem kell mást tennünk, mint meghívni ezt a függvényt.

```
private void jejeButton_Click(object sender,
                                        System.EventArgs e)
{
 jejeButton.Darken(10); // A szavazók számától függően.
}
```

Fordítás/Futtatás/Tesztelés

Azt tapasztaljuk, hogy amint sötétedik a gomb, egy idő után nem lehet elolvasni a feliratát. A sötétedés felénél váltsuk át a betűszínt fehérre! Térjünk vissza a VoteButton osztály kódjához!

```
public void Darken(byte decr)
{
 if (BackColor.R > decr)
    BackColor = System.Drawing.Color.FromArgb(
BackColor.R-decr,BackColor.G-decr,BackColor.B-decr);
 if (BackColor.R < 128)
    ForeColor = System.Drawing.Color.White;
}
```

A többi gomb választását is hasonló módon kell megoldanunk. Elegánsabb lenne, ha egy függvénnyel tudnánk lekezelni az összes eseményt. Ehhez azonban egyértelműen meg kell határoznunk a vezérlők indexét a Controls tömbben. A SzavSzamForm osztály konstruktorában tehetjük ezt meg.

```
public SzavSzamForm()
{
 InitializeComponent();
 // TODO: Add any constructor code after
 // InitializeComponent call
 Controls.SetChildIndex(jejeButton,0);
```

```
    Controls.SetChildIndex(gyozoButton,1);
    Controls.SetChildIndex(nagyButton,2);
    Controls.SetChildIndex(vesztButton,3);
    Controls.SetChildIndex(probaButton,4);
    Controls.SetChildIndex(jejeTextBox,6);
    Controls.SetChildIndex(gyozoTextBox,7);
    Controls.SetChildIndex(nagyTextBox,8);
    Controls.SetChildIndex(vesztTextBox,9);
    Controls.SetChildIndex(probaTextBox,10);
  }
```

Jelezzük az állást a szövegmezőben! Ehhez a legegyszerűbb, ha minden jelölthöz egy byte méretű segédváltozót használunk, mely számszerűen tartalmazza a szavazatok számát. A segédváltozókat is célszerű tömbben elhelyezni. Egyszerűen írjuk a kódot a SzavSzamForm osztály kódjába!

```
private byte[] szamlal = new byte[5] {0,0,0,0,0};
```

A gombválasztás eseménykezelőjét nevezzük Button_Click-nek! Nevezzük át a jejeButton Properties ablak Events gomjával megnyílt ablakban a Click esemény kezelőjét jejeButton_Click-ről Button_Click-re! A többi gomb esetén pedig ezt a függvényt válasszuk ki a Click esemény kezelőjeként!

```
private void Button_Click(object sender,
                                    System.EventArgs e)
{
  ((VoteButton)sender).Darken(10);
  // 10 a szavazók számától függően.
  int ind = Controls.GetChildIndex((VoteButton)sender);
  szamlal[ind]++;
  Controls[ind+6].Text = szamlal[ind].ToString();
}
```

Fordítás/Futtatás/Tesztelés

A kódban pontosan tudjuk, hogy a 'sender' VoteButton típusú, hisz csak ilyen gombok kiválasztása esetén hívtuk meg az eseménykezelőt. Tehát bátran konvertálhatjuk ilyen típusúra. Szükség is van erre, mert a Darken metódus csak az általunk írt osztályban létezik. Ha tudjuk a választott gomb indexét, akkor állíthatjuk a tömbben a hozzá tartozó értéket, és kiírhatjuk azt a megfelelő indexű TextBoxban.

5.1.4. A győztes meghatározása

Ez egy maximum keresést jelent. Mivel ki kell írnunk a győztes nevét is, célszerű a maximális elem indexét tárolni az értéke helyett. Tudjuk, hogy a tömb nem üres.

```
private void eredmButton_Click(object sender,
                               System.EventArgs e)
{
 int maxind=0;
 for (int i= 1; i<5; i++)
 {
   if (szamlal[i]>szamlal[maxind]) maxind=i;
 }
 MessageBox.Show("A győztes: "+Controls[maxind].Text+"
      "+szamlal[maxind].ToString()+" szavazattal.");
}
```

Fordítás/Futtatás/Tesztelés

A kód működik, csak ha ketten azonos számú szavazatot kapnak, akkor az első nevét írja ki. Pedig ebben az esetben a szavazás végeredménye döntetlen. Nézzük meg tehát, van-e a maximális elem után vele azonos értékű a tömbben!

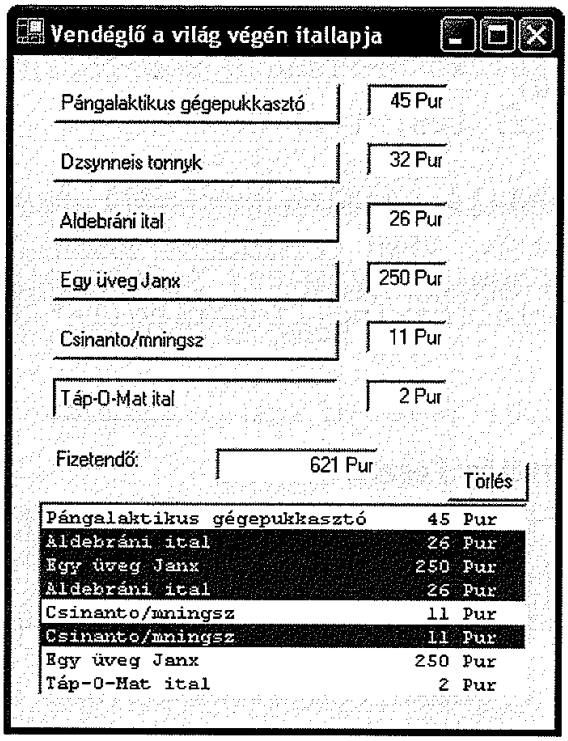
```
private void eredmButton_Click(object sender,
                                       System.EventArgs e)
{
 int maxind=0;
 for (int i= 1; i<5; i++)
 {
   if (szamlal[i]>szamlal[maxind]) maxind=i;
 }
 int j=maxind+1;
 while ((j<5) && (szamlal[j]!=szamlal[maxind])) j++;
 if (j<5)
   MessageBox.Show("Döntetlen!");
 else
   MessageBox.Show("A győztes: "+ Controls[maxind].Text
       +" "+szamlal[maxind].ToString() +" szavazattal.");
}
```

5.2. Vendéglő a Világ Végén

Douglas Adams nagysikerű sorozatának a Galaxis Útikalauz Stopposoknak egyik kötete hatására készült a következő feladat, mely a Vendéglő a Világ Végén bárjának itallapját mutatja be lehetővé téve az italok megrendelését, és a vendég

távozása előtt a számla elkészítését. A rendelés mutassa az eddig rendelt italok listáját és összárát!

Az itallap a különböző világokban kedvelt különleges italokat tartalmazza meglehetősen borsos, az ötcsillagos vendéglő különleges vendégkörének megfizethető áron.



5. Gyakorlat. 4. ábra A kész alkalmazás

Megjegyzés:

Mivel az olvasó nem biztos, hogy ismeri a művet, rövid ismertetőt fűznék az egyes italokhoz.

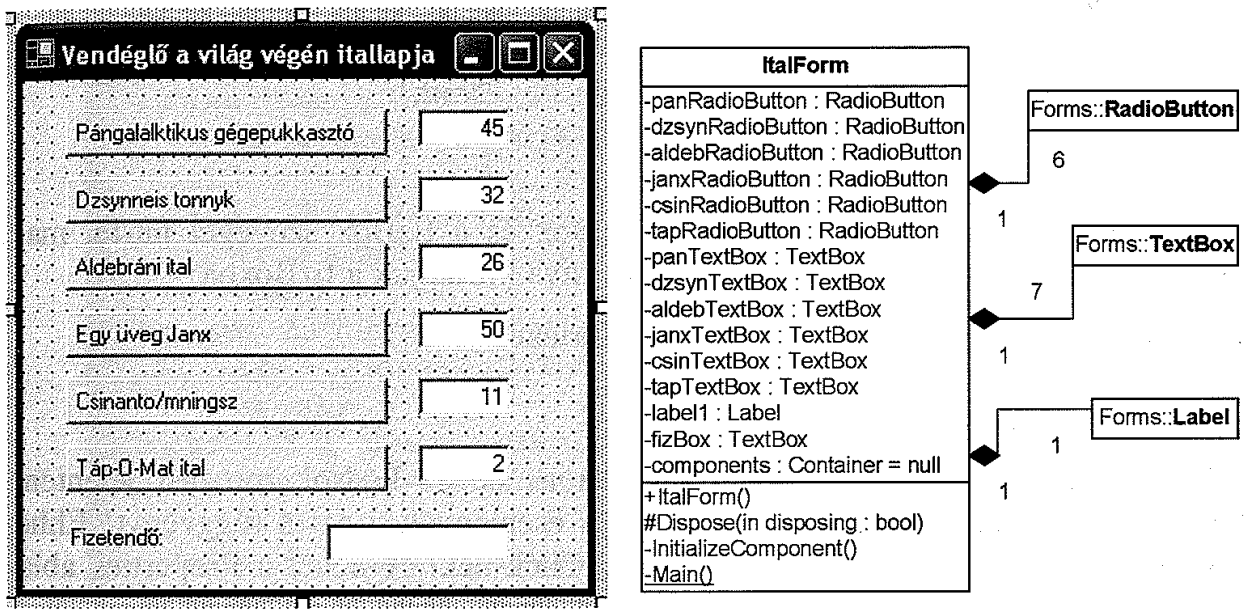
- Pángalaktikus gégepukkasztó: A Telepszichopatikus Turbomixerhez mérhetően szörnyű ital. Zaphod Beeblebrox egyik kedvence.
- Dzsynneis tonnyk: A Golgafrinchamről jött hajó kapitányának kedvence.
- Csinanto/mningsz: Sivolviai közönséges csapvíz, melyet szobahőmérsékletnél kicsit melegebben szervíroznak.
- Táp-O-Mat ital: A Szíriusz Kibernetikai Társaság által forgalmazott nutrimatikus italszintetizátor, mely a lehető legszélesebb választékát képes előállítani az italoknak a

mindenkori felhasználó ízlésének és metabolizmusának megfelelően.

Az italok ára trigani pur-ban van megadva, mely a Galaxis Útikalauz szerint a három szabadon konvertálható valuta egyike. 1 pur = 8 ningi. A ningi olyan háromszögletű gumiérme, melynek hatezer-nyolcszáz mérföld az oldalhossza, s ennélfogva senkinek sincs 1 purra való belőle. A Galaktibankok pedig nem bajlódnak aprópénzzel.

5.2.1. A felhasználói felület elkészítése

Generáljuk le az alkalmazást Itallap néven, a Form legyen ItalForm, és készítsük el a felhasználói felületét! A képen látható gombok RadioButton vezérlők. Nevük pan-, dzsyn-, aldeb-, janx-, csin-, tapRadioButton. A TextBoxok előtagjai is azonosak. A választógombok Appearance tulajdonsága legyen Button és FlatStyle tulajdonsága System, ezt az összes kijelölésével is beállíthatjuk.



5. Gyakorlat. 5. ábra A szerkesztőablak

A TextBoxok TextAlign tulajdonsága legyen Right! Legyen egy Fizetendő feliratunk, mellette egy fizBox nevű szövegmező, mely az összárat mutatja, szintén jobbra rendezett szöveggel.

Az osztálydiagram jól mutatja, hogy a System.Windows.Forms.Form-tól származó ItalForm osztály a System.Windows.Forms névtérben definiált RadioButton, TextBox és Label osztályok objektumait tartalmazza. A multiplicitás a tartalmazási kapcsolat mellett jelzi, melyik osztályból hány objektumot hoztunk létre.

5.2.2. Az italválasztás esemény kezelése

Ebben a feladatban kifejezetten a String osztály elemeivel szeretnénk ismerkedni, ezért nem tároljuk az árakat önálló tömbben, mindent a vezérlőkből fogunk kiolvasni, és a Text mezők alapján számolunk.

A Póló gombot kijelölve a Properties ablak Events  eszközgombját aktualizálva a Click eseményt válasszuk ki duplán! A fejlesztői környezet elkészíti a Click esemény kezelőmetódusát a `panRadioButton_Click` függvényt. Ez a függvény akkor hívódik meg, ha változott a gomb kiválasztása.

```
private void panRadioButton_Click (object sender,
                                   System.EventArgs e)
{
 int ar;
 int fiz;
 ar = Convert.ToInt32(panTextBox.Text);
 if (fizBox.Text == "")
   fiz = 0;
 else
   fiz = Convert.ToInt32(fizBox.Text);
 fizBox.Text=(fiz+ar).ToString();
}
```

A függvény kódja előbb átalakítja a poloBox és a fizBox szövegét int-té, majd összeadva őket a kapott számot stringgé alakítva kiírhatjuk a fizBoxba.

Ha a fizBox üres, akkor legyen 0 a fiz számértéke!

Fordítás/Futtatás

A kódot meg kellene írni a többi gombra is, de az kódsokszorozást jelentene, helyette rendeljük az összes gomb kattintás eseményéhez ezt a függvényt! Ehhez viszont nevezzük csak egyszerűen Button_Click-nek! A nevét a Properties ablak Events  eszközgombját aktualizálva a Click eseményt választva módosíthatjuk.

Megjegyzés:

Ellenőrizzük, hogy a legördülő listában nem maradt-e ott a régi függvény! Ha igen, akkor másoljuk át a kódot az újba, majd töröljük a régit!

Ezután minden gomb Click eseményéhez a Combo listájából válasszuk ki a Button_Click metódust! Nézzük meg a kódot, az InitializeComponent metódusban minden Button kódrészlete végén megjelent az esemény hozzárendelés sora.

Fordítás/Futtatás

Ezzel a függvény meghívódik, de természetesen minden esetben a pángalaktikus gégepukkasztó árával növeli az összeget. Honnan tudhatjuk meg, melyik gomb küldte az üzenetet? A sender paraméterből.

Tegyünk egy töréspontot a függvény valamelyik végrehajtható sorához! (F9) Majd futtassuk nyomkövetés üzemmódban a programot! (Start) Amikor egy gombot választunk, a program a nyomkövető ablakban vizsgálható, és a senderre mutatva egerünkkel elolvashatjuk, hogy melyik gomb küldte az üzenetet. Állítsuk le a nyomkövetést, és vegyük ki a töréspontot (F9)!

```
private void Button_Click(object sender,
                                System.EventArgs e)
{
 int ar = 0;
 int fiz;
 RadioButton r = (RadioButton)sender; // Átkonvertáljuk.
 switch (r.Text)
 {
   case "Pángalaktikus gégepukkasztó":
       ar = Convert.ToInt32(panTextBox.Text);
       break;
   case "Dzsynneis tonnyk":
       ar = Convert.ToInt32(dzsynTextBox.Text);
       break;
   case "Aldebráni ital":
       ar = Convert.ToInt32(aldebTextBox.Text);
       break;
   case "Egy üveg Janx":
       ar = Convert.ToInt32(janxTextBox.Text);
       break;
   case "Csinanto/mningsz":
       ar = Convert.ToInt32(csinTextBox.Text);
       break;
   case "Táp-O-Mat ital":
       ar = Convert.ToInt32(tapTextBox.Text);
       break;
 }
 if (fizBox.Text == "")
   fiz = 0;
 else
   fiz = Convert.ToInt32(fizBox.Text);
 fizBox.Text=(fiz+ar).ToString();
}
```

Fordítás/Futtatás

Ha szeretnénk, hogy az árat jelző számok mögött a 'Pur' ki legyen írva, akkor azt a számolás előtt ki kell szedni, míg a kiírás előtt vissza kell írni. Először írjuk be az árak mögé!

```
private void Button_Click(object sender,
                                    System.EventArgs e)
{
 int ar = 0;
 int fiz;
 char[]penz= {' ','P','u','r'};
 string sar = null;
 string sfiz = fizBox.Text.TrimEnd(penz);
 RadioButton r = (RadioButton)sender;
 switch (r.Text)
 {
   case "Pángalaktikus gégepukkasztó":
       sar = panTextBox.Text.TrimEnd(penz);
       break;
   case " Dzsynneis tonnyk ":
       sar = dzsynTextBox.Text.TrimEnd(penz);
       break;
...
 }
 ar = Convert.ToInt32(sar);
 if (fizBox.Text == "")
   fiz = 0;
 else
   fiz = Convert.ToInt32(sfiz);
 fizBox.Text=(fiz+ar).ToString()+" Pur";
}
```

Ha ezt a sok változtatást egyenként kellett volna minden gombhoz rendelt függvényen végrehajtani, bizonyosan lett volna olyan, amit elhibázunk, és még tesztelni sincs kedvünk, tehát a hiba akár a programban is maradhat.

Fordítás/Futtatás

Megjegyzés:

A fizBox.Text kiszámításánál akár el is hagyhatjuk a ToString függvényhívást, mert a + operátor, ha egyik operandusa sztring, a másik numerikus operandust automatikus konverzióval átalakítja a neki megfelelő sztringgé.

5.2.3. Legyen az ablakban egy a megrendelt italokat felsoroló listadoboz!

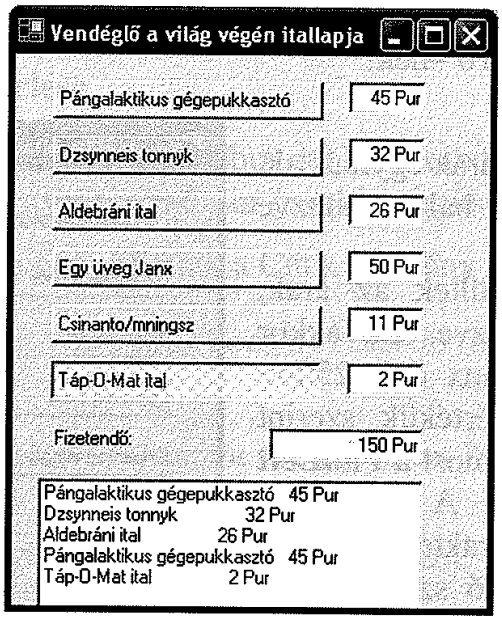
Az ItalForm tervezés nézetében (Design) az eszköztárból (ToolBox) válasszuk ki a ListBox elemet, és igazítsuk el az ablak felületén! A listadoboz neve legyen italokBox! A gomb kiválasztásakor az italt vegyük fel a listára!

```
private void Button_Click(object sender, …)
{…
 fizBox.Text=(fiz+ar).ToString()+" Pur";
 italokBox.Items.Add(r.Text+"\t\t"+sar+" Pur");
}
```

Fordítás/Futtatás

Ha több elemet veszünk föl a listára, görgetéssel érhetjük el azokat. Mivel a szövegek jelentősen eltérnek, azonos tabulátorszám különböző hosszúságú szöveget eredményez. A szöveg tetszőleges karakterrel történő adott szélességű kitöltését biztosítják a string osztály **Pad** függvényei. Ha nem adjuk meg a karaktert, szóközzel tölti ki a helyet.

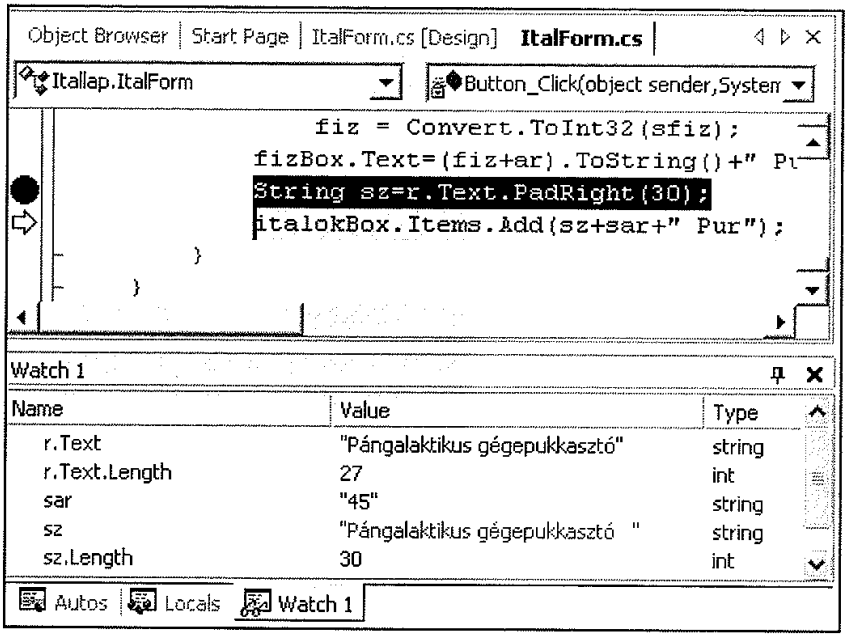
```
String sz=r.Text.PadRight(30);
italokBox.Items.Add(sz+sar+" Pur");
```



5. Gyakorlat. 6. ábra Az árak még mindig nincsenek egymás alatt a listadobozban

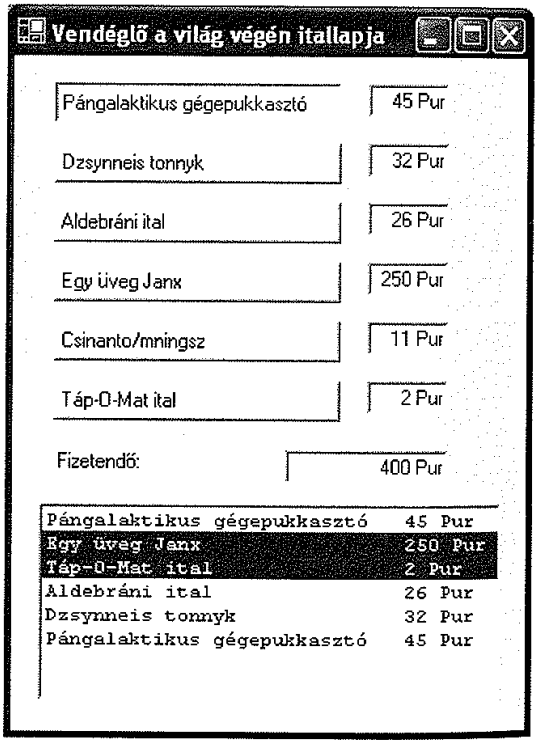
Még mindig nem kerültek egymás alá a számok! Nézzük meg **Debugger**rel, vajon 'sz' 30 hosszú-e! Tegyünk töréspontot a fenti kód első sorába (a szürke csíkon kattintva), majd futtassuk az alkalmazást! Amikor megállt a végrehajtás, válasszuk a Watch1 fület, és írjuk be a kívánt változókat!

Azt látjuk, hogy a sor végrehajtása (Step Over ikon) után 'sz' hossza 30 lett.



5. Gyakorlat. 7. ábra Az 'sz' hossza valóban 30 karakter

A magyarázat az, hogy az alapértelmezésben beállított Microsoft Sans Serif font ún. proporcionális font, vagyis cellaszélessége karakterenként változik (a szóközé a legkisebb). Ha fix szélességű fontot (pl. Curier) választunk a kiíráshoz, a kívánt eredményt érhetjük el. Válasszuk a listadobozunk **Font** tulajdonságát a … gomb felhasználásával Curier New 10 –nek!

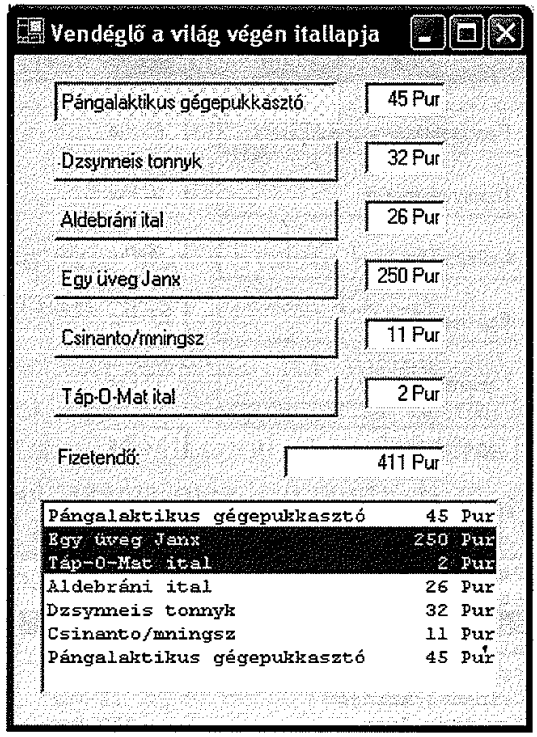


5. Gyakorlat. 8. ábra Az árak egymás alá balra rendezve

Most már egymás alá kerültek az árak, viszont balra vannak rendezve. Ez akkor látszik még jobban, ha a Janx árát 250-re emeljük. Az árakat helyi értékük szerint szoktuk egymás alá írni. Ezt most a **PadLeft** függvénnyel oldhatjuk meg. A következő kódsor szóközökkel 4 karakter szélesre egészíti ki az árat tartalmazó sztringünket, úgy, hogy a szóközöket balról teszi a szöveg elé.

```
sar = sar.PadLeft(4);
```

Fordítás/Futtatás



5. Gyakorlat. 9. ábra Az árak egymás alá jobbra rendezve

Vizsgáljuk meg a listadoboz viselkedését, ha sok elemmel töltjük fel! A gördítősáv megjelenéséhez szélesebb listadoboz kell.

5.2.4. A listából lehessen törölni is!

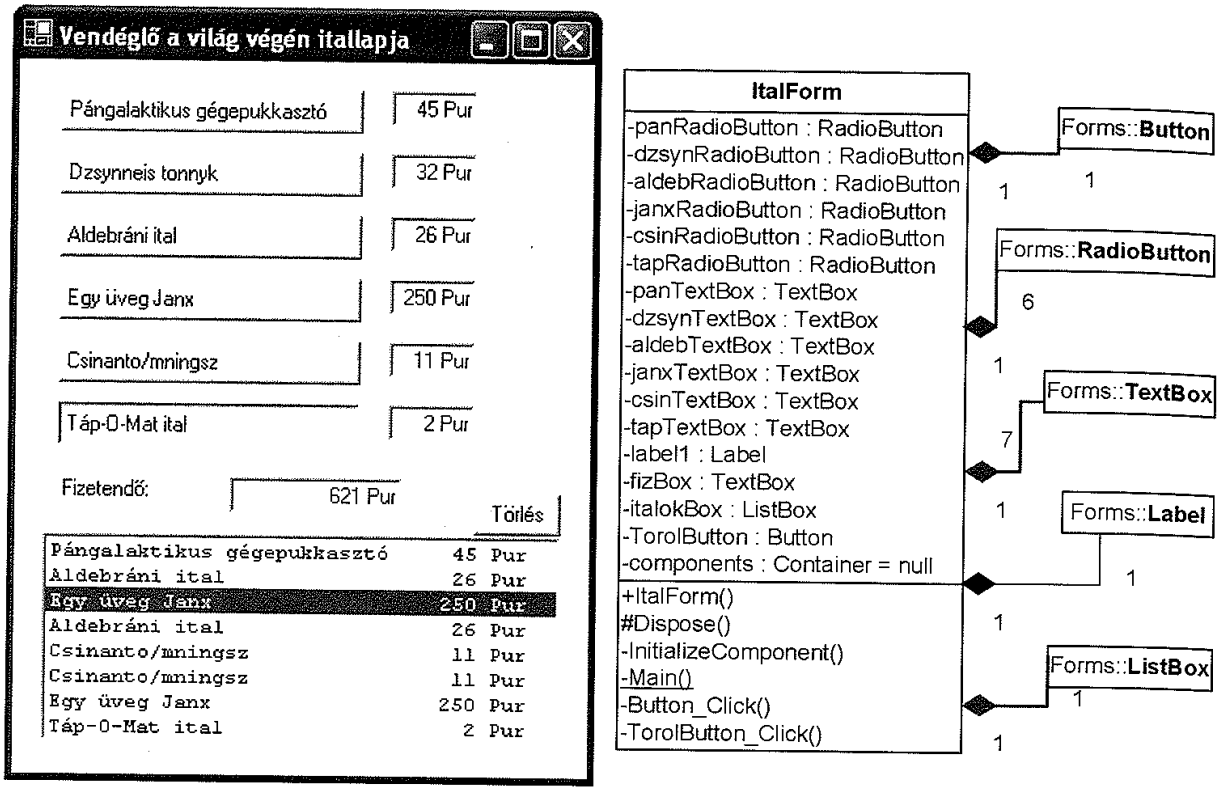
5.2.4.1. Egy elem törlése

Tegyünk egy töröl gombot a listadoboz jobb sarkához! (Name: TorolButton, Text: Törlés) Rendezzük a Layout eszközsor gombjai segítségével!

Duplán kattintva a gombon, írhatjuk a Click esemény kezelőmetódusát.

```
private void TorolButton_Click(object sender,
                                    System.EventArgs e)
{
 italokBox.Items.Remove(italokBox.SelectedItem);
}
```

Teszteljük a függvényt, ha nincs kiválasztott elem, nem töröl semmit, egyébként törli a kiválasztottat.



5. Gyakorlat. 10. ábra A törlés gomb és az osztálydiagram

5.2.4.2. Több elem törlése

Ha meg akarjuk engedni több elem kiválasztását is egyszerre, ahhoz először állítsuk a listadoboz **SelectionMode** tulajdonságát **MultiExtended**-re, majd a kezelőfüggvényt módosítsuk!

Azt gondolnánk, hogy egy egyszerű megoldás, ha a kiválasztott indexeket tartalmazó tömb elemein végigmegyünk és kitöröljük őket:

```
foreach(int i in italokBox.SelectedIndices)
  italokBox.Items.RemoveAt(italokBox.SelectedIndices[i]);
```

Azonban a futáskor kiderül, hogy ez nem működik. Vizsgáljuk meg nyomkövetéssel az okot! Tegyünk egy töréspontot a foreach sorra, és futtassuk az alkalmazást! Több elemet válasszunk és töröljünk! (A táblázat adatai 4 elem kiválasztásával állíthatók elő.) A Watch ablakban alul beállíthatjuk a megfigyelni kívánt adattagokat, majd lépésenként hajtsuk végre a programot:

Name	Value	Type
i	2	int
italokBox.SelectedIndices.Count	2	int
italokBox.SelectedIndices[i]	{"Index was outside the bounds of the array." }	System.IndexOutOfRangeException

Előbb vagy utóbb a tömb túlindexelése üzenetet kapjuk. Ha megnézzük a részleteket láthatjuk, hogy a kiválasztott elemek indexe végigmegy az eredetileg kiválasztott számig, de minden törléskor csökken a kiválasztott elemek száma. A fenti táblázat mutatja, hogy a kiválasztott elemek száma még 2 (count), és mi a 2 indexűt, vagyis a harmadikat szeretnénk törölni. (Az indexelés 0-val kezdődik.) Ez természetesen a tömb elemein túlmutat, tehát hibához vezet. Javítsuk a kódot!

```
private void TorolButton_Click(object sender,
                                        System.EventArgs e)
{
 int i = italokBox.SelectedIndex;
 while ( i >= 0)
 {
    italokBox.Items.RemoveAt(i);
    i= italokBox.SelectedIndex;
 }
}
```

Fordítás/Futtatás

Próbáljuk ki a SelectionMode **MultiSimple** tulajdonságát is!

5.2.4.3. Törléskor módosuljon a fizetendő értéke!

Ehhez a kiválasztott elem árát kell kivonnunk a fizetendő összegből. Mindkettőhöz sztringként jutunk hozzá, és mindkettő mögött ott a Pur felirat. Először olyan sztringgé alakítjuk őket, amiben csak szóköz és számjegyek vannak, majd számmá konvertáljuk, hogy elvégezhessük a kivonást. Utolsó lépésként visszaalakítjuk sztringgé a módosított értékét a fizetendő mezőnek.

```
private void TorolButton_Click(object sender,
                                    System.EventArgs e)
{
 int i = italokBox.SelectedIndex;
 string sar=null, sfiz;
 int fiz;
 char [] penz = {' ','P','u','r'};
```

```
    while ( i >= 0)
    {
        sar = italokBox.Items[i].ToString();
        // A kiválasztott.
        // Kiszedi a 4 hosszúságúra beállított számot,
        // Négy karakter a végén a Pur.
        sar = sar.Substring(sar.Length-8,4);
        sfiz = fizBox.Text.TrimEnd(penz);
        fiz = Convert.ToInt32(sfiz);
        fiz -= Convert.ToInt32(sar);
        fizBox.Text = fiz.ToString()+" Pur";
        italokBox.Items.RemoveAt(i);
        i= italokBox.SelectedIndex;
    }
}
```

Megjegyzés:

A Pur felirat levágható a Substring(0, fizBox.Text.Length-4) függvény hívásával is.

A Pur megjelenítése megoldható úgy is, hogy a háttérben végig karbantartunk fiz nevű numerikus adattagot, és annak értékét írjuk ki a képernyőre a ToString metódus segítségével a Purt hozzáfűzve.

Lehetőség az is, hogy a Pur feliratot egy Label segítségével tesszük ki a szövegmező mellé.

5.3. Feladatok

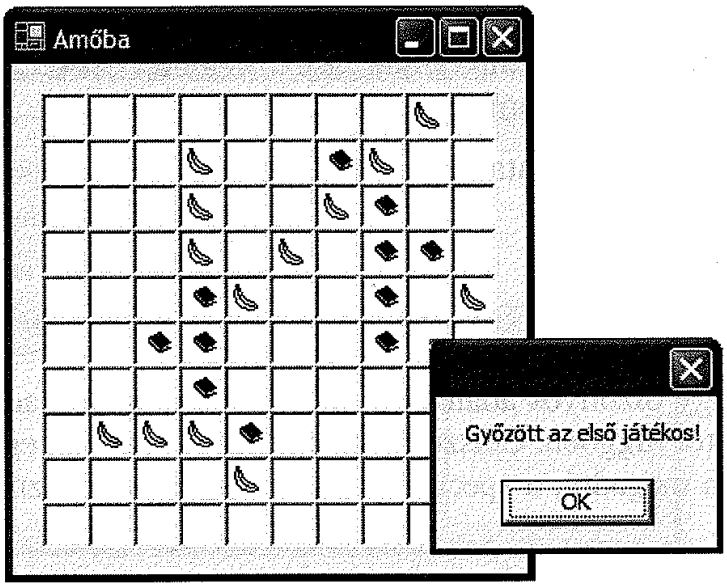
1. Reklámfilmhez szereplőválogatást tartanak. Minden meghallgatáson résztvevőről felveszik a név, telefonszám, időpont, pontszám, szöveges megjegyzés adatokat. Ezen kívül a színészeknél a színiiskolákat, eddig játszott szerepeket: táncosoknál a táncképzéseket, eddigi előadásokat, milyen típusú táncot ismer; kaszkadőröknél: milyen típusú trükköt vállal; statisztáknál: eddig hány filmben statisztált.*

2. Bizonyos összegű rendelés fölött a vendég törzsvendégkártyát kap, mely mindig aktualizálható kedvezményeket (ajándéktárgy, árengedmény...) jelent. Ezt jelezze a program a fizetőpincérnek úgy, hogy az összeg színe megváltozik, és vastagon szedett lesz.

3. Használjunk CheckedListBox vezérlőt az alkalmazásban, a kiválasztott lista tárolására a checked segítségével lehessen kijelölni a törlésre szánt elemeket!

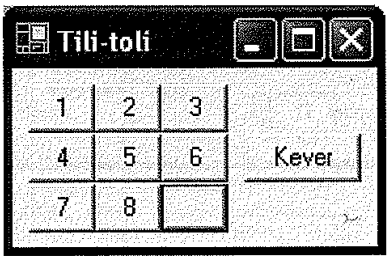
4. Oldjuk meg az itallap feladatot úgy is, hogy a háttérben numerikus adatok tárolják az árakat és az összeget!

5. **Amőba**, a jelek: ikonok, melyeket egérkattintással tesznek ki a játékosok egymást váltva. A győztes, aki 5 azonos jelet tud egymás mellé tenni.*



6. **Aknakereső**. Amelyik gombra az egér mutat – átszíneződik, kattintásra választható. Ha nem aknát választunk, írja ki, hogy hány szomszédos akna van! Aknatalálat esetén veszített, az ábra mutassa meg az összes akna helyét!

7. **Tili-toli játék**. Egy 3x3-as négyzetben 1-8-ig számok találhatóak. Egy kocka üres. A játékos az üressel szomszédos mezők valamelyikére kattintva, az ott elhelyezkedő számot az üres helyre cseréli. Cél a számok sorba rendezése.[1]*



8. **Választógombok**. Teszteljük a választógombok használatát! Legyen két csoport, az egyik számokat jelöljön (1, 2, 3), a másik betűket (a, b, c, d)! A két csoport gombjaihoz rendeljünk egy-egy eseménykezelőt, mely a kiválasztott gomb mellé kiírja: „választva: 2" vagy „választva: d"! A ki nem választott gombok felirata legyen az eredetivel megegyező!*

---

[1]Feladatötlet: Baga Edit, Delphi másképp, Budapest 1998, 88.old. A feladat szerepel a szerző A Visual C++ és az MFC könyvében is, de mindhárom kötet lényegében eltérő megoldást mutat be.

9. Az itallapon az italok árai módosíthatóak. Ne engedjük meg a felhasználónak, hogy számjegyektől eltérő karaktereket vigyen be a szövegmezőbe!*
10. Eladó alkalmazás. Készítsünk el tetszőleges üzlet (divatáru, autó, illatszer, könyv) eladását támogató alkalmazást!
11. Az Elado alkalmazásban a gombokat választva jelenjen meg egy további párbeszédablak, ahol beállíthatjuk a részleteket: méret, szín, garbó-e, ujja hossza..., vagy egy listából választhatunk a készletből.
12. Jelszo alkalmazásunk adatait mentsük fájlba!*

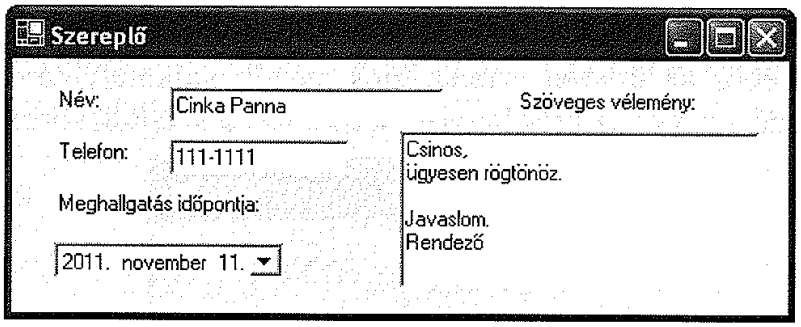
5.4. Megoldásötletek

1. Szereplőválogatás

Láthatjuk, hogy bizonyos adatokat mindenkinél meg kell adnunk, míg másokat, a foglalkozástól függően. Célszerű lenne a közös adatokat egy ős form osztályon SzereploForm megadni, a speciális adatokat pedig utódosztályok esetén bekérni (absztrakció).

Akár mindjárt az adatfelvétel elején megkérdezhetjük a jelentkező foglalkozását, s így a megfelelő form betöltésével kérdezhetjük meg adatait.

Készítsük el az ősosztály formját!



5. Gyakorlat. 11. ábra A SzereplőForm a további formok őse

Készítsük el az utódosztályokat!

Solution View

ReklamSzerep projekt jobbegér

Add

Add Inherited Form

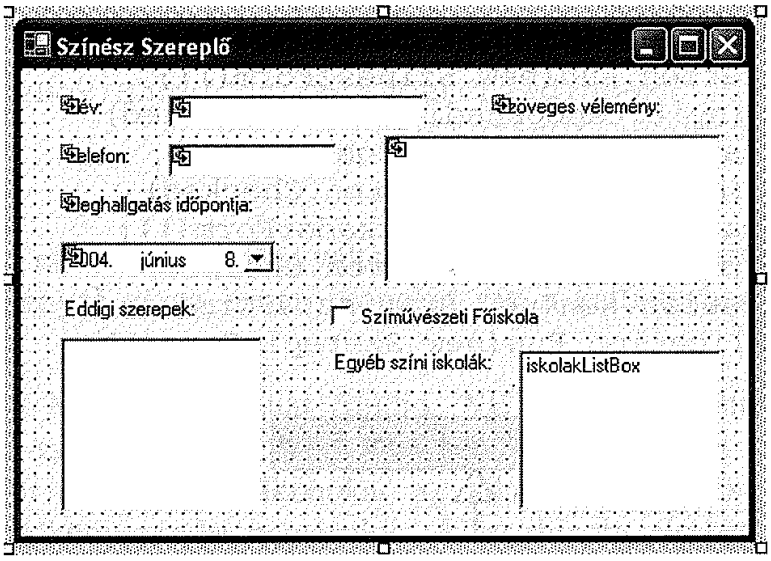
Name: SzineszForm.cs

Az Inherited Form legyen kiválasztva!!!

Open

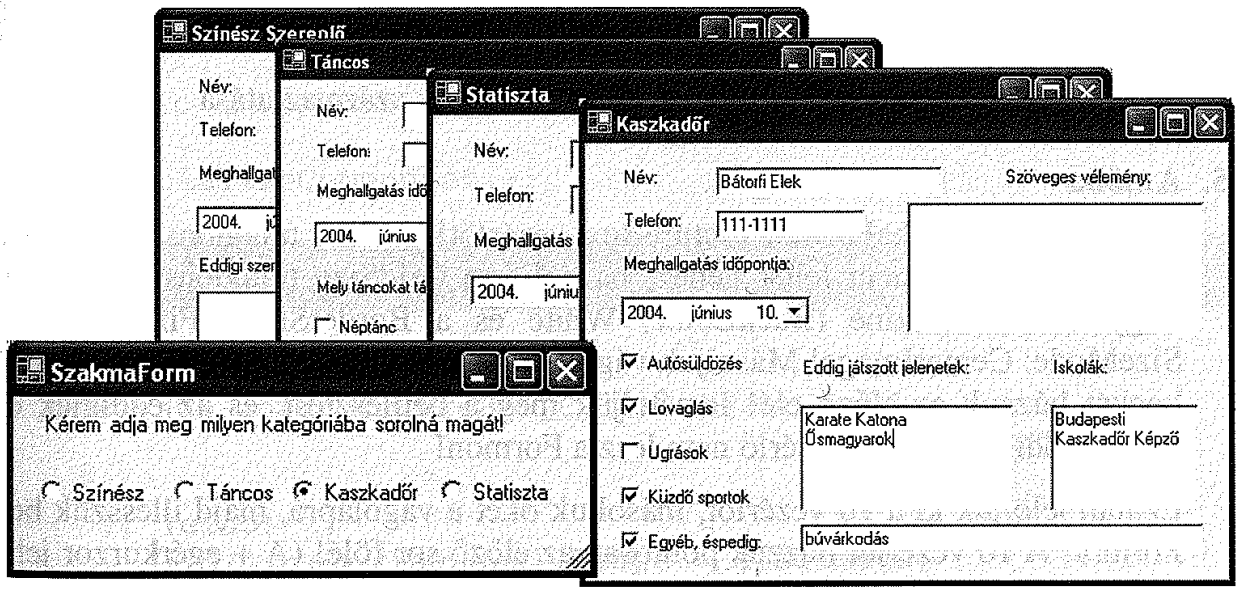
Inheritance Pickerben: SzereploForm

A szerkesztőablak mutatja az öröklött vezérlőket.



5. Gyakorlat. 12. ábra A SzínészForm a SzereplőForm utóda a szerkesztőben

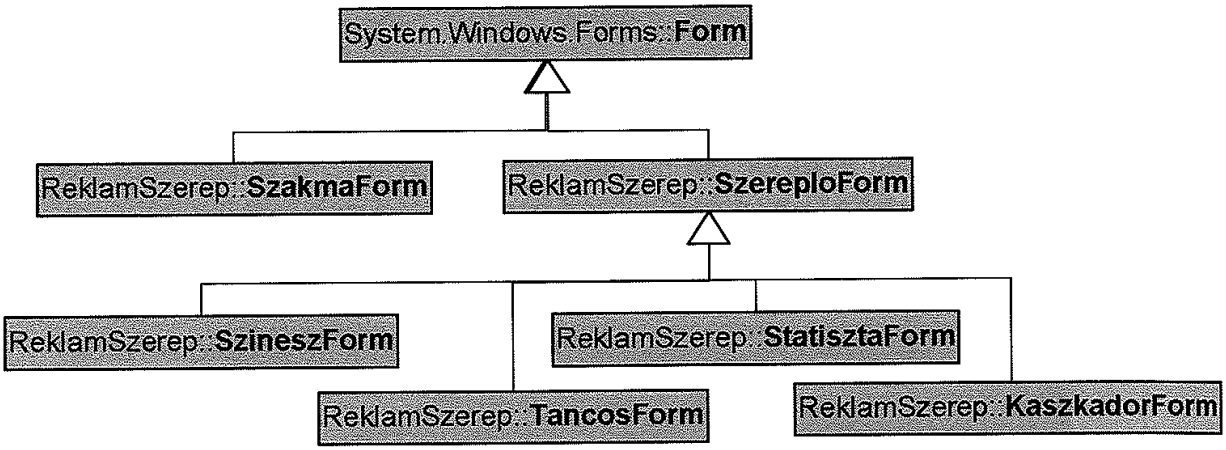
A többi utódosztályt is elkészítve, a főablak (SzereploForm) Main metódusában meghívjuk a nyitó ablakot, majd a választástól függően a megfelelő formot.



5. Gyakorlat. 13. ábra A megnyitó ablak (SzakmaForm) és az utód formok

A Main kódja:

```
static void Main()
{
 SzakmaForm nyit = new SzakmaForm();
 nyit.ShowDialog();
 if (nyit.szineszRadioButton.Checked)
   Application.Run(new SzineszForm());
 else if (nyit.tancosRadioButton.Checked)
   Application.Run(new TancosForm());
 else if (nyit.kaszRadioButton.Checked)
   Application.Run(new KaszkadorForm());
 else if (nyit.statRadioButton.Checked)
   Application.Run(new StatisztaForm());
 else Application.Run(new SzereploForm());
}
```



5. Gyakorlat. 14. ábra A form osztályok származtatása

5. Amőba

Húzzunk az AmobaFormra PictureBox vezérlőt! A Size tulajdonságát állítsuk 24; 24-es-re! Ez nem lehet nehéz, ha a Form GridSize tulajdonsága 8; 8-as. Legyen a háttérszíne (BackColor) White és a BorderStyle: Fixed3D és a SizeMode: CenterImage! Másoljuk vágólapra, majd illesszük be a Formra! Az új kockát húzzuk az előző elé! Ismételjük meg a beillesztést, és az eddigiek elé húzást addig, amíg 10 vezérlő nem lesz a Formon!

Ezután jelöljük ki a 10 vezérlőt, másoljuk őket a vágólapra, majd illesszük be a Formra! A 10 vezérlőt húzzuk pontosan az előző sor fölé! (A + egérkurzor jelzi, ha megfogtuk őket.) Ismételjük meg a beillesztést és az előző sor elé húzást addig, amíg 10 sorunk nem lesz!

Mivel az újra és újra beszúrt vezérlőket a fejlesztőkörnyezet a Controls lista elejére illeszti, vagyis az utoljára felvett lesz az első a listán, ezért készítjük el a felületet hátulról előre. Teszteljük, jól helyezkednek-e el a képek! Kezeljük az

összes vezérlőhöz a Click eseményt! Jelöljük ki (bekeretezéssel a vezérlőket), majd a Click eseményen kattintsunk duplán! A Design View / Properties / Events ablakban módosítsuk a nevét PictureBox_Click-re!

```
private void pictureBox_Click(object sender,
                                        System.EventArgs e)
{
 MessageBox.Show(Controls.GetChildIndex(
                          (PictureBox)sender).ToString());
}
```

Tároljuk egy 'elso' nevű logikai változóban, hogy melyik versenyző következik! Az első versenyző sárga banán ikonokat rak a kattintáskor, a második piros könyvet.

Az ikonok a Microsoft Visual Studio .NET 2003\Common7\Graphics\icons könyvtárban találhatók, de természetesen máshonnét is választhatunk. A legegyszerűbb a választott ikonokat az alkalmazásunk exe fájljával azonos (bin/Debug) alkönyvtárába bemásolni, akkor mindig elérhető lesz. Tároljuk a képeket egy-egy segédváltozóban, a többi referencia erre hivatkozzon! Egyrészt így nem kell mindig új memóriahelyre fájlból beolvasni, másrészt így összehasonlíthatóak lesznek.

```
private bool elso=true;
private Image elsoImage;
private Image masodikImage;

public AmobaForm()
{
 InitializeComponent();

 elsoImage = Image.FromFile( "Point11.ico");
 masodikImage = Image.FromFile( "BOOK01A.ico");
}

private void pictureBox_Click(object sender,
                                        System.EventArgs e)
{
 if (elso)
    ((PictureBox)sender).Image = elsoImage;
 else
    ((PictureBox)sender).Image = masodikImage;
 elso=!elso;
}
```

A győztest a Winner függvény keresi meg, melyben a lineáris keresés és a számlálás algoritmust alkalmazzuk.

```
private void pictureBox_Click(object sender,
                                    System.EventArgs e)
{
 if (elso)
    ((PictureBox)sender).Image = elsoImage;
 else
    ((PictureBox)sender).Image = masodikImage;
 if (Winner(sender))
 {
    if (elso)
        MessageBox.Show("Győzött az első játékos!");
    else
        MessageBox.Show("Győzött a második játékos!");
    for (int i=0;i<100;i++)
    {
        ((PictureBox)Controls[i]).Image=null;
    }
    elso=true;
 }
 else
    elso=!elso;
}

private bool Winner(object sender)
{
 Image image=((PictureBox)sender).Image;
 //Balra, föl kezd.
 int x=-1;
 int y=-1;
 int ind=Controls.GetChildIndex((PictureBox)sender);
 while (y<1) //Az utolsó irány x=1, y=0
             //(Az 1,1 a -1,-1 ellentéte már.)
 {
    int i=ind/10;
    int j=ind%10;  //Osztási maradék. ind mod 10
    // Addig megy, míg szélre ért, vagy azonosat talál.
    i+=y;     // sor
    j+=x;     // oszlop
    while ((i>-1) && (j>-1) && (i<10) && (j<10) &&
        (((PictureBox)Controls[i*10+j]). Image==image))
    {
        i+=y;
        j+=x;
    }
```

```
        //Most kezdi a számolást. Visszaindul.
        int count=0;
        i-=y;
        j-=x;
        while ((i>-1) && (j>-1) && (i<10) && (j<10) &&
            (((PictureBox)Controls[i*10+j]). Image==image))
        {
            count+=1;
            i-=y;
            j-=x;
        }
        if (count < 5) //Irányt vált.
        {
            if (x<1) x++;
            else y++;
        }
        else return true; // Győzött.
    }
    return false;
}
```

Ne engedjünk kép fölé kattintani, a következő játékost jelezhetjük a kurzor megváltoztatásával!

7. Tili-toli

Oldjuk meg a feladatot úgy, hogy létrehozunk egy olyan Button utódosztályt, mely tartalmaz egy byte sor és egy byte oszlop adattagot, melyet kívülről is elérhetünk (internal)! Mentsük el! Tegyünk 9 gombot az ablakra! A gombok legyenek ilyen TiliButton típusúak, és feliratuk a képen látható! A bal felső sarok sora=0, oszlopa=0, mellette 0,1 és 0,2; a következő sor értékei: 1,0; 1,1; 1,2, és a legalsó sor 2,0; 2,1; 2,2. Az utolsó gomb legyen felirat nélküli! A TiliForm osztály tartalmazzon még egy uresButton TiliButton típusú adattagot, mely kezdőértékül a button9-et kapja!

Írjuk meg a csere függvényt, melynek hatására az üres az új elemre mutat, a régi üres pedig megkapja a választott gomb szövegét!

```
public void Csere(TiliButton b)
{
 uresButton.Text=b.Text;
 uresButton=b;
 b.Text="";
}
```

Jelöljük ki az összes gombot bekeretezéssel, és a Properties ablak Events gombját választva, a Click eseményen duplán kattintva létrehozza a

Button1_Click eseménykezelőt, melyet visszatérve a Properties ablakba átnevezhetünk Button_Click-re. Ha az üressel szomszédos gombra kattintunk, hívjuk meg a cserét!

```
private void button_Click(object sender,
                                        System.EventArgs e)
{
 TiliButton b=(TiliButton)sender;
 if(Math.Abs(b.sor-uresButton.sor)+
            Math.Abs(b.oszlop-uresButton.oszlop)==1)
   Csere(b);
}
```

Nem minden számsorrend ad megoldható feladatot. Úgy érhetjük el, hogy a feladat mindig megoldható legyen, hogy az elején a cserék segítségével megkeverjük a gombokat.

```
private void keverButton_Click(object sender,
                                        System.EventArgs e)
{
 Random rand=new Random();
 int n=rand.Next(100);
 for (int i=0; i<n; i++)
   button_Click(Controls[rand.Next(9)+1], e);
}
```

8. Választógombok

```
private void radioButton_CheckedChanged(object sender,
                                        System.EventArgs e)
{
 if (((RadioButton)sender).Checked)
    ((RadioButton)sender).Text="választva:
                          "+((RadioButton)sender).Text;
 else
     ((RadioButton)sender).Text=
              ((RadioButton)sender).Text.Substring(11,1);
}
```

9. Az itallap árai csak számjegyek lehetnek!

Ha a Pur szöveget nem a szövegmezőben, hanem azon kívül egy label segítségével írjuk ki az ablakba, nem kell sztring műveletekkel levágnunk a szám végéről ezeket a karaktereket. Ez esetben megtehetjük, hogy nem

engedünk mást, csak számjegyeket a szövegmezőbe írni. Ehhez kezeljük a szövegmező KeyPress eseményét!

```
private void KeyPress(object sender,
                                    System.EventArgs e)
{
 if (!Char.IsDigit(e.KeyChar))
 {
   e.Handled = true;
 }
 else
 {
   e.Handled = false;
 }
}
```

Ha a negatív előjelet is meg akarjuk engedni, akkor a feltétel így módosul:

```
 if (!Char.IsDigit(e.KeyChar)|| e.KeyChar == '-')
```

12.Jelszavak fájlból olvasása

A legegyszerűbb ha a meglevő tömbünk adatait előbb fájlba írjuk, majd ezzel a fájllal dolgozunk.

```
System.IO.FileStream fs =
            new System.IO.FileStream("jelszo.pwd",
            System.IO.FileMode.OpenOrCreate);
System.IO.BinaryWriter bw =
                  new System.IO.BinaryWriter(fs);
bw.Write(numOfUsers);
foreach (string str in user)
 bw.Write(str);
foreach (string str in jelszo)
 bw.Write(str);
fs.Close();
```

A betöltéskor beolvassuk a tárolt adatokat.

```
private void JelszoForm_Load(object sender,
                                    System.EventArgs e)
{
 try
```

```
    {
        System.IO.FileStream fs =
                    new System.IO.FileStream("jelszo.pwd",
                    System.IO.FileMode.Open);
        System.IO.BinaryReader br =
                    new System.IO.BinaryReader(fs);
        numOfUsers = br.ReadInt32();
        user = new string[numOfUsers];
        jelszo = new string[numOfUsers];
        for (int i=0; i<numOfUsers; i++)
            comboBox1.Items.Add(br.ReadString());
        for (int i=0; i<numOfUsers; i++)
            jelszo[i] = br.ReadString();
        fs.Close();
    }
    catch (System.IO.FileNotFoundException fe)
    {
      MessageBox.Show("A(z) '"+ fe.FileName+ "' fájl nem
                                        elérhető");
      Application.Exit();
    }
}
```

6. Gyakorlat. A Toolbox komponensek

Szeretnénk az alkalmazásunkból más alkalmazásokat indítani, illetve az általunk indított programokat szeretnénk leállítani. A gyakorlat célja, hogy egyszerűen úgy indítsunk alkalmazásokat, hogy futás közben nem kívánunk az alkalmazásokkal kommunikálni, azok önállóan, az általunk fejlesztett alkalmazással párhuzamosan futnak.

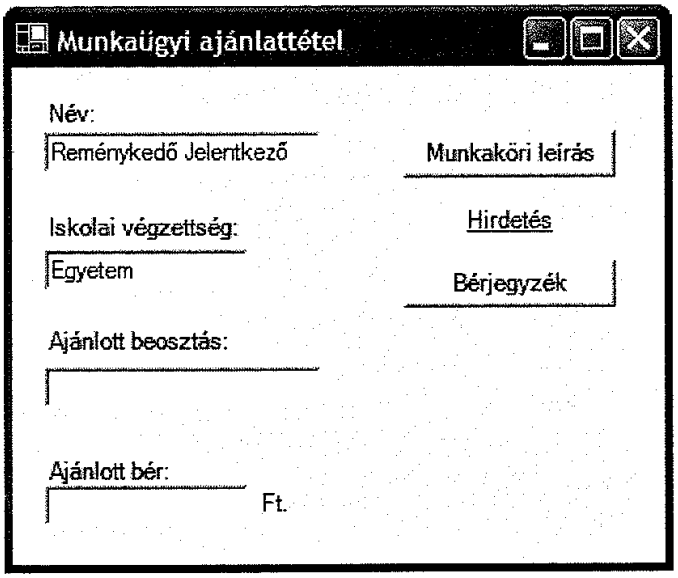
A gyakorlat 2. feladata az időzítő használatát mutatja be.

6.1. Több folyamat indítása

Ha egy alkalmazást készítünk, felmerülnek olyan igények, hogy az alkalmazásból szeretnénk már elkészült dokumentumainkat elérni.

Egy munkaügyi előadó tevékenységét támogató alkalmazásban az előadónak az adatok kitöltéséhez, az ajánlattételhez szüksége lehet a következő dokumentumokra:

- **Internet Explorer** segítségével egy **link label**ről szeretné elérni az interneten meghirdetett álláshirdetés pontos szövegét.
- A cég dokumentumai közt található a meghirdetett állás munkaköri leírása **MSWord** formátumú .doc fájlként. Ezt egy gomb segítségével kívánja megnyitni.
- A cég dokumentumai közt szerepel a dolgozók bérének nyilvántartása **Excel** fájlban. A bérajánlathoz e fájlt is szeretné olvasni munkája során.



6. Gyakorlat. 1. ábra Az alkalmazás főablaka

Készítsük el a felhasználói felületet az ábrának megfelelően! A 'Munkaköri leírás' gomb legyen workButton, a 'Hirdetés' LinkLabel legyen advertiseLinkLabel, a 'Bérjegyzék' gomb salaryButton nevű!

6.1.1. A folyamat indítása statikus metódushívással

A dokumentum elérésének legegyszerűbb módja, ha a Process osztály statikus **Start** metódusának hívásakor használjuk az operációsrendszer fájlkezelőjének beállítását, mely a dokumentum kiválasztásakor, annak kiterjesztése alapján elindítja a dokumentumot kezelő alkalmazást.

Egy folyamatot szeretnénk indítani! Keressük meg a súgóban a **Process** osztályt! Az about Process class leírás végén találjuk, hogy a System.dll-ben a **System.Diagnostics** névtérben szerepel a leírása. A System.dll-t már használjuk. Ha úgy véljük, a programban többször fogunk a Process osztállyal dolgozni, és nem kívánjuk minden alkalommal a teljesen minősített nevét megadni (System.Diagnostics.Process), akkor a Diagnostics névteret célszerű a using direktívával elérhetővé tenni.

```
using System.Diagnostics;
```

A dokumentumot a workButton választásakor szeretnénk megnyitni:

```
private void workButton_Click(object sender,
                              System.EventArgs e)
{
  Process.Start("Munkakor.doc");
}
```

Fordítás/Futtatás/Tesztelés

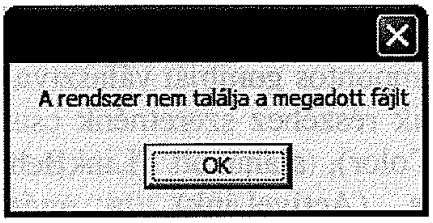
Ha a munka projekt bin / Debug alkönyvtárában ott van a kívánt dokumentum, a gomb választás hatására elindul a Word és megnyitja a fájlt.

Ha azonban nem érhető el a Munkakor.doc fájl, akkor kezeletlen kivétel hibaüzenetet kapunk. Alkalmazzuk a kivételkezelést arra a kivételes helyzetre, ha nem érhető el a fájl!

```
private void workButton_Click(object sender,
                                        System.EventArgs e)
{
 try
 {
    Process.Start("Munkaor.doc");
 }
 catch(Exception exception)
 {
    MessageBox.Show(exception.Message);
 }
}
```

Fordítás/Futtatás/Tesztelés

Látjuk, hogy a kivételt nem mi dobtuk, hanem a Process osztály Start metódusa. Ha nem nézünk utána a súgóban, hogy a függvény milyen kivételt dob, vagy egyszerűen bármely kivételt el szeretnénk kapni, akkor a legegyszerűbb az összes kivételosztály ősének, az **Exception** osztálynak egy példányát elkapnunk. Az ős referencia hivatkozhat utódra, tehát minden kivételt el tud kapni. Mivel az Exception osztálynak van **Message** tulajdonsága, így minden kivételnek van Message tulajdonsága. Ha ezt írjuk ki egy üzenetablakban, akkor láthatóan az operációs rendszer nyelvén kulturált és tájékoztatást nyújtó hibaüzenetet kapunk.



6. Gyakorlat. 2. ábra A kezelt kivétel által kiírt hibaüzenet

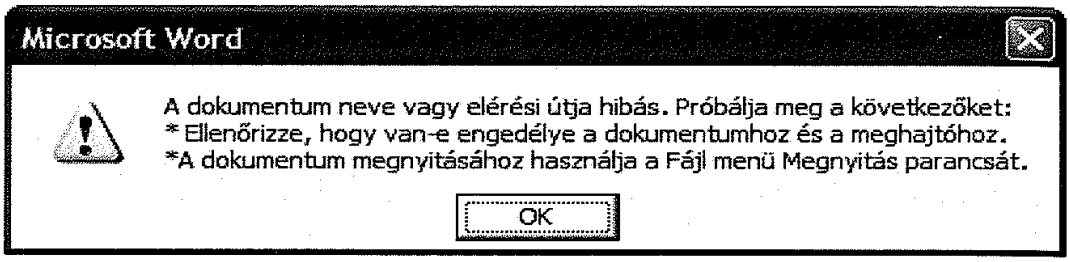
Ha nem kívánjuk felhasználni a kiterjesztéshez (doc) rendelt alkalmazás (WinWord) beindítását – ami a fájlkezelőben van beállítva–, akkor meg kell adnunk a kezelő fájl nevét is a statikus Start függvény paramétereként.

```
Process.Start("WinWord.exe","Munkakor.doc");
```

Fordítás/Futtatás/Tesztelés

Itt is kihasználtuk, hogy a WinWord.exe benne van az elérési útban, és a doc fájl az aktuális könyvtárban. Egyébként a teljes elérési útvonallal kell megadnunk a fájlneveket.

Ha ez utóbbi esetben a megadott fájl nincs meg (vagy más néven van ott), a Word elindulása után a következő hibaüzenetet kapjuk:



6. Gyakorlat. 3. ábra A hibaüzenet szövege

Láthatóan a WinWord lekezelte a kivételt, így a mi kivételkezelőnk nem hívódott meg. Ha azonban maga a WinWord.exe nincs telepítve, vagy nincs az elérési útban, akkor szükséges a kivételkezelésünk.

A programunkból indított Word alkalmazást használata után a felhasználónak kell leállítani. Ha azonban az általunk indított alkalmazást le is akarjuk állítani a programból, akkor szükségünk lesz egy objektumra, amin keresztül elérhetjük.

6.1.2. A LinkLabel vezérlő használata

Helyezzünk egy LinkLabel vezérlőt a listadoboz fölé!

Properties

Text Hirdetés

A LinkArea tulajdonság hárompontos gombját választva megjelenik egy ablak, ahol beállíthatjuk, a szöveg melyik részéhez szeretnénk csatolni. Alapértelmezésben a linket jelző szín kék (linkColor), aláhúzott (LinkBehavior SystemDefault). Az éppen feltöltő link piros (ActiveLink), a már látogatott link bordó (VisitedLinkColor).

6.1.3. A folyamat indítása objektumból

A dokumentumok elérésének másik egyszerű módja, ha az adott dokumentumot kezelő alkalmazást egy Process objektumból indítjuk el. Az indításkor egyszerűen

tulajdonság beállításával megadhatjuk, hogy az alkalmazás mely dokumentumot nyissa meg indításakor.

Tegyünk fel egy Process komponenst a ToolBox Components lapjáról!

Name: iexplorerProcess

StartInfo

FileName: Az iexplore.exe elérési úttal az adott gépen, pl.:
C:\Program Files\Internet Explorer\IEXPLORE.EXE
Arguments: A megjeleníteni kívánt honlap helye.

Ha nincs Internetelérésünk, akkor egyszerűen a Visual Studio File / New /File / HTML Page / Open segítségével létrehozhatjuk a fájlt, melyet Hirdetes.htm néven elmenthetünk. Ezután az Arguments tulajdonságba a Hirdetes.htm fájlt adjuk meg a teljes elérési útvonalával együtt! Itt ugyanis a fájlt nem a mi alkalmazásunk, hanem az iexplore.exe keresi, és neki nem a mi Debug könyvtárunk az aktuális könyvtár!

Megjegyzés:

Abban az esetben, ha az indítani kívánt dokumentumhoz hozzá van rendelve az őt kezelő alkalmazás, itt is megtehetjük, hogy a FileName tulajdonságba egyszerűen pl. egy .txt fájl nevét írjuk. Ilyenkor többnyire a Notepad indul el a megadott txt fájlt megnyitva, htm esetén a böngésző, doc esetén a Word...
Nézzük meg, hogy a `Windows 'Form Designer generated code'` szakaszban az `'InitializeComponent()'` metódus kódjában a beírt `iexplorerProcess.StartInfo.FileName` és `Arguments` tulajdonságaiban az útvonal dupla \\ jeleket tartalmaz, a C nyelvekben ugyanis a '\' sztringekben vezérlőkarakter. \n pl. nem n betűt, hanem soremelést jelent.

Most már csak a LinkLabel választásra azt kell mondanunk:

```
private void advertiseLinkLabel_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
 try
 {     iexplorerProcess.Start();     }
 catch(Exception exception)
 {     MessageBox.Show(exception.Message); }
}
```

Fordítás/Futtatás/Tesztelés

Természetesen itt is szükséges a kivételkezelés, mert bár az Interenet Explorerer kezeli a hibás honlap vagy fájl paramétert, de ha nincs a gép elérési útjában az iexplore.exe, akkor kezeletlen kivétel keletkezne.

A Process objektum Startinfo.FileName tulajdonságába írt kiterjesztéstől függően változik a Verb tulajdonság listája. Ha pl. egy doc kiterjesztésű fájlt indítunk a folyamatunkból, a Verb tulajdonság listáján választhatjuk a Print értéket, amely esetén nem megnyitja, hanem kinyomtatja a kívánt dokumentumot alkalmazásunk.

Húzzunk új Process vezérlőt az ablakunkra, neve legyen excelProcess! Állítsuk a FileName tulajdonságát Salary.xls-re (melyet az Excelből létrehoztunk és a Debug könyvtárba elmentettünk), Verb tulajdonságát pedig Print-re!

```
private void salaryButton_Click(object sender,
                                        System.EventArgs e)
{
 try   //Kivételkezelés, ha nem lenne Excel a gépen.
 {     excelProcess.Start();   }
 catch(Exception exception)
 {     MessageBox.Show(exception.Message); }
}
```

Fordítás/Futtatás/Tesztelés

A gomb választására az xls fájlt kinyomtatja alkalmazásunkat az alapértelmezett nyomtatón.

Ha a Word (vagy Excel) hívását ismételjük (Munkaköri leírás gomb), az alkalmazás csak aktivizálja a már megnyitott fájlt. Ez a WinWord beépített szolgáltatása. Ha azonban az Internet Explorert (Hirdetés link) hívjuk meg többször, az minden alkalommal új és új böngészőt nyit.

6.1.3.1. Csak egy példány fusson!

Ha nem akarjuk, hogy több példányban induljon el, szükségünk lesz még egy logikai változóra (isExplorerProcess), mely megmondja, elindítottuk-e már a böngészőt. Kezdőértéke legyen false, és indításkor állítsuk true-ra!

```
private bool isExplorerProcess = false;
private void advertiseLinkLabel_LinkClicked(object sender,
 System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
 if (!isExplorerProcess)
 {
```

```
        try
        {
            iexplorerProcess.Start();
            isExplorerProcess = true;
            advertiseLinkLabel.LinkVisited = true;
        }
        catch(Exception exception)
        {       MessageBox.Show(exception.Message); }
    }
}
```

Fordítás/Futtatás/Tesztelés

6.1.3.2. Visszajelzés leállításkor

Ha munka közben a felhasználó leállítja a böngészőt, a logikai változónknak is mutatni kell a változást! Ahhoz, hogy üzenetet tudjon küldeni a folyamat, az **EnableRaisingEvents** tulajdonságát igazra kell állítanunk.

Rendeljünk kezelő metódust a folyamat **Exited** eseményéhez!

```
private void explorerProcess_Exited(object sender,
                                    System.EventArgs e)
{
    isExplorerProcess = false;
}
```

Fordítás/Futtatás/Tesztelés

A link szöveg színe is módosítható, ha ezzel kívánjuk jelezni, a háttérben meg van-e nyitva az alkalmazás.

Megjegyzés:

A Process osztály rendelkezik egy **HasExited** tulajdonsággal, mely segítségével a már elindított folyamatról megtudhatjuk, hogy leállt-e. A mi esetünkben használatára nem volt szükség.

6.1.3.3. Kilépéskor álljon le az elindított folyamat is!

```
private void MunkaForm_Closing(object sender,
             System.ComponentModel.CancelEventArgs e)
{
 if (isExplorerProcess)
    iexplorerProcess.CloseMainWindow();
}
```

Fordítás/Futtatás/Tesztelés

! Ha az Internet Explorer (iexplore.exe) helyett a fájlkezelőt (explorer.exe) indítjuk, az is megtalálja a kiterjesztés alapján az indítani kívánt fájlt, de leállítani már nem tudja.

Megjegyzés:

A **CloseMainWindow** tagfüggvény helyett használhattuk volna a folyamat leállítására a Kill metódust is. Míg a CloseMainWindow elküldi a folyamat üzenetsorába a leállításhoz szükséges üzenetet, és így a szokásos leállást produkálja, a **Kill** azonnali befejezést jelent, abnormális programleállást így adatvesztéssel járhat.
Olyan folyamatok esetén, melyeknek nincs ablakuk, a Kill az egyetlen módja a folyamat leállításának.

6.2. Az időzítő kezelése

Tegyünk a képernyőre egy gombot, mely rövid időközönként változtatja helyét! A felhasználó feladata a gomb 'elkapása', a gombon történő kattintás. A gomb neve legyen catchButton, felirata: Kapj el!

Húzzunk a formra a ToolBox WindowsForms eszközei közül egy Timer komponenst! A neve legyen catchTimer! Ha fél másodpercenként szeretnénk az üzenetküldést, Interval tulajdonságát állítsuk 500-ra!

A gomb új helyének meghatározásához használjuk a véletlen szám generátort!

ClassView

CatchForm jobbegér

Add

Add Field

Field access: private

Field type: Random

Field name: rand

Finish

A form konstruktorában adjunk neki kezdőértéket, és indítsuk el az időzítőt! A gombon való kattintáskor állítsuk le az időzítőt!

```
public CatchForm()
{
 InitializeComponent();
 catchTimer.Start();
 rand = new Random();
}

private void catchButton_Click(object sender,
                                        System.EventArgs e)
{
 catchTimer.Stop();
}
```

Megjegyzés:

A timer az Enabled tulajdonságának true-ra állításával is indítható.

A catchTimer változón duplán kattintva, vagy catchTimer a Properties ablak Events gombját választva a Tick eseményhez kezelőt rendelve mozgassuk a gombot!

```
private void catchTimer_Tick(object sender, System.EventArgs e)
{
 catchButton.Left = rand.Next(Width-catchButton.Width);
 catchButton.Top = rand.Next(Height-catchButton.Height);
}
```

Fordítás/Futtatás/Tesztelés

Azt tapasztaljuk, hogy időnként a gomb az ablak alján épp hogy látható. Mivel az ablak magasságába beletartozik a címsor is, így hiába vontuk ki a magasságból a gomb magasságát, még a címsor és a keret vastagságát is ki kellene vonni. Akkor már egyszerűbb a kliensterület méretével dolgozni.

```
private void catchTimer_Tick(object sender, System.EventArgs e)
{
 catchButton.Left = rand.Next(this.ClientSize.Width-
                                        catchButton.Width);
 catchButton.Top = rand.Next(this.ClientSize.Height-
                                        catchButton.Height);
}
```

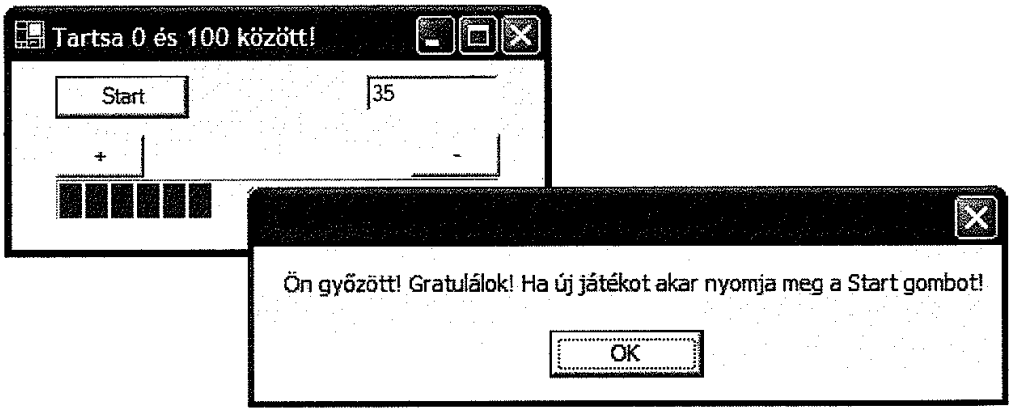
Fordítás/Futtatás/Tesztelés

Most már a teljes gomb a kliensterületen belül mozog, méghozzá az ablakméret változásával kihasználja a rendelkezésre álló teret. Ha elég kicsire választjuk az ablak méretét, el tudjuk kapni a gombot, ezzel tesztelve az időzítő leállítását.

A játék még látványosabb, ha az egérkurzort a formon és a gombon is kéz alakúra állítjuk (Properties/Cursor/Hand), és a gomb valamilyen képet mutat.

6.3. Feladatok

1. Készítsük el a 'Munka' alkalmazásunkat úgy, hogy a megtekinthető dokumentumokat egy menüből lehessen kiválasztani! Itt mód nyílik arra is, hogy egy almenüből több azonos típusú, de különböző tartalmú dokumentum választását felkínáljuk.
2. Indítsunk el a programból egy PowerPoint bemutatót, kezeljük a lezárását is!
3. **Futó alkalmazások**. Készítsünk alkalmazást, mely egy ablakban kilistázza az adott nevű éppen futó alkalmazásokat! Egy gomb választás hatására az összes adott nevű alkalmazás álljon le!*
4. **Fényújság**. Készítsen egy mezőt az ablakban, melyben egy szöveg jobbról balra folyamatosan mozog!
5. Az alkalmazás ablakában egy szövegmező és egy folyamatkijező mutassa az indítás óta eltelt időt! Több folyamatkijelzőnk is lehet. Pl. egyik 10 perces időtartamot jelez, a másik az eltelt 10 perceket pl. 25 perc az 2 egység az egyik, míg 5 a másik kijelzőn.
6. **Folyamatkijezős játék**. Legyen az ablakban egy folyamatkijelző egy '+' és egy '–' gomb. A progressbar 0 és 100 közötti értékeket mutat. Kezdetben épp 50-et. A '+' gombbal +10 a '–' gombbal -10 értékkel módosíthatjuk értékét. A háttérben egy időzítő 0,5 másodpercenként hozzáad vagy levon egy 0 és 30 közötti számot. Ha a kijelző eléri a 0 vagy a 100 értéket, a gép nyert, ha a játékos bírja 10 másodpercig, akkor a játékos nyert. Egy szövegmező mutassa a folyamatkijelző aktuális értékét! A játék egy Start gomb segítségével indítható, újraindítható.*



6. Gyakorlat. 4. ábra A játékos győzelme

7. A progressbaros játék továbbfejlesztése úgy, hogy be lehessen állítani az időzítő gyorsaságát, a '+' és '–' gombok által hozzáadott, levont értéket és a Maximális értéket és egy menet idejét!

6.4. Megoldásötletek

3. Futó alkalmazások

Ha csak a felhasználói felülettel rendelkező folyamatokat kívánjuk kilistázni, akkor, mivel nekik van főablakuk és annak címsora, a legtöbb információt e címsor szövegéből kapjuk. A frissítés (refresh) gomb hatására a processListBox listadobozba kilistázza az éppen futó processek címsorát, ha az nem üres.

```
private void RefreshButton_Click(object sender,
                                        System.EventArgs e)
{
 processListBox.Items.Clear();
 foreach (Process p in Process.GetProcesses())
   if (p.MainWindowTitle!="")
       processListBox.Items.Add(p.MainWindowTitle);
}
```

A valóságban az itt felsoroltnál jóval több folyamat fut. A többségnek vagy nincs ablaka, vagy a főablak címsora üres. Ezt megnézhetjük a Tools menüpont **Spy++** almenüpontját választva, majd a Processes eszközgombra kattintva. Ha ezek mindegyikét ki akarjuk listázni az ismétlődő System.Diagnostics.Process és a zárójelek nélkül:

```
foreach (Process p in Process.GetProcesses())
{
 str=p.ToString();
 str=str.Remove(0,28); // System.Diagnostics.Process (
 str=str.Remove(str.Length-1,1); // )
 processListBox.Items.Add(str);
}
```

A Notepad gombon kattintva bezárja az összes Jegyzettömb alkalmazást.

```
private void NotepadButton_Click(object sender,
                                        System.EventArgs e)
{
 Process[] p=Process.GetProcessesByName("Notepad");
 for (int i= p.Length-1; i>-1; i--)
   p[i].Kill();
}
```

A folyamatok leállításához itt azért használtuk a Kill metódust, mert ha valamilyen okból – pl. internet honlapot keresünk, de nincs internet kapcsolat, vagy épp meg akarunk nyitni a Jegyzettömbben egy fájlt – egy modális ablak van éppen kinyitva, akkor is leállítja a folyamatot a CloseMainWindow-val ellentétben.

Megjegyzés:

Ha a lezárásnál a GetProcesses tagfüggvényt hívtuk volna, az mindent visszaad, így metódusunk mindent leállítana, még a Windowst is!

6. Folyamatkijelzős játék

A feladathoz két időzítő kell: timer változtatja az értéket, Interval tulajdonsága 500, a timeTimer méri a játékidőt, Interval tulajdonsága 10 000.

A megjelenített értéket célszerű egy int típusú adattagban (val) tárolni, hisz állandóan számolnunk kell vele. A véletlen szám hozzáadáshoz kell egy véletlen szám generátor (rand).

```
private Random rand;
private int val;

public ProgressForm()
{
 InitializeComponent();
 val = 50;
 textBox.Text=val.ToString();
 rand=new Random();
}
```

A folyamatkijelző ha túlcsordul vagy alulcsordul, azaz ha a maximális értéknél többet, vagy a minimális értéknél kevesebbet vesz fel, kivételt dob. A túlcsordulást és alulcsordulást egy önálló tagfüggvényben (OverFlow) vizsgáljuk, mert a timer és a két (+, –) gombon kattintás eseménykezelője is módosítja az értéket, így mindhárom esetben vizsgálni kell. Ilyenkor győz a gép.

Célszerű a mindhárom esetben ismétlődő egyéb tevékenységeket pl. textBox-ba kiírás is itt megvalósítani a kódtöbbszörözés elkerülése érdekében.

```
private void plusButton_Click(object sender,
                                    System.EventArgs e)
{
 val+=10;
```

```
 OverFlow(val);
}

private void startButton_Click(object sender,
                                        System.EventArgs e)
{
 timer.Start();
 timeTimer.Start();
 textBox.Text=val.ToString();
 progressBar.Value=val;
}

private void timer_Tick(object sender, System.EventArgs e)
{
 val += rand.Next(60)-30;
 OverFlow(val);
}

public void OverFlow(int val)
{
 textBox.Text=val.ToString();
 if (val>100 || val<0)
 {
    timer.Stop();
    timeTimer.Stop();
    MessageBox.Show("Sajnos a gép győzött! Próbálja újra!");
    val = 50;
 }
 else
    progressBar.Value=val;
}
```

Ha letelt a 10 másodperc, és nem állt meg a timeTimer, akkor el tudja küldeni az üzenetét, s így a játékos győzött.

```
private void timeTimer_Tick(object sender, System.EventArgs e)
{
 timer.Stop();
 timeTimer.Stop();
 MessageBox.Show("Ön győzött! Gratulálok! Ha új játékot
                  akar, nyomja meg a Start gombot!");
}
```

7. Gyakorlat. A komponensek felhasználása

Az alkalmazásunkkal párhuzamosan szeretnénk futtatni egy már megírt alkalmazást oly módon, hogy nemcsak általánosan mint folyamatról szeretnénk információt kapni, hanem az alkalmazással kapcsolatos konkrét utasításokat szeretnénk küldeni neki. A Microsoft rendelkezésünkre bocsátja a gépünkre telepített office alkalmazások komponenseit.

7.1. Számlaírás, nyomtatás

A 'Vendéglő a Világ Végén' feladatban a vendég számlát szeretne kérni a fogyasztásáról. Ehhez tegyünk egy Számla feliratú billButton nevű gombot az ablakra!

7.1.1. A Word indítása a programból

Solution Explorer

References jobbegér

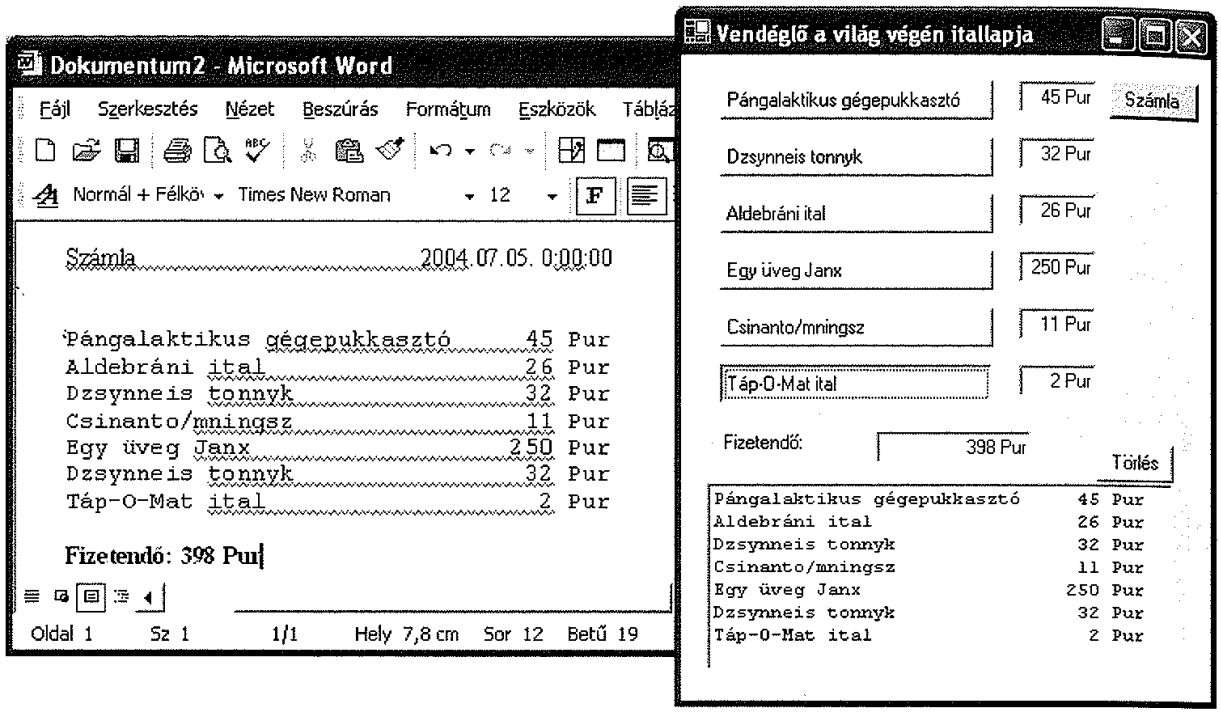
Add Reference

COM fül

Microsoft Word X.X Object Library (a verziószám a gépre telepített Word verziójától függ)

Select

OK



7. Gyakorlat. 1. ábra A számla a Wordbe írva

A választás hatására a References listában négy új komponens jelent meg, közöttük a Word, mely az Interop.Word.dll-ben van leírva. A **privát assembly**ket — Interop.Microsoft.Office.Core.dll, az Interop.VBIDE.dll és az Interop.Word.dll — a Visual Studio alapértelmezésben (Copy Local = True) a könyvtárunkba másolja. Míg az stdole.dll **osztott assembly**t a **GAC**-ből hivatkozza le az alkalmazás. (Solution Explorer / References / stdole / Properties.) Keressük meg őket a könyvtárakban!

```
private void billButton_Click(object sender,
                              System.EventArgs e)
{
 Word.Application word = new Word.Application();
 word.Visible = true;    // Ha látni akarjuk.
}
```

Fordítás / Futtatás

! A Word névteret ki kell írnunk, mert nem adtuk meg a using direktíva mögött. Itt ne is tegyük, mert ha beírjuk: `using Word;` akkor az alkalmazás Main metódusában hívott Application osztályról a fordító nem tudja eldönteni, hogy a System.Windows.Forms névtérben vagy a Word névtérben megadott deklarációt használja. `'Application' is an ambiguous reference` hibaüzenetet kapunk.

7.1.2. Írás a Wordbe a programból

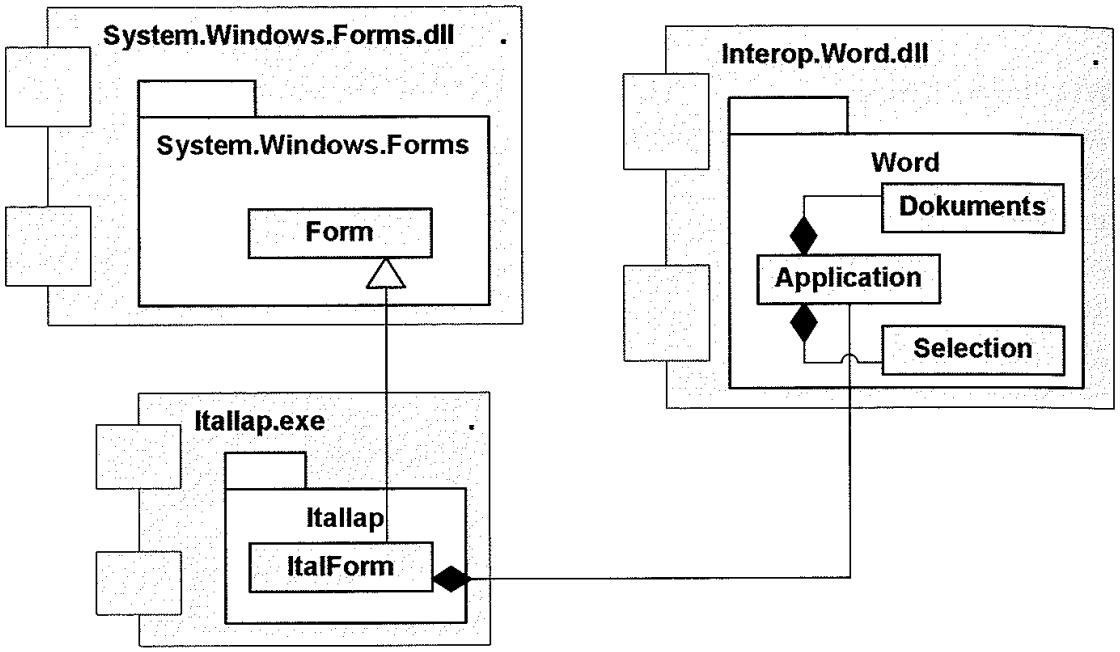
A Wordbe új dokumentumot a Documents.Add függvénnyel adhatunk, mely négy paramétert vár. A C# függvények nem ismerik az alapértelmezett paraméter fogalmát, viszont egy speciális paraméterrel helyettesíthetjük a hiányzó argumentumokat. A Missing osztály Value statikus adattagja lehetővé teszi, hogy a hiányzó paramétereket helyettesítsük vele. A Missing osztály a System.Reflection névtér tagja.

```
using System.Reflection;

private void billButton_Click(object sender,
                              System.EventArgs e)
{
 Word.Application word = new Word.Application();
 word.Visible = true;    // Ha látni akarjuk.
 object m = Missing.Value;
 word.Documents.Add(ref m,ref m,ref m,ref m);
}
```

Az íráshoz egy Selection objektum szükséges, melyen szükség esetén beállíthatjuk a felhasznált font tulajdonságait. Írni a TypeText metódussal, sort emelni a TypeParagraph metódussal, vagy a szövegben szokásos \n-el lehet. Ezt itt felváltva használjuk.

```
private void billButton_Click(object sender,
                              System.EventArgs e)
{
 Word.Application word = new Word.Application();
 word.Visible = true;    // Ha látni akarjuk.
 object m = Missing.Value;
 word.Documents.Add(ref m,ref m,ref m,ref m);
 Word.Selection selection = word.Selection;
 selection.TypeText("Számla \t\t\t\t"+
                    DateTime.Today.ToString());
 selection.TypeParagraph();
```

```
 selection.TypeParagraph();
 for (int i=0; i<italokBox.Items.Count ;i++)
 {
   selection.TypeParagraph();
   selection.TypeText(italokBox.Items[i].ToString());
 }
 selection.Font.Bold = 1;
 selection.TypeText("\n\nFizetendő: "+fizBox.Text);
}
```



7. Gyakorlat. 2. ábra A felhasznált főbb komponensek, névterek és osztályaik

Fordítás / Futtatás / Tesztelés

Az egyes tételek árai most sincsenek egymás alatt, de most már tudjuk, hogy ez a proporcionális fontszélesség miatt van. Ha Courier New fontot választunk a tételek kiírásához, a probléma megoldódik.

```
selection.Font.Name = "Courier New";
for (int i=0; i<italokBox.Items.Count ;i++)
{
 selection.TypeParagraph();
 selection.TypeText(italokBox.Items[i].ToString());
}
selection.Font.Name = "Times New Roman";
```

Megjegyzés:

A kiíratásnál a fontot még animálni is lehet, a lehetőségekről a legördülő súgó tájékoztat. Pl. a következő beállítás villogtatja a mögötte kiírt karaktereket.

```
selection.Font.Animation                                    =
      Word.WdAnimation.wdAnimationBlinkingBackground;
```

Fordítás / Futtatás / Tesztelés

Ne készítsünk számlát! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd ne zárjuk be! Lépjünk ki az alkalmazásból! ***FUT TOVÁBB.***

Indítsuk a számlát, majd zárjuk be! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd zárjuk be! Indítsuk újra a számlaírást! ***JÓ.***

Indítsuk többször a számlaírást! Lépjünk ki az alkalmazásból! ***FUTNAK TOVÁBB.***

Ha kilépünk az alkalmazásból, azt szeretnénk, ha az általunk nyitott Word ablakok is bezáródnának. Bezáráskor a főablak Closing eseménye hívódik meg, itt kellene bezárni az ablakokat. Azonban most a word objektumunk a billButton_Click függvény lokális objektuma, más tagfüggvény nem láthatja.

Tehát először az alkalmazás minden tagfüggvénye számára hozzáférhetővé kell tennünk a word objektumot. Vagyis tagváltozóként kell felvennünk. Az inicializálást végezzük a konstruktorban!

```
private Word.Application word;

public ItalForm()
{
 InitializeComponent();

 word = new Word.Application();
 word.Visible = false;
}
```

Ezután csak láthatóvá kell tennünk a billButton_Click elején.

Fordítás / Futtatás / Tesztelés

Bár valóban láthatóvá válik, mégis a háttérben marad. Aktívvá is kell tennünk!

```
word.Visible = true;      // Ha látni akarjuk.
word.Activate();
```

7.1.3. A párhuzamosan megnyitott alkalmazást zárjuk be kilépéskor!

Ezután már leállíthatjuk alkalmazásunkat.

```
private void ItalForm_Closing(object sender,
             System.ComponentModel.CancelEventArgs e)
{
 object m = Missing.Value;
 word.Quit(ref m, ref m, ref m);
}
```

Fordítás / Futtatás / Tesztelés

Ne készítsünk számlát! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd ne zárjuk be a Word ablakot! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd ne zárjuk be a számlát! Fókuszáljunk egy másik alkalmazásra, majd vissza a Vendéglőhöz! Lépjünk ki az alkalmazásból! *NEM VESSZÜK ÉSZRE, HOGY RÁKÉRDEZ, MENTSE-E AZ ADATOKAT.* ***A HÁTTÉRBEN MARAD.***

Indítsuk a számlát, majd zárjuk be! Lépjünk ki az alkalmazásból! ***HIBA:*** *Az RPC-kiszolgáló nem érhető el. A Quit hívásánál.*

Indítsuk a számlát, majd zárjuk be! Indítsuk újra a számlaírást! ***HIBA:*** *Az RPC-kiszolgáló nem érhető el. Az Activate()-nél.*

Indítsuk többször a számlaírást! Lépjünk ki az alkalmazásból! ***JÓ.***

A hibák tehát lényegében két helyzetben lépnek fel, ha már bezártuk az alkalmazást és újra aktivizálni akarjuk, vagy ha újra be akarjuk zárni. Vagyis nem működik az alkalmazás elején megnyitom a végén zárom elv, mert a felhasználó is bezárhatja. Tehát, ha szükségünk van rá, akkor kell megnyitni (ez hatékonyabb is)! Az ablak szükség esetén történő létrehozása visszakerül a billButton_Click metódusba.

```
private void billButton_Click(object sender,
                                        System.EventArgs e)
{
 if (word == null)       //Ha még nem hoztuk létre.
 {
   word = new Word.Application();
 }
```

```
try   //Ha már bezártuk, akkor újra létre kell hozni.
{
  word.Visible = true; // Ha látni akarjuk.
}
catch
{
  word = new Word.Application();
  word.Visible = true; // Ha látni akarjuk.
}
word.Activate();  // Ha felül akarjuk látni.
...
}
```

Az elején a word == null vizsgálat akár fölösleges is lehet, hisz ha még nem hoztuk létre, akkor ugyanúgy kivételt dob a következő sorban, mint amikor bezártuk. A kérdés az, hogy kivételes esetnek tekintjük-e a létre nem hozást? Ha igen, akkor hagyjuk el az első elágazást! A bezárást így valósítjuk meg.

```
private void ItalForm_Closing(object sender,
             System.ComponentModel.CancelEventArgs e)
{
 try
 {
   word.Visible=true;
   word.Activate();
   object m = Missing.Value;
   word.Quit(ref m, ref m, ref m);
 }
 catch {};
}
```

A catch blokk üres, hisz ha nincs word, akkor nem kell kilépni belőle.

Fordítás / Futtatás / Tesztelés

Ne készítsünk számlát! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd ne zárjuk be a Word ablakot! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd ne zárjuk be a számlát! Fókuszáljunk egy másik alkalmazásra, majd vissza a Vendéglőhöz! Lépjünk ki az alkalmazásból!

HA A MÁS ALKALMAZÁS EGY MÁSIK WORD ABLAK, AKKOR FÖLÉ KERÜL, EGYÉBKÉNT SÁRGÁN FIGYELMEZTETVE VILLOG A START TÁLCÁJÁN A WORD IKONJA. ***JÓ.***

Indítsuk a számlát, majd zárjuk be! Lépjünk ki az alkalmazásból! ***JÓ.***

Indítsuk a számlát, majd zárjuk be! Indítsuk újra a számlaírást! ***JÓ.***

Indítsuk többször a számlaírást! Lépjünk ki az alkalmazásból! ***JÓ.***

7.1.4. A nyomtatási kép

Ha csak a nyomtatási képét szeretnénk látni, akkor kezdetben legyen a Visible tulajdonsága false, majd a kódrészlet végén:

```
selection.Document.PrintPreview();
word.Visible=true;
```

Fordítás / Futtatás / Tesztelés

7.1.5. Háttérben nyomtatás

Ha pedig úgy akarjuk kinyomtatni, hogy közben ne nyíljon ki a word, akkor a word.Visible = false; sor legyen az elején, a végén pedig meghívjuk a PrintOut függvényt csupa Missing.Value paraméterrel:

```
selection.Document.PrintOut(ref m,ref m,ref m,ref m,ref
m,ref m,ref m,ref m,ref m,ref m,ref m,ref m,ref
m,ref m,ref m,ref m,ref m);
```

Már csak annyi a feladat, hogy a 'Vendéglő a Világ Végén' feliratot, és egy párbeszédablakba bekérve a vevő adatait is kiírjuk a számlára. Ezt már az olvasó az eddig tanultak alapján meg tudja valósítani.

Fordítás / Futtatás / Tesztelés

7.2. Aláírásgyűjtés digitálisan

Képzeljük el, hogy egy TabletPC segítségével gyűjtünk aláírásokat egy petícióhoz! Az aláírások feldolgozása — feltéve, hogy adatbázisban tároljuk őket— sokkal egyszerűbb. Az ismétlődések és a nem létező személyek kiszűrése egyszerűen megoldható. Az ily módon rendelkezésünkre álló aláírások száma azonnal megtudható. Mi most e feladatban a felhasználói felület elkészítésével foglalkozunk. Mivel nem áll rendelkezésünkre aktív toll és TabletPC, ezért aláírni az egérrel tudunk, de az alkalmazást TabletPC-re másolva aktív tollal is működik.

- A letölthető forráskód működéséhez is szükséges a Tablet PC Platform SDK. Letöltése nélkül eltűnik a Sign alkalmazás [Design] nézetéből és a lefordított alkalmazásból is az InkPanel vezérlő.

7.2.1. Honlapról letöltött komponens használata

Töltsük le a Microsoft honlapról a Tablet PC Platform SDK-t! Ez

www.microsoft.com

Search Microsoft.com for: Tablet PC Platform SDK **GO**

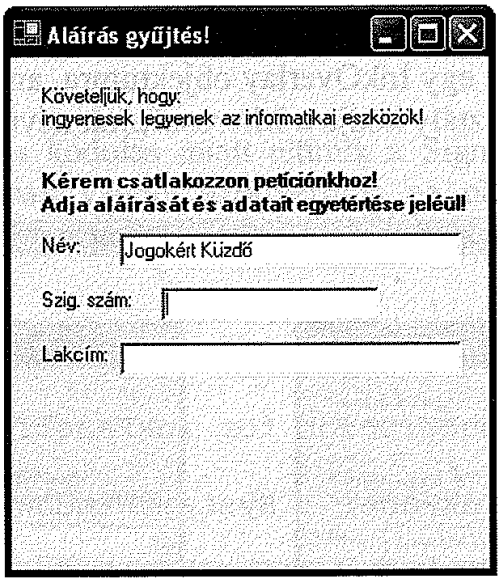
Download details: Tablet PC Platform Software Development Kit (SDK)

Méret az 1.5 verzió esetén: 11020 KB

Download

A letöltött telepítőt elindítva, telepíthetjük az SDK-t. A telepítés után a Start menü / Microsoft Tablet PC Platform SDK alatt részletes dokumentációt és mintaprogramokat találunk.

Készítsük el az aláírásgyűjtő (Sign nevű) alkalmazásunk párbeszédfelületét!



7. Gyakorlat. 3. ábra Az aláírásgyűjtő felhasználói felülete

A felület aljára egy olyan panelt teszünk, amin alá tud írni a jelentkező. Ehhez a Microsoft.Ink.dll-re lesz szükségünk.

Solution View

References jobbegér

Add Reference

.NET fül

Microsoft Tablet PC API

Select

OK

Ha a Visual Studiot még a komponens telepítése előtt indítottuk, előfordulhat, hogy nem találjuk az ablakban a komponenst. Ekkor újra kell indítanunk a Visual Studiot.

7.2.2. Az írható panel osztály elkészítése

A ToolBoxban rendelkezésünkre áll egy Panel, de mi azt szeretnénk, ha írni is lehetne rá! Ehhez származtassunk egy utódosztályt belőle!

Solution View

Sign projekt, jobbegér

Add

Add class

Name: InkPanel.cs

Open

A kódban szükségünk lesz egy InkOverlay objektumra, ami lehetővé teszi az írást a panelen. Az InkOverlay osztály kódja a Microsoft.Ink névtérben szerepel.

```
using System;
using Microsoft.Ink;

namespace Sign
{
  /// <summary>
  /// Summary description for InkPanel.
  /// </summary>
  public class InkPanel : System.Windows.Forms.Panel
  {
    private Microsoft.Ink.InkOverlay inkOverlay;
    public InkPanel()
    {
      inkOverlay = new InkOverlay(this.Handle);
      inkOverlay.Enabled = true;
    }
  }
}
```

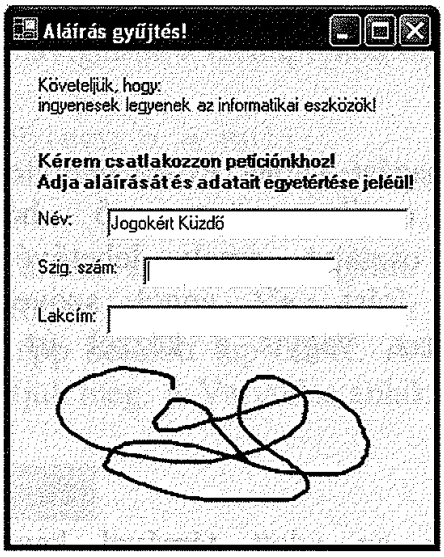
Ezután húzzunk egy Panel objektumot a formunkra, és a forráskódban állítsuk át az osztályát Panel-ről InkPanelre! A kódot a SignForm osztály adattagjai közt és az `InitializeComponent`-ben a konstruktor hívását módosítsuk:

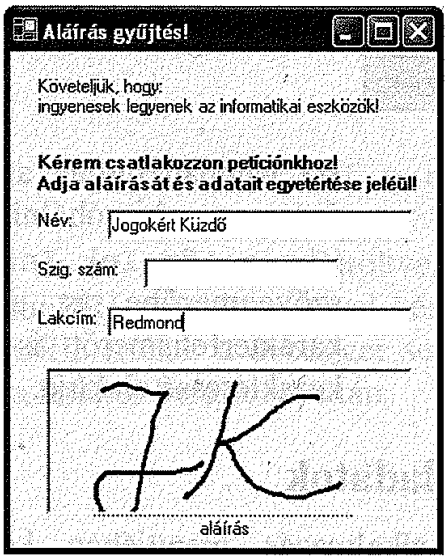
```
public class SignForm : System.Windows.Forms.Form
{
 private System.Windows.Forms.TextBox cimTextBox;
 private InkPanel inkPanel;
...
 private void InitializeComponent()
 {
    this.cimTextBox = new System.Windows.Forms.TextBox();
    this.inkPanel = new Sign.InkPanel();
    this.SuspendLayout();
...
 }
...
}
```

Fordítás / Futtatás / Tesztelés

Megjegyzés:

Ha a hibás kódolás miatt eltűnik a SignForm Design nézetéből a panel, annak valami szoftverhiba az oka. A legegyszerűbben a kódban történt javítások átnézésével (visszagörgetésével) hozhatjuk helyre.

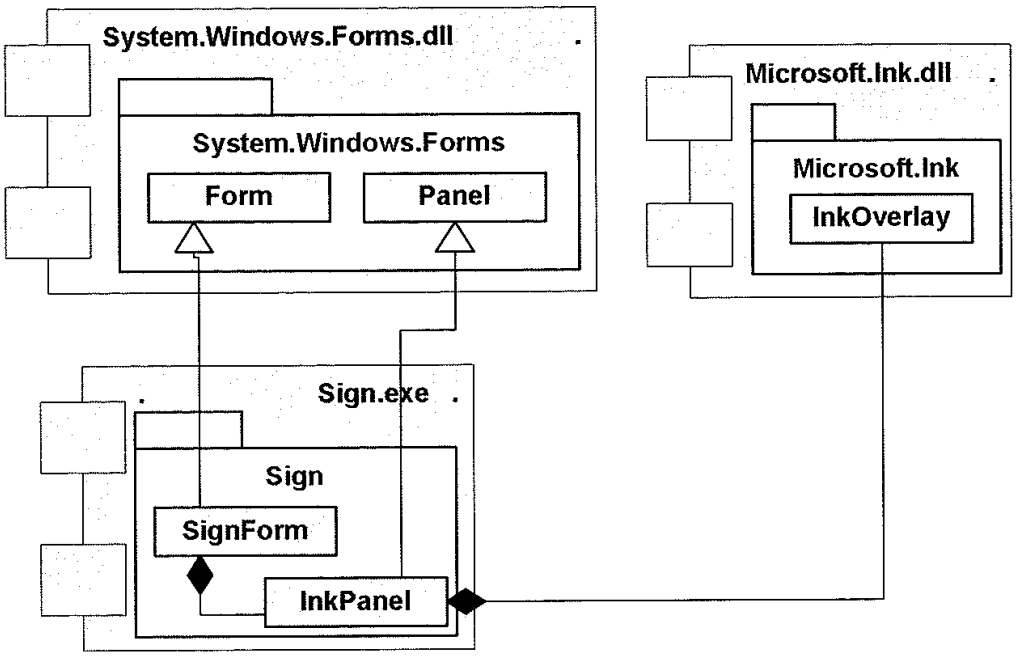




7. Gyakorlat. 4. ábra Az aláírásgyűjtő felhasználói felülete az inkPanellel

Szeretnénk jelölni a jelentkező számára, hol tud aláírni!

Tegyünk egy aláírás feliratot az inkPanel alá — úgy méretezzük, hogy kétsoros legyen és TextAlign BottomCenter —, és állítsuk a panel BorderStyle tulajdonságát Fixed3D-re!



7. Gyakorlat. 5. ábra Az aláírásgyűjtő főbb komponensei, névterei és osztályai

A Label osztályt nem tudjuk a TrancparencyKey tulajdonsággal átlátszóvá tenni, mert ez a Form osztály tulajdonsága. Viszont készíthetünk egy írható Label osztályt. Azt használva az aláírás szöveghez megtehetjük, hogy aláírásunk a pontozott vonal alá csússzon.

Megjegyzés:

Ha valóban tudjuk az alkalmazást Tablet PC-re telepíteni, akkor érdemes kipróbálni az egyik szövegmező helyett a `Microsoft.Ink.InkEdit` osztályt, amely lehetővé teszi a szövegmezőbe is a tollal írást, sőt angol szöveg esetén karakterfelismerőt is tartalmaz, vagyis a kézzel írt szöveget karakteressé alakítja.

7.3. Feladatok

1. Egy alkalmazás menüjében készítsünk az adott témával kapcsolatban tájékoztatást adó menüpontokat! Az információt különböző honlapokon érhetjük el. A feladatot Internet Explorer objektum segítségével oldjuk meg, melynek címét a menüpontok választásával állíthatjuk be!*
2. **Számlaírás mentése**. Mielőtt bezárnánk a Számlát készítő alkalmazásunkat, ne visszakérdezzünk, hanem kérdés nélkül mentsük el a háttérben még nyitva levő számlákat!*

3. Tegyünk egy gombot az Eladó alkalmazás felületére, mely lehetővé teszi számlanyomtatását MS Word felhasználásával!
4. **Italösszesítő Excelben.** Készítsünk az Itallap feladathoz egy háttér Excel táblázatot, mely folyamatosan tárolja az elfogyasztott italok számát és árát!*
5. Indítsunk programunkból egy PowerPoint bemutatót!
6. Indítsuk el, és kezeljük a programunkból a Windows Media Playert!
7. Valamely eddig elkészült alkalmazásunk hátterében játsszunk zenét a Windows Media Player segítségével! A hangot lehessen ki- és bekapcsolni!

7.4. Megoldásötletek

1. Navigálás az Internet Explorerben menüpontok választására

Az Internet Explorer az SHDocVw komponensben és névtérben van leírva, melyet az Add Reference ablak COM füle alatt találunk.

```
private SHDocVw.InternetExplorer ie;
```

Az ie indítása:

```
ie = new SHDocVw.InternetExplorer();
ie.Visible = true;
```

Az ie navigálása a menüpontok hatására:

```
Object o = null;
ie.Navigate("C:\\proba.html", ref o,ref o,ref o,ref o);
```

Csak akkor érdemes navigálni, ha nem az az oldal van betöltve, mert az újraletöltés időigényes lehet. Az ie.LocationURL tulajdonságával kérdezhetjük le az épp betölött lap címét. A LocationURL értékét a debugger segítségével nézhetjük meg. A Variables ablak mutatja a letöltés befejezése után.

Megjegyzés:

Régebbi verzióknál fájl esetén nem elég az elérési utat beírnunk, elé kell tenni a file nevet és a ///jelet.

```
if (ie.LocationURL != "file:///C:/proba.html")
        // Web cím esetén azonos a Navigate paraméterével.
```

Megjegyzés:

Lehetőség van a betöltés folyamatának figyelésére és kijelzésére egy folyamatkijelző segítségével. Azt is szabályozhatnánk, hogy csak akkor mutassa a linklabel színe, hogy kiválasztották, ha valóban letöltődött a fájl, ne minden esetben. Az is megoldható, hogy ha több helyen tárolt anyag az egyik helyen nem érhető el, keressük a másik helyen.

2. A számlaírás mentése

A dokumentumokat az utolsóval kezdve zárjuk be, így a sorszámokat is csökkenő sorrendben célszerű kiosztani.

```
private void ItalForm_Closing(object sender,
             System.ComponentModel.CancelEventArgs e)
{
 try
 {
    word.Visible=true;
    word.Activate();
    object m = Missing.Value;
    int i = word.Documents.Count;
    foreach (Word.Document doc in word.Documents)
    {
        object filename= "Számla"+i--;
        doc.SaveAs(ref filename,ref m,ref m,ref m,ref m,
              ref m, ref m, ref m, ref m,ref m, ref m, ref
              m,ref m,ref m,ref m,ref m);
    }
    word.Quit(ref m, ref m, ref m);
 }
 catch {};
}
```

Megjegyzés:

A Word a dokumentumait, Dokumentum1, Dokumentum2 ... névvel látja el, vagyis a számlálásukat a C típusú nyelvektől eltérően nem 0-val, hanem 1-gyel kezdi. Erre figyelnünk kell, ha ciklussal valósítjuk meg a mentést! Ráadásul az utoljára írt lesz az első.

```
for(int i =1 ;i<word.Documents.Count+1;i++)
{
        object filename= "Számla"+i;
        object o=(object)i;
        Word.Document doc = word.Documents.Item(ref o);
        doc.SaveAs(ref filename,ref m,ref m,ref m,ref m,
              ref m, ref m, ref m, ref m,ref m, ref m,
              ref m,ref m,ref m,ref m,ref m);
}
```

Mivel csak a számlák nevét adjuk meg, a fájlok a Word alapértelmezett könyvtárába, a Dokumentumokba mentődnek.

4. Italösszesítő Excelben

Az Excel komponens a COM lapon Microsoft Excel X.X Object Library név alatt érhető el.

Készítsünk egy Excel táblázatot az excelben, és mentsük a Dokumentumok könyvtárba ItallapFogyaszt.xls néven! Az Ösz Ár oszlop celláit és az összes fogyasztást képlettel adjuk meg! (=B3*C3; =SZUM(D3:D8))

	Egységár (Pur)	Fogyás	Összár (Pur)
Pángalaktikus gégepukkasztó	45	0	0
Dzsinneis Tonnyk	32	0	0
Aldebráni ital	26	0	0
Janx	250	0	0
Csinanto / mningsz	11	0	0
Táp-O-Mat ital	2	0	0
	Összes fogyasztás (Pur):		0

Ha nem akarjuk minden ital választásakor újra és újra nyitni és zárni az Excelt, akkor a program elején / végén tegyük ezt meg!

```
private Excel.Application excel;

public ItalForm()
{
 InitializeComponent();

 excel = new Excel.Application();
 excel.Visible = true;
 object m = Missing.Value;
 Excel.Workbook workb = excel.Workbooks.Open(
    "ItallapFogyaszt.xls",m,m,m,m,m,m,m,m,m,m,m,m,m);
}
```

```
private void ItalForm_Closing(object sender,
                System.ComponentModel.CancelEventArgs e)
{...
 //Excel
 try
 {     excel.Quit();     }
 catch {};
}
```

Készítsünk egy Italok nevű függvényt az ItalForm osztályba, mely növeli az italválasztáskor a cella tartalmát! (Class View / ItalForm / Add / Add Method)

```
public void Italok(int sor)
{
 Excel.Worksheet worksheet =
                    (Excel.Worksheet)excel.Worksheets[1];
 Excel.Range range = (Excel.Range)worksheet.Cells[sor,3];
 int db;
 db = Convert.ToInt32(range.Value2.ToString());
 worksheet.Cells[sor,3]=db+1;
}
```

Az ital választásakor ezt a függvényt hívjuk meg a megfelelő sorszámmal!

```
private void Button_Click(object sender,
                                        System.EventArgs e)
{…
case "Pángalaktikus gégepukkasztó":
 sar = panTextBox.Text.TrimEnd(penz);
 //Excel
 Italok(3);
 break;
 …
}
```

Ne feledjük még a törlés megvalósítását!

Ha több alkalmazásból egyszerre kívánjuk írni az excel fájlunkat, az már problémát jelent. Hisz ez a megvalósítás minden alkalmazásból új és új Excelt nyit benne az aktuálisan mentett dokumentummal. Az azonos alkalmazásból nyitást még meg lehetne valósítani, de ha úgy akarunk a dokumentumba írni, hogy az adat beolvasása óta valaki más már módosította annak értékét mire mi a módosított értéket írni akarjuk (concurrency violation), ez problémát okoz. Ilyen esetben javasolt az adatok – párhuzamos adatkezelést biztosító – adatbázis-kezelő alkalmazásban történő feldolgozása excel fájl helyett.

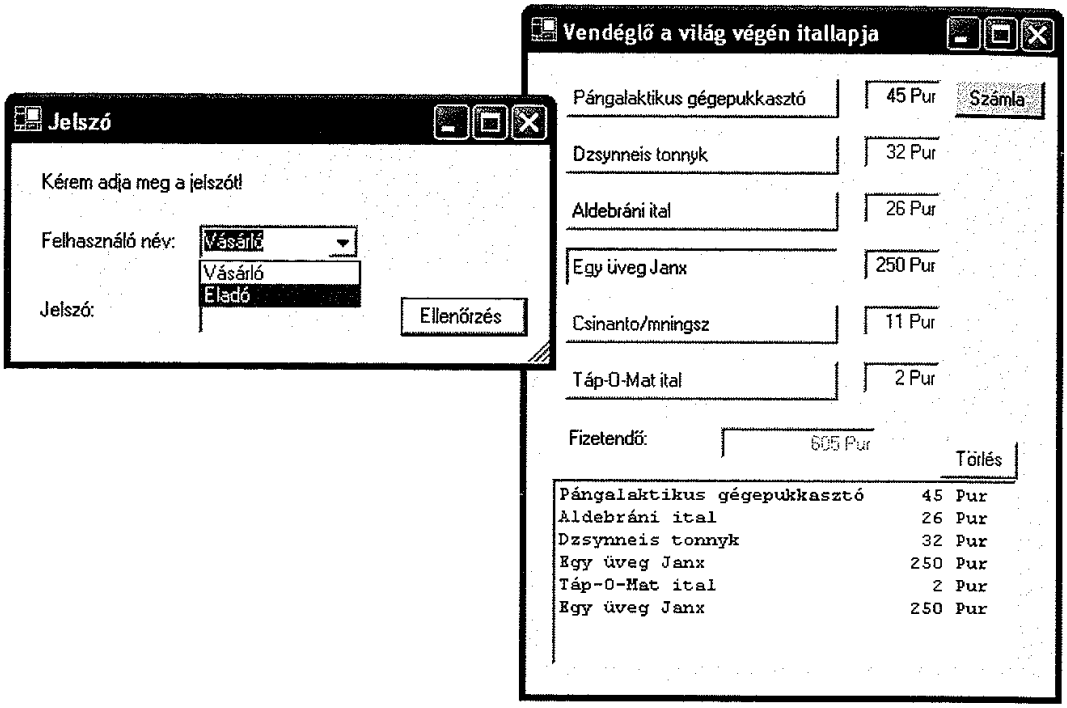
8. Gyakorlat. Osztálykönyvtár készítése

Az előzőekben megtanultunk osztálykönyvtárakat felhasználni, most készíteni szeretnénk egyet. Az osztálykönyvtár lényege, hogy több alkalmazás is használhatja. A 2. gyakorlaton készítettünk egy jelszóbekérő alkalmazást. A jelszóbekérés azonban nem egy önálló alkalmazás, hanem tipikusan a feltétele egy alkalmazás használatának.

Sok különböző alkalmazás indulhat jelszóbekéréssel. Természetesen a különböző alkalmazások használóinak jogosultságai eltérőek, de az elv mégis azonos, ahhoz, hogy elindíthassák az alkalmazást, vagy ahhoz, hogy futtatás közben bizonyos műveleteket elvégezhessenek, szükségük van a jogosultság ellenőrzésre, az azonosítójuk és a jelszavuk megadására. Tehát a jelszó bekérése tipikusan több alkalmazás által használt tevékenység. Ha dll-t készítünk belőle, akkor az egyszer megírt kódunkat több alkalmazásban is felhasználhatjuk.

A feladat tehát a következő: a jelszóbekérő alkalmazásunkat alakítsuk át osztálykönyvtárrá! Így a JelszoForm osztály funkcióit több alkalmazásból is felhasználhatjuk.

Ha alkalmazásunkból módosítás nélkül szeretnénk dll-t készíteni, azt nagyon egyszerűen megtehetjük. A Solution Explorerben a project kijelölése után nyissuk meg a Project menü Properties ablakát! A Common Properties / General / Output Type tulajdonságát változtassuk Windows Application-ről Class Library-re! Ha az Alkalmaz gombot megnyomjuk, az Output File exe-ről dll-re változik. A legtöbb esetben azonban, mint most is, módosítani szeretnénk a kódot. Ekkor inkább a forrásfájlok újrahasznosítása a kívánatos.



8. Gyakorlat. 1. ábra A Vendéglő a Világ Végén alkalmazás a jelszóbekérő ablakkal

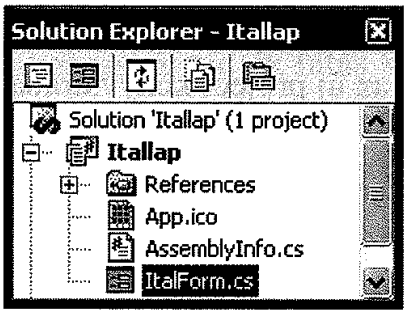
8.1. Jelszóbekérő komponens készítése

Készíthetünk úgy is osztálykönyvtárat, hogy először egy tesztalkalmazás segítségével teszteljük. Azonban a legtöbb esetben a dll készítés egy élő feladat kapcsán merül fel. Mindig is megvolt az igény a programfejlesztőkben, hogy megírt kódjaikat más alkalmazásokban is újrahasznosítsák. Ezért igyekeztek a feladatot minél általánosabban megvalósítani. Természetesen csak annyira, hogy az ne menjen a fejlesztési idő jó kihasználásának kárára. Az osztálykönyvtár csak egy további lehetőséget kínál a kód-újrahasznosításra.

Elhatároztuk, hogy a Vendéglő a Világ Végén alkalmazásunkat jelszóbekéréssel indítjuk. Az első felmerülő kérdés, hogy a JelszoForm és az ItalForm közül melyik lesz az alkalmazás főablaka. Bár sorrendben először a JelszoForm fog megjelenni, és látszólag ez az ablak nyitja meg az ItalFormot, mégis az alkalmazás lényegi tevékenységét nem a JelszoForm végzi. Vagyis a fő tevékenység az Itallap alkalmazásban van leírva, csak indulása előtt még bekéri a jelszót.

A jelszó bekérése egy önálló tevékenység. Jól elkülöníthető a rendelésektől és a számlaadástól. Ez a tevékenység el is hagyható az alkalmazásból, esetleg más alkalmazásokban is megjelenhet. Ez indokolja az önálló dll-ként történő megvalósítást. A dll tesztelését most közvetlenül az Itallap projektből végezhetjük.

Olvassuk be az Itallap projektet a Visual Studioba!



8. Gyakorlat. 2. ábra A Solution Explorer egy projekttel

Szeretnénk a jelszóbekérő alkalmazás már megírt kódját felhasználni ebben az alkalmazásban is. Készítsünk belőle egy dll-t mely a jelszó bekéréséért felelős. Ez már csak azért is előnyös, mert ha hasonló jogosultságokkal más alkalmazásnál is használni szeretnénk a jelszó bekérését, erre dll esetén lehetőségünk lesz. Több alkalmazás is elérheti ugyanazt a dll-t.

8.1.1. Második project létrehozása a solutionbe

File

New

Project

Project Types: Visual C# Projects

Templates: Class Lybrary

Name: Jelszo

Add to Solution kiválasztva

Folder Name: Ha az őt használó projekt alatt kívánjuk tárolni, a Browse gomb segítségével megkereshetjük a kívánt könyvtárat, ha önállóan, akkor a projektek könyvtárgyűjteménye megfelelő hely. Ha itt már van Jelszo, akkor vagy más nevet adunk, vagy más könyvtárba tesszük a projektet.

OK.

Megkapjuk a Jelszo névteret az üres Class1 osztállyal. Ha most megnézzük a Solution Explorert, most már két project van az alkalmazásunkban.

8.1.2. A már megírt osztály felhasználása

A Jelszo project egyenlőre egy Class1 nevű osztállyal rendelkezik. Mi szeretnénk a már megírt jelszóalkalmazásunk kódját felhasználni az új project elkészítésében.

Lehetőségünk van a már megírt forrásfájlt eredeti helyén csatolni a projecthez, de már előre tudjuk, hogy meg szeretnénk változtatni a kódját, így jobb lehetőség, ha a meglevő kódot átmásoljuk az új project könyvtárába, és így az eredeti változatlanul hagyása mellett lehet módosítani a fájlokat.

Az intézővel másoljuk át a jelszóalkalmazásunk könyvtárából a JelszoForm.cs fájlt az Elado könyvtár alatti Jelszo alkönyvtárba!

Solution Explorer

Jelszo project, jobbegér

Add

Add Existing Item

JelszoForm.cs

Open

A Class1.cs-t töröljük ki a Jelszo projectből! Kijelölés, Del gomb választás.

Próbáljuk meg a project önálló fordítását!

Solution Explorer

Jelszo project, jobbegér

Build

Több hibaüzenetet kapunk a Task List ablakban, melyek a Drawing és a Windows névtereket hiányolják, de a zárójeles mondatban megmondják a teendőt is. Hiányzik ezek hivatkozása a References közül.

```
C:\CS\Jelszo\JelszoForm.cs(2): The type or namespace name
'Drawing' does not exist in the class or namespace
'System' (are you missing an assembly reference?)
```

Valóban, ha a Solution Explorerben megnézzük az Itallap project hivatkozásait, ott szerepelnek a hivatkozások, míg a Jelszo projectben nem. Javítsuk a hiányt!

Solution Explorer

Jelszo project, jobbegér

Add Reference

.NET fül

System.Drawing.dll **Select**

System.Windows.Forms.dll **Select**

OK.

Próbáljuk újra a project felépítését! Most már sikerült. Nézzük meg a Jelszo alkönyvtár bin alkönyvtárában elkészült a Jelszo.dll! Mivel dll önállóan nem futtatható, csak valamilyen alkalmazásból tudjuk futás közben tesztelni. Ez lesz az Itallap.exe.

Most próbáljuk meg tehát a Solution-t futtatni, azt látjuk, hogy az Itallap alkalmazásunk az eddigieknek megfelelően fut, de ha megnézzük az Output ablakot, akkor látjuk, hogy mindkét projektet felépítettük. A jelszó ablakot még nem hívtuk meg!

Mikor szeretnénk meghívni? Mielőtt az ItalForm megjelenne.

```
static void Main()
{
 JelszoForm jFrm = new JelszoForm();
 jFrm.Show();
  Application.Run(new ItalForm());
}
```

Létrehoztuk az objektumot, majd a Show metódusával kitettük a képernyőre.

A fordító nem ismeri fel a JelszoForm-ot. Szükségünk van a using direktívára, hisz a JelszoForm osztály a Jelszo névtérben található. Írjuk tehát az ItalForm.cs fájl elejére a többi using mögé:

```
using Jelszo;
```

Ezen kívül még szükségünk lesz rá, hogy a referenciák közé is felvegyük a Jelszo.dll-t!

Solution Explorer

Itallap project, jobbegér

Add Reference

Projects fül

Jelszo.dll **Select**

OK.

Fordítás/Futtatás

Mindkét ablak megjelenik és önállóan működik. A Jelszo bezárása után az Itallap még fut, az Itallap bezárja a Jelszot is. Hibás felhasználónév esetén tovább vár, de három hibás jelszó megadása után (2 másodperccel) vége az alkalmazásunknak.

8.1.3. Jelszóbekérés párbeszédablakból

A gond csak az, hogy azonnal megjelenik mindkét ablak, tehát az Itallap alkalmazás a jelszó megadása nélkül is használható.

Nyissuk meg modálisan a jelszóbekérő ablakunkat, akkor nem lehet a másik ablakkal kommunikálni. Ha párbeszédablakként szeretnénk megnyitni, akkor azt a Show helyett a ShowDialog metódussal tehetjük.

Fordítás/Futtatás

Ekkor valóban előbb a jelszóbekérő ablak jelenik meg, s csak annak bezárása után indul az alkalmazás. Az ItalFormot a jelszó helyes megadásától függően szeretnénk indítani. A Formnak van egy DialogResult tulajdonsága, mely kilépéskor lekérdezhető. Tehát két tennivalónk van. Ezt a visszatérési értéket produkálni a 'Jelszo' projectben, és lekérdezni az 'Itallap' projektben.

ItalForm.cs:

```
static void Main()
{
 JelszoForm jFrm = new JelszoForm();
 if (jFrm.ShowDialog()==DialogResult.OK)
    Application.Run(new ItalForm());
}
```

JelszoForm.cs:

```
public JelszoForm()
{
 InitializeComponent();

 DialogResult=DialogResult.Cancel;
}

private void ellenorButton_Click(object sender,
                                 System.EventArgs e)
{
 // Megkeresi a user indexét.
 int i = 0;
 while ((i<numOfUsers) && (userBox.Text != user[i]))
       i++;

 if (i<numOfUsers) // Ha megtalálta a usert.
    if (jelszoBox.Text == jelszo[i]) // Helyes jelszó.
//       label.Text = "Helyes jelszó!";
       this.DialogResult = DialogResult.OK;
```

```
        else      // Hibás jelszó.
        {
            label.Text = "Hibás jelszó!";
            if (++missed == 3)
            {
                  label.Refresh();
                  // Frissítésre kényszeríti a vezérlőt.
                  System.Threading.Thread.Sleep(2000); // Vár.
//                          Application.Exit();
                  this.DialogResult = DialogResult.Cancel;
                  this.Close();
            }
...
}
```

Fordítás/Futtatás/Tesztelés

Vizsgáljuk meg, mi történik, ha egyszerűen kilépünk a jelszóbekérő ablakból, ha hibás a felhasználónév, ha hibás a jelszó egyszer, kétszer, háromszor; és mi történik, ha jó a felhasználónév és a jelszó is.

Láthatóan a Jelszo projectben nincs szükség a Main metódusra, kissé zavaró is, hisz nem hívódik meg. Amíg alkalmazás volt a Jelszo, addig ez kellett az indításhoz. Egy dll-nél nincs rá szükség. Tehát tegyük megjegyzésjelek mögé, vagy egyszerűen töröljük ki a forráskódból.

```
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
//      [STAThread]
//      static void Main()
//      {
//            Application.Run(new JelszoForm());
//      }
```

8.1.4. Módosítsuk a jelszóbekérő ablak felületét!

Ha a JelszoFormot design módban is látni akarjuk, akkor a JelszoForm.resx fájlra is szükség lesz. A Solution View / JelszoForm.cs / jobbegér / View Designer hatására kinyílik a Design View.

Csak két felhasználónk legyen: Vásárló és Eladó. Ezek közül egy combo boxból szeretnénk választani. A vásárlónak nem kell jelszó (ne is kérjünk), eladó esetén igen.

Töröljük a userBox nevű TextBoxunkat, és tegyünk a helyére egy szintén userBox nevű ComboBox vezérlőt!

Properties

Items hárompontos gomb

Vásárló

Eladó

Text Vásárló

TabIndex 0

Kezdetben, amikor a Vásárló van beállítva, ne kérjünk jelszót! Tiltsuk le a jelszó mezőt!

jelszoBox

Properties

Enabled False

Ha másik felhasználónevet választunk, akkor jelszót kell kérnünk.

userBox

Properties

Events

SelectedIndexChanged duplán kattintunk

```
private void userBox_SelectedIndexChanged(object sender,
                                          System.EventArgs e)
{
 jelszoBox.Clear();
 if(userBox.Text=="Eladó")
 {
    jelszoBox.Enabled = true;
    jelszoBox.Focus();
 }
 else
    jelszoBox.Enabled = false;
}
```

Módosítsuk a kódot:

```
numOfUsers=2;//5;
user = new   string[numOfUsers]{"Vásárló","Eladó"};
             //"Anna", "Pali", "Józsi","Mari","Kati"};
jelszo = new       string[numOfUsers]{"","ea"};
             //"Anna", "Pali", "Józsi","Mari","Kati"};
```

Fordítás/Futtatás/Tesztelés

8.1.5. Szükségünk van a JelszoForm adataira az ItalForm tagfüggvényeiben!

Mivel a JelszoForm objektum a Main lokális változója, a többi metódusból nem érhetjük el. Vagy az egész jFrm objektumot az ItalForm adattagjává kell tennünk, de kisebb helyet foglal, így hatékonyabb megoldás, ha csak egy user nevű tagváltozót veszünk fel az ItalFormban is.

Class View

ItalForm jobbegér

Add

Add Field

Field access: private

Field type: string

Field name: user

Field modifiers: Static

Finish

Értékét a Main metódusban kell beállítanunk, ahol még látjuk a jFrm lokális változót! Mivel a Mainben állítjuk be, és a Main statikus, tehát az adattagnak is statikusnak kell lennie.

```
private static string user;

static void Main()
{
 JelszoForm jFrm = new JelszoForm();
 if (jFrm.ShowDialog()==DialogResult.OK)
 {
    user = jFrm.userBox.Text;
    Application.Run(new ItalForm());
 }
}
```

Ez lenne a megoldás, de a userBox a JelszoForm privát tagja, így nem érhető el más osztályból. Az elnevezési konvenciók ajánlása szerint ilyenkor a tagváltozó nyilvánossá tétele helyett egy public tulajdonságot szokás készíteni. Az objektumorientált programozás egységbezárás elve is ezt ajánlja.

Nekünk itt is fölösleges az egész usrBox vezérlőhöz tulajdonságot rendelni, hisz mindig a kiválasztott felhasználóra vagyunk kíváncsiak. Ne engedjük a jelszoForm osztályon kívülről írni a felhasználót! Ezzel biztosítjuk a jelszóbekérést. Tehát csak get tulajdonság legyen!

Class View

JelszoForm jobbegér

Add

Add Property

Property access: public

Property type: string

Property name: CurrentUser

get

Finish

```
public string CurrentUser
{
 get
 {
    return userBox.Text;
 }
}
```

Ezután az ItalForm Main kódja a következőre módosul:

```
static void Main()
{
 JelszoForm jFrm = new JelszoForm();
 if (jFrm.ShowDialog()==DialogResult.OK)
 {
    user = jFrm.CurrentUser;
    Application.Run(new ItalForm());
 }
}
```

Az ItalForm betöltésekor eladó esetén engedélyezzük az árak módosítását, egyébként ne! Az ItalForm Load metódusát (a formon duplán kattintva) módosítsuk a user adattag mezőjéből kiolvasva a felhasználót! Alapértelmezésben az italok árát a tulajdonságlapon állítsuk Enabled False-ra!

```
private void ItalForm_Load(object sender,
                                    System.EventArgs e)
{
 if ( user == "Eladó")
 {
    aldebTextBox.Enabled = true;
    csinTextBox.Enabled = true;
    dzsynTextBox.Enabled = true;
    janxTextBox.Enabled = true;
    panTextBox.Enabled = true;
    tapTextBox.Enabled = true;
 }
}
```

Fordítás/Futtatás/Tesztelés

Azt még érzékeljük, hogy van egy kis 'szépséghibája' a dll-nek, csak a bevésett Vásárló és Eladó felhasználók bevésett jelszavát tudja használni. A későbbiekben ezt még javítjuk.

8.2. Feladatok

1. Készítsük el az ItalForm alkalmazást úgy, hogy a program futása során bármikor módosíthassuk a felhasználót! Készítsünk az ItalForm felületére egy 'Felhasználó' feliratú gombot, melynek választása ezt a módosítást lehetővé teszi!

2. Készítsünk egy párkereső programot! Az ablakban két comboBox nők és férfiak listáját tartalmazza. Mindkét listában legyen aktuális elem! Egy + feliratú gomb segítségével párokat hozhatunk létre az aktuális elemekből. Egy listadoboz tartalmazza a már létrejött párokat. Megvalósításfüggő, hogy a párba bekerülő személyt töröljük–e a combo listájából. A párok listájából lehessen törölni!

   Legyen lehetőség új jelentkezők felvételére, egy 'Új jelentkező' gomb hatására nyíljon meg egy párbeszédablak, ahol felvehetjük a jelentkező adatait! Az új jelentkező kerüljön a megfelelő combo listájára! Legyen az új jelentkező ablak egy önálló dll-ben megvalósítva!

3. Egy ablakban mutassuk meg egy raktár árukészletét! Egy listadoboz mutassa a készletelem leírását (név, ár, db)! A listából lehessen törölni, és lehessen árut beszállítani egy új ablak segítségével, melyben megadjuk az áru nevét (esetleg comboBoxból választva), árát és darabszámát. A felvesz gomb beteszi az árut a szállított listába. Az ablak bezárása után a beszállított áru növelje a raktárkészletet! Legyen a felvitel ablak egy önálló dll!

4. Az Elado alkalmazásban bárki átírhatja az áruk árát. Ehhez csak az eladóknak legyen joguk!
5. **Árengedmény a törzsvásárlóknak**. Az Elado alkalmazásunkban adjunk árengedményt a törzsvásárlóknak és az eladóknak! Ez a vásárlót csábító fogás csak akkor éri el hatását, ha a vásárló látja is az árengedményt.*

A feladatok csoportmunkával is megoldhatók! Pl. a raktárkészlet feladatot két személy készíti. Először a két ablak felületét készítik el az elnevezésekkel, és az új áruknál egy konstans listával. Ezután a dll beszerkeszthető a főablakhoz, és a feltöltés művelet megvalósítható. A beszállító dll önállóan fejleszthető tovább, elkészültével csak ki kell cserélnünk a régi dllt.

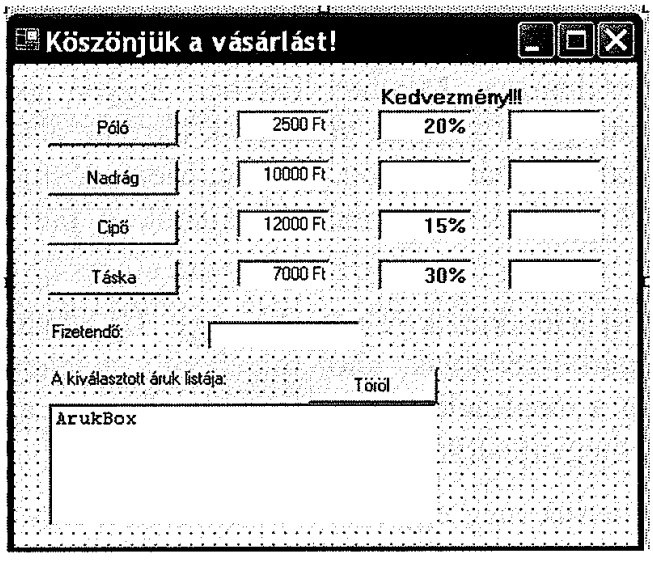
8.3. Megoldásötletek

5. Árengedmény a törzsvásárlóknak

Ez három felhasználót jelent, Vásárló, Eladó és Törzsvásárló.

Tegyünk a kedvezményes áruk ára mellé két szövegmezőt, melyek az engedmény mértékét és a kedvezményes árat tartalmazzák. Adjunk 20% árengedményt a pólókra, 15%-ot a cipőkre és 30%-ot a táskákra.

A vezérlőket a legegyszerűbben az erőforrás-szerkesztővel hozhatjuk létre, majd láthatatlanra állítva őket, csak akkor jelenjenek meg, ha a megfelelő felhasználó használja a programot.



8. Gyakorlat. 3. ábra Az árengedmény vezérlői

TextBox, Name: KedvLabel, TextAlign: BottomLeft,

Name	poloBoxKSZ	nadragBoxKSZ	cipoBoxKSZ	taskaBoxKSZ
Text	20%		15%	30%

Name	poloBoxK	nadragBoxK	cipoBoxK	taskaBoxK
Text				

Kijelölve az összes kedvezménymezőt, ForeColor: 192;0;0, Font: Félkövér, 10. méret tulajdonságokat állítsuk be! És minden vezérlő legyen láthatatlan! Majd a feltöltés során derül ki, melyiket tegyük láthatóvá.

```
private void EladoForm_Load(object sender,
                                System.EventArgs e)

 if (user == "Eladó")
 {
   poloBox.Enabled = true;
   nadragBox.Enabled = true;
   taskaBox.Enabled = true;
   cipoBox.Enabled = true;
 }
 if (user == "Eladó" || user== "Törzsvásárló")
 {
   this.Size = new Size(448,364); //Elférjen a leárazás.
   int ar, kedv;
   char [] penz = {' ','F','t'};
   label2.Visible = true;      //Kedvezmény szöveg!

   if (poloBoxKSZ.Text !="")
   {
       poloBoxKSZ.Visible = true;
       poloBoxK.Visible = true;
       ar=Convert.ToInt32(poloBox.Text.TrimEnd(ft));
     kedv=Convert.ToInt32(poloBoxKSZ.Text.TrimEnd('%'));
       ar= ar*(100-kedv)/100;
       poloBoxK.Text=ar.ToString()+" Ft";
   }
 }
}
```

Természetesen a fizetendőt is ebből az árból kell számolnunk!

Megvalósíthatjuk származtatott Form segítségével is. Eladó felhasználó esetén az eredeti, míg kedvezményes vásárlók esetén az utód form töltődik be.

9. Gyakorlat. Vezérlő készítése

9.1. Digitális időkijelző vezérlő készítése

9.1.1. A felhasználói interfész elkészítése

Készítsünk egy digitális időt kijelző vezérlőt, mely az indítástól eltelt idő mérésére és kijelzésére alkalmas. Ezt a vezérlőt több alkalmazásban szeretnénk felhasználni.

Fájl

New

Project

Project Types: Visual C# Projects

Templates: Windows Control Library

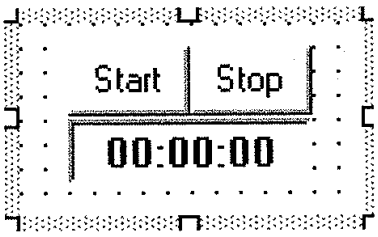
Name: Ora

Close Solution

OK

Nevezzük át a készen kapott UserControl1 nevű osztályunkat, a forrásfájlt, a konstruktort és az osztályhoz tartozó megjegyzéssort OraControlra!

A vezérlő tartalmazzon egy időkijelző szövegmezőt, egy indító és egy megállító gombot!



9. Gyakorlat. 1. ábra A digitális stopper a szerkesztőben

	Name	Text	Read Only	BackColor	Font	Text Align
Text-Box	timeBox	00:00:00	True	Highlight Text	Félkövér, 10. méret	Center
Button	startButton	Start				
Button	stopButton	Stop				

A TextBox TabIndex tulajdonságát állítsuk 1-re, hogy megjelenéskor ne legyen kiemelve a szövege! A felhasználó ne módosíthassa értékét (Read Only), de ne legyen szürke a háttere (BackColor)!

Húzzunk be a ToolBoxból, a Windows Forms listáról (ne a Components listáról) egy **Timer** komponenst! A timer nem kerül a vezérlő felületére, hisz nem egy látható elem, a tervezőnézet alján egy erre kialakított sávban kerül bemutatásra. Neve (Name) legyen timer!

Mivel az időzítő a **Tick** üzenet küldésének intervallumát ezredmásodpercekben méri, ahhoz hogy másodpercenként kapjunk üzenetet, Interval tulajdonságát állítsuk 1000-re!

9.1.2. A funkciók megvalósítása

A vezérlő hátterében célszerű az időt egy **DateTime** típusú (time), változóban tárolni, mert ebből kényelmesen olvashatjuk ki az idő értékét, nem kell a perc, óra átváltásokkal foglalkozni. A time értékét másodpercenként növeljük! A time értékéből írhatjuk ki a képernyőre formázva a megjeleníteni kívánt időt. Az időzítő feladata e szám automatikus növelése lesz.

9.1.2.1. Hozzuk létre a time tagváltozót!

Class View

OraControl

Add

Add Field

Field access private

Field type DateTime

Field name time

Finish

A time kezdőértéke 0, melyet a konstruktorban állítunk be.

```
public OraControl()
{
 // This call is required by the Windows.Forms Form Des.
 InitializeComponent();

 // TODO: Add any initialization after the InitForm call
 time = new DateTime(0);
}
```

9.1.2.2. A timer növelje másodpercenként a time értékét!

Ezt a Timer Tick eseményének kezelésével valósítjuk meg.

OraControl.cs [Design]

Timer

Properties fül

Events eszközgomb

Tick eseményen duplán kattintunk.

```
private void Timer_Tick(object sender,System.EventArgs e)
{
 time = time.AddSeconds(1);
 timeBox.Text = time.Hour.ToString().PadLeft(2,'0')+":"
              +time.Minute.ToString().PadLeft(2,'0')+":"
              +time.Second.ToString().PadLeft(2,'0');
}
```

Egy vezérlőt nem lehet magában futtatni, de a fordítási és szerkesztési hibákat megkaphatjuk.

Solution Explorer

Ora project jobbegér

Build

Indítsuk és állítsuk meg az időzítőt!

A két gomb Click eseményének kezelője legyen a következő:

```
private void StartButton_Click(object sender,
                               System.EventArgs e)
{
 timer.Start();
}

private void StopButton_Click(object sender,
                              System.EventArgs e)
{
 timer.Stop();
}
```

9.1.3. A teszteléséhez készítsünk egy alkalmazást, mely használja ezt a vezérlőt!

File

New

Project

Project Types: Visual C# Projects

Templates: Windows Application

Name: OraTeszt

Add to Solution

OK

A tesztalkalmazásunk legyen a kezdő alkalmazás! (Itt keresse a Maint.)

Solution Explorer

OraTeszt project jobbegér

Set as Startup Project

A ToolBox eszközsorban a UserControls végén ott az Ora nevű eszközgomb, amit drag&drop technikával a Formra húzhatunk. Ha megnézzük a Solution Explorert, azt látjuk, hogy Egy Ora hivatkozás került be az OraTeszt project References listájába.

Fordítás/Futtatás/Tesztelés

9.1.4. Írjunk az óra vezérlőhöz indító és megállító metódust!

Ezeket a metódusokat kívülről, a vezérlőt használó programból is el akarjuk érni, így public láthatóságúra állítjuk őket!

Class View

OraControl

Add

Add Method

Method access public

Return type void

Method name Start

Comment Indítja a stoppert

Finish

```
/// <summary>
/// Indítja a stoppert
/// </summary>
public void Start()
{
 timer.Start();
}
```

Ismételjük a Stop metódussal! A StartButton_Click és a StopButton_Click privát metódusokból ezeket a függvényeket hívjuk meg! Vagyis timer.Start() helyett Start() szerepeljen!

9.1.5. A vezérlőhöz tulajdonság készítése

Azt tapasztaljuk, hogy a stopper nem kezdi újra a számlálást újraindításkor. Ez a funkció feladatfüggő. Van amikor épp továbbszámolni akarunk, van amikor mindig újrakezdeni. Jó lenne, ha lenne egy olyan tulajdonsága a vezérlőnek, amivel beállíthatnánk ezt a funkciót!

Egy tulajdonság az osztályban tárolt adatokon dolgozik. Vagy már meglevő adattagokat ír, olvas, vagy újakat. Az újraindításhoz nekünk egy logikai változóra van szükségünk.

Class View

OraControl jobbegér

Add

Add Field

Field access private

Field type bool

Field name restart

Finish

Class View

OraControl jobbegér

Add

Add Property

Property access public

Property type bool

Property name Restart

Finish

```
private bool restart = true;

public bool Restart
{
  get
  {
    return restart;
  }
  set
  {
    restart = value;
  }
}
```

Fordítás/Futtatás/Tesztelés

A Start gomb leütésekor, ha igaz a Restart, akkor kezdje újra a számlálást, egyébként ne!

```
public void Start()
{
 if (Restart)
    time = new DateTime(0);
 timer.Start();
}
```

Fordítás/Futtatás/Tesztelés

Állítsuk a Properties ablakban False-ra a Restart tulajdonságot, úgy is teszteljük az alkalmazást!

A tulajdonság természetesen futásidőben is módosítható.

9.1.6. Legyen az óra kezdőértékének beállítását lehetővé tevő tulajdonság!

Mivel később a felhasználás során lehetővé akarjuk tenni a mutatott idő beállítását, ezért hozzunk létre a time adattaghoz egy public elérésű írható, olvasható tulajdonságot!

Class View

OraControl

Add

Add Propety

Property access public

Property type DateTime

Property name Time

Finish

```
public DateTime Time
{
 get
 {
    return time; // return new DateTime(); helyett
 }
 set
 {
    time = value;
    timeBox.Text = time.Hour.ToString().PadLeft(2,'0')
       +":" +time.Minute.ToString().PadLeft(2,'0')+":"
       +time.Second.ToString().PadLeft(2,'0');
 }
}
```

A tulajdonság természetesen nem jelenik meg az első fordításig a Form1 vezérlőjének tulajdonságlapján!

Fordítás/Futtatás/Tesztelés

Ha azt akarjuk, hogy az újraindításkor a tulajdonságban beállított kezdőértékről induljon a stopper, akkor ezt az értéket tárolnunk kell egy tagváltozóban, melyet kívülről, a stoppert használónak is célszerű elérnie. Tehát hozzunk létre egy startTime tagváltozót, és kapcsoljunk hozzá egy StartTime tulajdonságot is!

Class View

OraControl jobbegér

Add

Add Field

Field access private

Field type DateTime

Field name startTime

Finish

Class View

OraControl

Add

Add Propety

Property access public

Property type DateTime

Property name StartTime

Finish

```
private DateTime startTime;

public OraControl()
{
//This call is required by the Windows.Forms Form Designer.
 InitializeComponent();

 // TODO: Add any initialization after the InitForm call
 time = new DateTime(0);
 startTime = new DateTime(0);
}
```

```
public DateTime StartTime
{
  get
  {
     return startTime;
  }
  set
  {
     startTime = value;
  }
}

public void Start()
{
  if (Restart)
     Time = startTime;//new DateTime(0);
  timer.Start();
}
```

Mivel itt a Time tulajdonságnak adtunk értéket, ez a set metódusának hívását jelenti, a set pedig beállítja a timeBox szövegét is. Ha a time adattag kapta volna az értéket, akkor csak egy másodperc múlva jelenik meg a képernyőn az újraindításkor kapott érték.

Fordítás/Futtatás/Tesztelés

A teszteléshez természetesen be kell állítanunk a tulajdonságok értékét. Ezt a Form1(Design) nézetében az oraControl1 tulajdonságablakában tehetjük. Ha csak az idő értékét gépeljük be (3:11:5), akkor az aktuális dátumot automatikusan elé teszi.

Jól látható a futáskor, a vezérlő kezdetben a Time tulajdonságban beállított időt mutatja, de indításkor Restart true esetén a StartTime tulajdonságban beállított értékről indul, egyébként a Time tulajdonságban beállított értékről. Természetesen futás közben a Time értékének módosítása azonnal jelentkezik.

Ha a vezérlőnk kódjának módosításakor olyan hibát vétünk, amit a felhasználó alkalmazás fordítója nem tud értelmezni, a fordító kitörölheti a `'#region Windows Form Designer generated code'` blokkból a rá vonatkozó sorokat, és így eltűnik vezérlőnk a szerkesztőfelületről. E nem túl szimpatikus megoldás indoka a fejlesztő hibakeresésének egyszerűsítése lehet. Ha elolvassuk a blokk kezdő megjegyzését láthatjuk, hogy ezt a kódrészt a Visual Studio generálja és módosítja.

```
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
```

A vezérlő kódjának javítása után az őt használó alkalmazásban a Visszavonás eszközgomb segítségével állíthatjuk vissza az eredeti, most már hibátlan állapotot.

Hasonló jelenség léphet fel, ha a Sign alkalmazásunk InkPanel vezérlőjét olyan gépen kívánjuk fordítani, melyre nincs telepítve a Tablet PC Platform SDK.

9.2. Feladatok

1. **StartVisible, StopVisible tulajdonság a Behavior kategóriában.** Készítsünk az óra vezérlőnkhöz olyan tulajdonságot, mely megadja, látszik e a vezérlőn a Start és a Stop gomb! A tulajdonság az Behavior kategóriában jelenjen meg!*
2. Készítsünk ébresztőóra formájú ablakban egy digitális ébresztőórát! Az óra ketyegjen is!
3. A Stop gomb kétszeri megnyomására a stopper nullázza ki a kezdőértéket, vagy állítsa be a StartTime tulajdonságban megadott értékre!
4. **Visszaszámlálás**. Készítsünk az óra vezérlőnkhöz egy tulajdonságot, mely megadja, hogy az óra a beállított kezdőértéktől visszafelé vagy előre számole! Visszaszámolásnál a null érték elérésekor megáll és hangjelzést ad.*
5. **Az óra vezérlőnknek legyen óraformájú ikonja!***
6. Készítsünk a sakkjátszmáknál szokásos, a hátralevő játékidő kijelzésére alkalmas programot! Az óravezérlők működtessenek egy-egy folyamatkijelzőt is, mely mutatja a játékidő végének közeledtét! A játékot lehessen újraindítani! Ha elfogy a játékos ideje, akkor a partner nyert.
7. Készítsünk olyan vezérlőt, melyre egérrel vagy aktív tollal írni lehet!
8. **Az amőba alkalmazásunkat bővítsük ki hanggal!***
9. Készítsünk alkalmazást, mely bemutatja tevékenységünket! A bemutató tartalmazzon az alkalmazásba beépített ablakban levetített videofilmeket, hanganyagokat!
10. **Honlapok az alkalmazás ablakában.** Nyissunk alkalmazásunkban egy ablakot, mely vezérlőink állapotától függően különböző honlapokat mutat be!*

11. Készítsük el a szavazatszámláló gombot vezérlőként! Lehessen beállítani a maxérték és a sötétedik / világosodik tulajdonságát!

12. Készítsünk a pénznemet kiíró, csak számot, +,- jelet elfogadó vezérlőt, mely a negatív számokat pirossal írja ki, és ezres bontásban szóközt tesz a számjegyek közé!

9.3. Megoldásötletek

1. StartVisible, StopVisible tulajdonság a Behavior kategóriában

```
[Category("Behavior")]public bool StartVisible
{
 get
 { return startButton.Visible;}
 set
 { startButton.Visible = value;}
}
```

Ha azt akarjuk, hogy a láthatatlan gombok helye ne legyen fönntartva a felhasználás során, az OraControl vezérlőnk TextBox vezérlőjének Anchor tulajdonságát állítsuk Bottom, Left-re! Ekkor a két gomb láthatóságát kikapcsolva a vezérlő mérete a kijelző méretére csökkenthető. Természetesen ilyen esetben a programból kell a Start és Stop metódusok segítségével vezérelni az órát.

4. Visszaszámlálás

```
/// <summary>
/// no==true esetén az idő értéke másodpercenként nő.
/// </summary>
public bool no=true;

public bool No
{
 get
 { return no;      }
 set
 { no = value;     }
}

private void Timer_Tick(object sender,System.EventArgs e)
{
 //time = time.AddTicks(TimeSpan.TicksPerSecond);
 if (no)
    time = time.AddSeconds(1);
 else
```

```
    time = time.AddSeconds(-1);
  timeBox.Text = time.Hour.ToString().PadLeft(2,'0')+":"
               +time.Minute.ToString().PadLeft(2,'0')+":"
               +time.Second.ToString().PadLeft(2,'0');
  if (time.Hour==0 && time.Minute==0 && time.Second==0)
  {
    timer.Stop();
  }
}
```

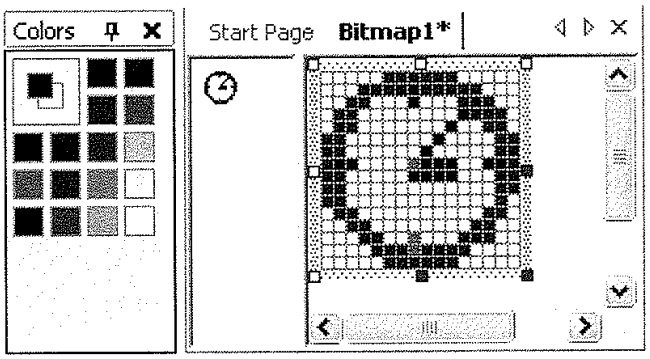
5. Az óra vezérlőnknek legyen óraformájú ikonja!

Először hozzunk létre egy 16x16 képpontos – ez a szokásos ikonméret – bitmapet!

File / New /File / Bitmap file / Open.

A generált kép 49x49-es, a jobb alsó sarkát megfogva nyomjuk össze 16x16 képpontosra! A Visual Studio státuszsora mutatja a jobb alsó képkocka sor és oszlopindexét. A bitmap úgy lesz 16x16-os, ha az összenyomáskor 17x17-et mutat a külső keret, vagy ha a bitmap jobb alsó kockájára 15x15-öt ír ki a státuszsor. Mentésnél a könyvtárban a fájl neve alatt 16x16 Bitkép legyen kiírva!

Ezután a rajzeszközökkel szerkeszthetjük.



9. Gyakorlat. 2. ábra Az óra vezérlő ikonja a szerkesztőben

Természetesen a kellékek Paint alkalmazásával is készíthetünk 16x16 képpontos bitmapet.

Solution Explorer / Add / Add Existing Item / Files of type: Image files / Ora.bmp / Open.

Solution Explorer / Ora.bmp aktív / Properties fül / Build Action tulajdonság / Embedded Resource

```
[ToolboxBitmapAttribute(typeof(OraControl),"Ora.bmp")]
public class OraControl :System.Windows.Forms.UserControl
```

OraTeszt / Form1 [Design] / Toolbox / My User Controls jobbegér / Add/Remove Items / Browse / Ora.dll / OK

A forráskódból törölt fájlok a resource állományból nem törlődnek, ezért pl. egy korábban felvett erőforrás törléséhez szükség lehet a resx állomány törlésére. A következő felépítéskor újra létrejön az állomány.

A resource állományba beépülést ellenőrizhetjük, ha átmásoljuk valahová Ora.bmp állományunkat, majd töröljük a projektből. A fordítás mindaddig sikerül, míg nem töröltük resource állományunkat. (Ilyenkor közös solution esetén érvénytelenné válhat az automatikusan felvett ikon, de az Ora.dll újrafelépítése után ez is megszűnik.) Ha a resource állományt töröljük, annak felépítéséhez szükség lesz újra az Ora.bmp állományra.

8. Az amőba alkalmazásunkat bővítsük ki hanggal!

Tegyünk a Toolbox-ra egy Windows Media Player vezérlőt! Húzzuk be az AmobaForm felületére, és kapja az axMediaPlayer nevet! Visible tulajdonsága legyen False! A kívánt hangfájlokat tároljuk privát string típusú mezőkben!

```
public AmobaForm()
{
 masodikImage = Image.FromFile( "BOOK01A.ico");
 elsoHang = "C:\\WINDOWS\\Media\\ding.wav";
 masodikHang = "C:\\WINDOWS\\Media\\Ir_begin.wav";
 gyozHang = "C:\\WINDOWS\\Media\\tada.wav";
}

private void pictureBox_Click(object sender,
                              System.EventArgs e)
{
 if (elso)
 {
    ((PictureBox)sender).Image = elsoImage;
    axMediaPlayer.FileName = elsoHang;
 }
 else
 {
    ((PictureBox)sender).Image = masodikImage;
    axMediaPlayer.FileName = masodikHang;
 }
 axMediaPlayer.Play();
 if (Winner(sender))
 {
    axMediaPlayer.FileName = gyozHang;
    axMediaPlayer.Play();
...
```

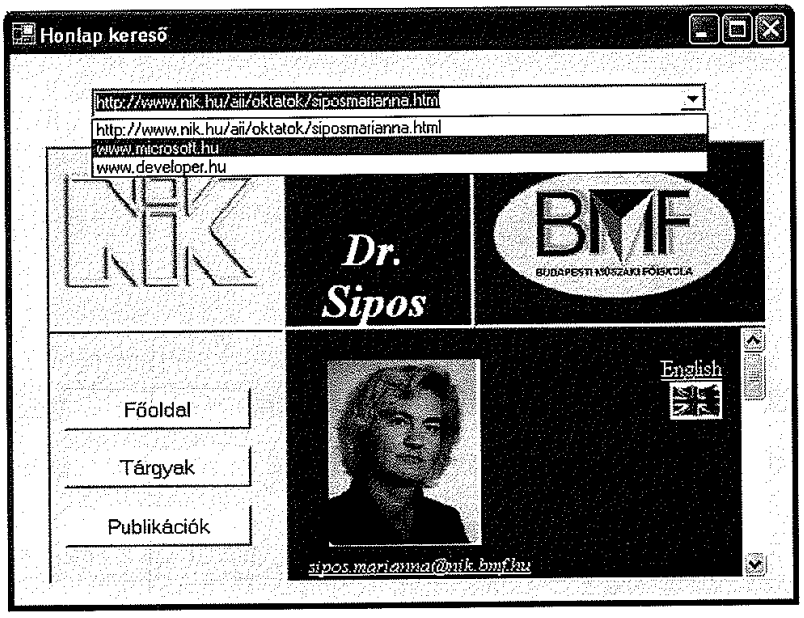
Ha teszteljük, a hangok megszólalnak, de a Controls tömbben indexelési hibát kapunk. A 0. indexű elemmel van baj. Amikor a médialejátszót felvettük a formra, ez az eszköz lett a 0. indexű, mert az utoljára felvett lesz mindig az. Ez okozza a hibát. Módosítsuk a médialejátszó indexét!

```
public AmobaForm()
{
 gyozHang = "C:\\WINDOWS\\Media\\tada.wav";
 Controls.SetChildIndex(axMediaPlayer,100);
}
```

10. Honlapok az alkalmazás ablakában

Válasszuk ki a Toolboxba a Microsoft webböngészőt a Costumize Toolbox ablak COM Components listájából!

```
private void addressComboBox_SelectedIndexChanged(object
                              sender, System.EventArgs e)
{
 Object o = null;
 axWebBrowser.Navigate(addressComboBox.Text, ref o,
                                    ref o, ref o, ref o);
}
```



9. Gyakorlat. 3. ábra Honlapböngésző az alkalmazásban

10. Gyakorlat. Adatbázis-elérés

A Jelszó alkalmazás azonosítóit és jelszavait tároljuk adatbázisban! Minden, a jelszó komponenst használó alkalmazás más és más adatbázisfájlon dolgozik, így lehetőség nyílik különböző jogosultságok meghatározására.

Ez az adatbázis természetesen nagyon egyszerű, de a cél nem az adatbázis-kezelés megismerése, hanem az adatokhoz történő hozzáférés bemutatása.

Ha van a gépünkre Microsoft SQL Server telepítve, azt is használhatjuk, de ilyen kis mennyiségű adat tárolására Access adatbázis is bőségesen megfelelő. Az adatokat egyszerűen fájlba is menthetnénk – amint azt az 5. gyakorlat 12. feladatánál tettük–, de mi most az adatbázis-elérést kívánjuk bemutatni.

A feladat megoldásához a gépre telepített SQL Serverre vagy Access adatbázis-kezelőre van szükség.

10.1. Az adatbázis létrehozása

10.1.1. SQL Server adatbázis létrehozása

Indítsuk el a Visual Studiot! Nem kell projektet létrehoznunk, hogy az adatbázison dolgozni tudjunk!

Server Explorer

Servers

Saját szerver megnyitása

SQL Servers

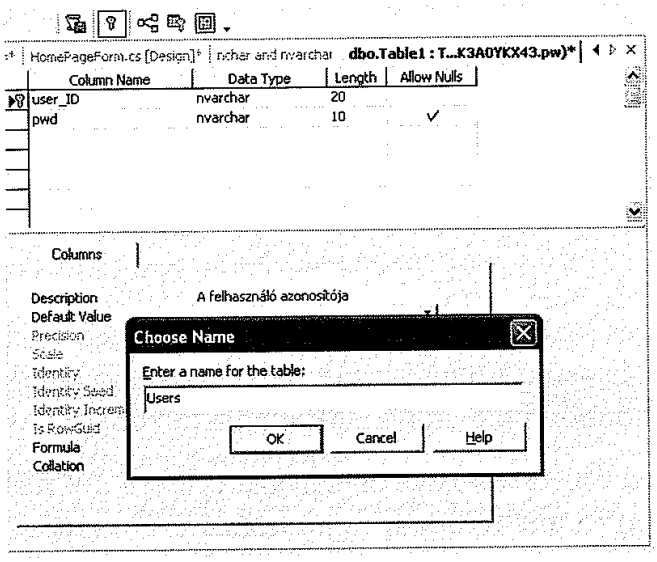
Saját SQL Server jobbegér

New Database

New Databse Name: pw

OK

Jelszó esetén nem érdemes ilyen nyilvánvaló adatbázisnevet adnunk, de most érthetőségre törekszünk. A pw adatbázist kinyitva a Server Explorerben válasszuk a Tables foldert! A gyorsmenü New Table parancsával táblákat hozhatunk létre.



10. Gyakorlat. 1. ábra A Users tábla mezői

A user_ID nvarchar típusa max 20 karakter hosszú, de változó hosszúságú adatbevitelt tesz lehetővé. Engedi a nemzeti karakterek használatát, így a karaktereket 2 byte helyen tárolja.

Az ábra tetején látható Table eszközsor kulcs ikonját a kijelölt mező(k)höz választva azok a tábla kulcsmezői, vagyis egyedi azonosítói lesznek.

A mentés ikon választására nevet adhatunk a táblának. A nevén duplán kattintva kitölthetjük, értékeit megnézhetjük, módosíthatjuk.

user_ID	pwd
Eladó	ea
Vásárló	<NULL>
Törzsvásárló	tv

10. Gyakorlat. 2. ábra A Users tábla mezőinek feltöltése

A ceruza az első oszlopban a tárolt adatbázishoz képest a módosítást jelzi. Az adatok mentése a sorról történő lelépéskor valósul meg. Ekkor eltűnik a ceruza.

10.1.2. Access adatbázis létrehozása

Az Access segítségével történik.

Access

Fájl

Új

Üres adatbázis

Hely: Megadjuk a tárolás helyét.

Fájlnév: pwd.mdb

Létrehozás

Tábla létrehozása tervező nézetben

Egy, az előzőhöz nagyon hasonló ablakot kapunk, melyben létrehozhatjuk a tábla mezőit. A user_ID 20 mezőméretű szöveg, pwd 10 hosszú szöveg. A kulcsmezőt itt is a kulcs ikonnal állíthatjuk be. A tábla neve mentéskor legyen Users!

Access

Nézet menü

Adatlap nézet

Kezdődhet a mezők feltöltése. Itt is ceruza jelzi a módosított és nem mentett sort, és a sorról lelépve történik a mentés az adatbázisba. Ne töltsük föl teljesen adatokkal!

Visual Studio

Server Explorer

Data Connections jobbegér

Add Connection

Szolgáltató fül

Microsoft Jet 4.0 OLE DB Provider

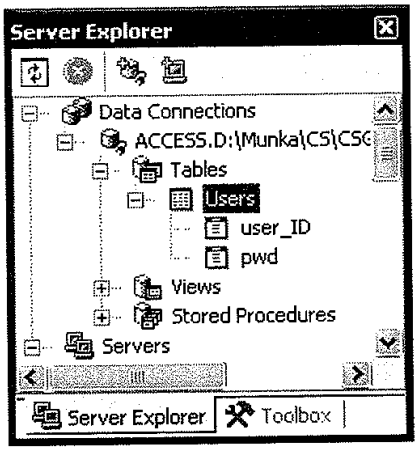
Tovább

Jelöljön ki, vagy adjon meg egy adatbázisnevet: pw.mdb

Kapcsolat tesztelése, A kapcsolat vizsgálata sikeres.

OK

Az adatbázist megnyitva, adatai az SQL szerverhez hasonlóan szerkeszthetők.



10. Gyakorlat. 3. ábra A pw.mdb Users táblája a Server Explorerben

Az adatokkal történő feltöltést folytassuk!

10.2. Az adatbázis adatainak megjelenítése az alkalmazásban

Vegyük elő Jelszó komponensünket! De újra is írhatjuk, mert szinte a teljes kód megváltozik.

10.2.1. Az adatbázis-adatok hozzáférését támogató objektumok

Húzzuk pw adatbázisunk Users tábláját a JelszóForm felületére [Design]! Mivel az adatbázis-kapcsolat és az adapter is láthatatlan komponensek, ezért az ablakhoz tartozó alsó osztott sávba kerülnek. Ha az SQL adatbázis tábláján dolgozunk, sqlConnection1 és sqlDataAdapter1 az objektumok neve. Ha az Access adatbázison, oleDbConnection1 és oleDbDataAdapter1. Nevezzük át jelszoConnection és jelszoAdapterre!

Mivel mi most csak olvasni akarunk adatokat, nézzük meg a jelszoAdapter tulajdonságablakában a SelectCommand CommandText értékét! Ez egy SQL lekérdezés, mely a Users tábla mindkét mezőjét visszaadja.

jelszoAdapter jobbegér

Preview Data

Fill Dataset

Hatására a megjelenő ablakban megnézhetjük az adapter által lekérdezett adatokat. Ha ezeket egy Dataset objektumban tárolni akarjuk:

jelszoAdapter jobbegér

Generate Dataset

New: JelszoSet

OK

A Users tábla ki volt jelölve. A generált JelszoSet osztály ezt a táblát tartalmazza.

Nézzük meg a Class View ablakban a Jelszo névteret! A JelszoForm osztály mellé elkészült a JelszoSet osztály.

Az osztály objektuma a JelszoForm.cs [Design] ablak alsó sávjában megjelenő jelszoSet1 objektum. A mentés után a JelszoForm osztályban meg is jelenik a privát láthatóságú jelszoSet1. Mivel a láthatósága privát, más assemblyk nem fognak hozzáférni. A mi projektünk C# nyelvű. A C# case-sensitive, így megtehetjük, hogy osztály és objektuma csak a kis és nagy kezdőbetűben tér el. Vagyis nevezhetjük objektumunkat a Properties Name tulajdonságát átírva jelszoSet-nek!

Kattintsunk duplán a Class View JelszoForm osztály jelszoSet adattagján! Az adattag deklarációjára ugrik a kurzor a forráskódban:

```
private Jelszo.JelszoSet jelszoSet;
```

Különböző DataSetek esetén a sqlDelete-, sqlInsert-, sqlSelect-, sqlUpdateCommand1 objektumoknak is beszédesebb nevet kellene adnunk.

10.2.2. A jelszoSet és a userBox feltöltése adatokkal

A kezdőértékadás a konstruktor feladata. Az SQLDataAdapter Fill metódusa a beállított lekérdezésnek megfelelően tölti föl a jelszoSet objektumot. A feltöltött jelszoSet-ből soronként kiolvashatóak az értékek.

Ahhoz, hogy csak az adatbázis értékei legyenek a userBoxban, a Properties ablak Items tulajdonságának gyűjteményét ki kell törölnünk.

```
public JelszoForm()
{
  InitializeComponent();

  jelszoAdapter.Fill(jelszoSet);
  foreach (JelszoSet.UsersRow row in jelszoSet.Users)
    userBox.Items.Add(row.user_ID);
}
```

A userBox a tulajdonságlap DataSource és DisplayMember tulajdonságának beállításával is feltölthető lett volna. Lásd elmélet.

Fordítás/Futtatás/Tesztelés

Mivel általános megoldást kívánunk készíteni, nem tölthetjük ki előre a userBox szövegét, mert lehet, hogy nincs is Vásárló a felhasználók között. Tehát a Text tulajdonságát is töröljük, és a jelszoBox legyen engedélyezve (Enabled: True)!

Így tehát a felhasználó kénytelen felhasználó nevet választani a listából. Ha a kiválasztott index jelszava üres, ne kérjünk jelszót! Egyébként igen!

```
private void userBox_SelectedIndexChanged(object sender,
                                          System.EventArgs e)
{
 jelszoBox.Clear();
 if (jelszoSet.Users[userBox.SelectedIndex].pwd == "")
    jelszoBox.Enabled = false;
 else
    jelszoBox.Enabled = true;
}
```

Fordítás/Futtatás/Tesztelés

10.2.3. A jelszó bekérése, ellenőrzése

Ha a Vásárlót választjuk a listából, a program kezeletlen, kivétel hibával leáll. Valóban a jelszónak nincs értéke, nem üres sztring az értéke, amire rákérdeztünk. Kapjuk el a kivételt, és állítsuk üres sztringre a pwd értékét! A problémát megoldhattuk volna úgy is, hogy az adatbázisban üres sztring alapértelmezett értéket adunk a jelszó mezőnek, de ezt nem várhatjuk el az összes felhasználótól!

```
private void userBox_SelectedIndexChanged(object sender,
                                          System.EventArgs e)
{
 jelszoBox.Clear();
 try
 {
    if (jelszoSet.Users[userBox.SelectedIndex].pwd == "")
       jelszoBox.Enabled = false;
    else
       jelszoBox.Enabled = true;
 }
 catch
 {
    jelszoSet.Users[userBox.SelectedIndex].pwd = "";
    jelszoBox.Enabled = false;
 }
}
```

Megjegyzés:

A pwd mező értékét a jelszoSet-ben üres sztringre állítottuk, de nem mentettük az adatbázisba. Az adatbázist megnézhetjük a Server Explorerben. Így a program futása alatt nem dob többet kivételt az érték vizsgálata (ellenorButton_Click), miközben nem nyúltunk az esetlegesen más program által is használt adatbázishoz.

Állítsuk a userBox **DropDownStyle** tulajdonságát **DropDownList**-re! A beállítással azt értük el, hogy a beviteli mezőbe csak a listán szereplő érték vihető be, de ahhoz elég annyi karaktert leütnünk, amennyi egyértelműen azonosítja. A mi esetünkben minden felhasználó neve más-más betűvel kezdődik, így elég egyetlen betűt leütnünk a választáshoz. A listaelem kiválasztásával a fenti függvény is meghívódik, hisz változott az index.

Most már koncentrálhatunk a jelszó ellenőrzésére.

```
private void ellenorButton_Click(object sender,
                                 System.EventArgs e)
{
 bool helyesJelszo = false;
 if (jelszoSet.Users[userBox.SelectedIndex].pwd ==
                                        jelszoBox.Text)
   helyesJelszo = true;
 else
   missed++;

 if (helyesJelszo)
   this.DialogResult = DialogResult.OK;
 else if (missed ==3)
   this.DialogResult = DialogResult.Cancel;
   else label.Text = "Hibás jelszó!";
}
```

10.3. Különböző adatbázisok elérése

A probléma már csak az, hogy az adatelérést támogató adatbázis a tábla behúzásakor beégetődött a programba. Ahhoz, hogy a Jelszo komponens különböző adatbázisokon használható legyen, az adatbázis megadását kívülről is lehetővé kell tennünk.

Mivel típusos DtaSettel dolgozunk, a különböző adatbázisok szerkezetének meg kell egyeznie. Ennek leírását mellékelni kell a komponenshez. A mi esetünkben csak Microsoft SQL Server adatbázis lehet, mert az objektumaink SqlConnection, SqlAdapter, SqlCommand típusúak. A tábla neve kötelezően Users, és a mezők

neve és típusa is meg kell egyezzen a pw adatbáziséval. Ugyanis a létrehozott, és a kódban szereplő JelszoSet osztály ilyen adatokon dolgozik.

Azonban az adatbázis neve és az SQL Server amin fut – más és más lehet. Így a különböző felhasználók létrehozhatják a saját adatbázisukat, vagy egy gépen a különböző programok különböző adatbázisokon dolgozhatnak.

A csatlakozás beállítására még a jelszoSet feltöltése előtt szükséges. Ez pedig a konstruktorban történik. Ha kívülről egy másik alkalmazásból akarjuk megadni a szerver és az adatbázis nevét, azt csak paraméterezett konstruktorral tehetjük.

Másoljuk le a jelenlegi paraméter nélküli konstruktorunkat a JelszoForm osztályba, és adjunk neki két string típusú paramétert! Az InitializeComponent hozza létre az objektumokat, tehát utána adjuk meg a ConnectionString értékét!

```
public JelszoForm(string database, string server)
{
 InitializeComponent();

 string conStr ="Persist Security Info=False;Integrated
                                  Security=SSPI;database=";
   conStr += database;
   conStr += ";server=" + server + ";";
 this.jelszoConnection.ConnectionString = conStr;

 jelszoAdapter.Fill(jelszoSet);
 foreach (JelszoSet.UsersRow row in jelszoSet.Users)
   userBox.Items.Add(row.user_ID);
}
```

Ezután a hívó alkalmazásban – amikor a JelszoForm objektumot létrehozzuk – a konstruktorát két paraméterrel, az adatbázis nevével és az SQL szerver nevével kell meghívnunk.

10.4. Feladatok

1. Készítsünk új adatbázist, mely egy másik alkalmazáshoz adja meg a felhasználókat és jelszavukat! Használjuk ezt az adatbázist a jelszóbekérésére!
2. Készítsünk egy új alkalmazást 'Megyék' néven! Az alkalmazás Magyarország megyéinek adatait mutatja be táblázatos formában. Az adatokat lehessen módosítani is!*
3. Tároljuk a rendelkezésünkre álló video- és hangfájlok nevét és elérési útját adatbázisban! Készítsünk alkalmazást, mely az adatbázisban tárolt fájlokat lejátssza!

4. Készítsünk osztálykönyvtárat, mely egy játékprogramhoz tárolja az eddig játszó játékosok nevét és eredményét, és új játékos esetén felveszi azt a listába, régi játékos esetén pedig, ha a most elért eredmény jobb, mint az eddigi legjobb, akkor javítja azt.
5. Termékeket bemutató ablak. A termék lehet pulóver, autó, illatszer, könyv, utazási ajánlat ... Mindenki válasszon kedvére valót!
6. A Kapj el, az Amőba ... feladat továbbfejleszthető úgy, hogy az ablak méretét nem engedi változtatni, helyette fokozatot lehet választani a játék indításakor. A fokozat az ablakméretet állítja. Egy másik időzítő méri a játék idejét, és az indításkor bekért játékos nevén rögzíti azt, ha a játékos győz. Egy fájlban vagy adatbázisban tárolja az eddigi győzelmek rangsorát. Legyen dll, hisz sok játék használhatja!
7. Készítsünk alkalmazást, mely egy rendszer felügyeletét ellátó személyek ügyeleti nyilvántartását kezeli! Az adatokat adatbázisban tárolja. Lehessen meghatározni, kinek hány órát kell kifizetni az ügyeletre!

10.5. Megoldásötletek

2. Megyék adatai táblázatosan.

Készítsünk Access adatbázist a megyék adatainak tárolására! Vegyük föl a Server Explorerbe a Data Connections közé!

A Server Explorerből általunk választott tábla Formra húzásával az OleDbConnection és OleDbDataAdapter típusú objektum létrejön. Nevük: megyeDbConnection, megyeDbDataAdapter.

Az adatokat egy táblázatban szeretnénk az alkalmazásunkban látni. Ehhez egy DataGrid vezérlőt húzzunk a Formra a ToolBox Windows Forms listájáról! A táblázat neve legyen megyeGrid! A táblázat adatforrásként (DataSource) egy DataSet objektumot vár, mely biztosítja a táblázatban megjelenő adatokat. Ezt a DataAdapterből generálhatjuk.

MegyeForm.cs[Design]

megyeDbDataAdapter jobbegér

Generate Dataset

New : megyeDataSet

Choose which table(s) to add to the dataset:

Megyék

Add this dataset to the designer.

OK.

Ezután már kiválaszthatjuk a dataset objektumot a gridben történő megjelenítéshez.

MegyeForm.cs[Design]

megyeGrid kijelölve

Properties

DataSource: megyeDataSet.Megyék

Már a tervező módban látjuk, hogy megjelennek a táblázat címsorában az adatbázisban kiválasztott tábla mezőnevei.

Futtatva az alkalmazást, még üres a táblázat. Az adatok beolvasását a megyeDataSetbe az ablak megnyitásakor még el kell végeznünk! A Form Load eseményének kezelőjébe (duplakattintás a formon):

```
private void MegyeForm_Load(object sender,
                                    System.EventArgs e)
{
 try
 {
   megyeDbDataAdapter.Fill(megyeDataSet);
 }
 catch (System.Exception ex)
 {
   MessageBox.Show(ex.Message);
 }
}
```

Az adatok az alkalmazáson keresztül szerkeszthetők (törölhetők, módosíthatók, új adatok vehetők fel).

Az adatok módosítása csak a megyeDataSet-ben történik meg. Ha azt akarjuk, hogy a módosítás az adatbázisban is megtörténjen, akkor az alkalmazás befejezésekor módosítsuk az adatbázist!

```
protected override void Dispose( bool disposing )
{
  try
  {
     megyeDbDataAdapter.Update(megyeDataSet);
  }
  catch (System.Exception ex)
  {
     MessageBox.Show(ex.Message);
  }
  if( disposing )
  {
     if (components != null)
     {
         components.Dispose();
     }
  }
  base.Dispose( disposing );
}
```

11. Gyakorlat. Web alkalmazás készítése

11.1. A szükséges előkészületek

Ha a Visual studio telepítéséhez nem volt szükség az IIS-re, akkor eddigi feladataink IIS nélkül is megoldhatók voltak. Azonban ha Web alkalmazást (ASP.NET Web Application), hálózati szolgáltatót (ASP.NET Web Service) vagy hálózati alkalmazások vezérlőit (Web Control Library) szeretnénk készíteni, azt csak akkor engedélyezi a Visual Studio, ha operációs rendszerünk telepítése után telepítettük az Internet Information Servicest (IIS). Ha ezt eddig nem tettük meg, a szolgáltatás utólag is telepíthető a Windows operációs rendszer CD-jéről.

Ha van **IIS** azt a Vezérlőpult / [Teljesítmény és karbantartás /] Felügyeleti eszközök közt találjuk meg. (11. Gyakorlat. 2. ábra)

Ahhoz, hogy az elkészült kódot egy böngészőben futtatni tudjuk, a helyi hálózat proxykiszolgálójának beállítása szükséges. (A beállításokat az Internet Explorer / Eszközök / Internet beállítások / Kapcsolatok / LAN beállítások alatt is megtehetjük.)

Visual Studio

Tools

Options

Projects

Web Settings

Connection Settings

LAN beállítások:

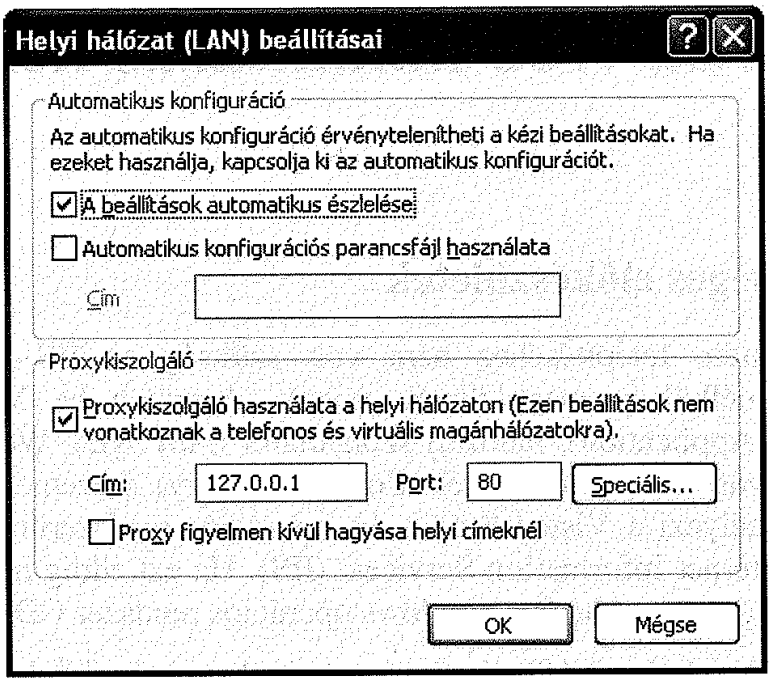
Proxikiszolgáló használata a helyi hálózaton igen

Cím és Port megadása a számítógép beállításainak megfelelően.

Az IP címet a Vezérlőpult /[Hálózati és Internetes kapcsolatok /] Hálózati kapcsolatok / Helyi kapcsolat / Támogatás fülét választva olvashatjuk ki. Ehhez élő Internetkapcsolattal kell rendelkeznünk. Mindenképp működik a localhost IP címe, a 127.0.0.1, amit a generált Web alkalmazás Web.config fájljában is megnézhetünk:

```
stateConnectionString="tcpip=127.0.0.1:42424"
```

a kettőspont előtti szakasz.



11. Gyakorlat. 1. ábra A proxykiszolgáló beállítása

11.2.Készítsünk Web alkalmazást!

File menü

New

Project

Project Types: Visual C# Projects

Templates: ASP.NET Web Application

Location: http://localhost/Tanfolyam

Close Solution

OK

A projekt létrehozása az eddigieknél egy kicsit tovább tart.

Ha megnézzük az elkészült WebForm1.aspx formot, az a Windows Formhoz hasonlóan szerkeszthető, csak az ablak alján, a Design fül mellett van egy HTML nézet is. Vagyis szerkeszthetjük a tervezőeszközzel, de írhatjuk az oldal HTML kódját is. A fejlesztés során a két lehetőséget felváltva szoktuk alkalmazni.

Természetesen a gyorsmenüben a View Code menüpontot választva a C# forráskódot is megnézhetjük WebForm1.aspx.cs fájlban.

Nevezzük át:

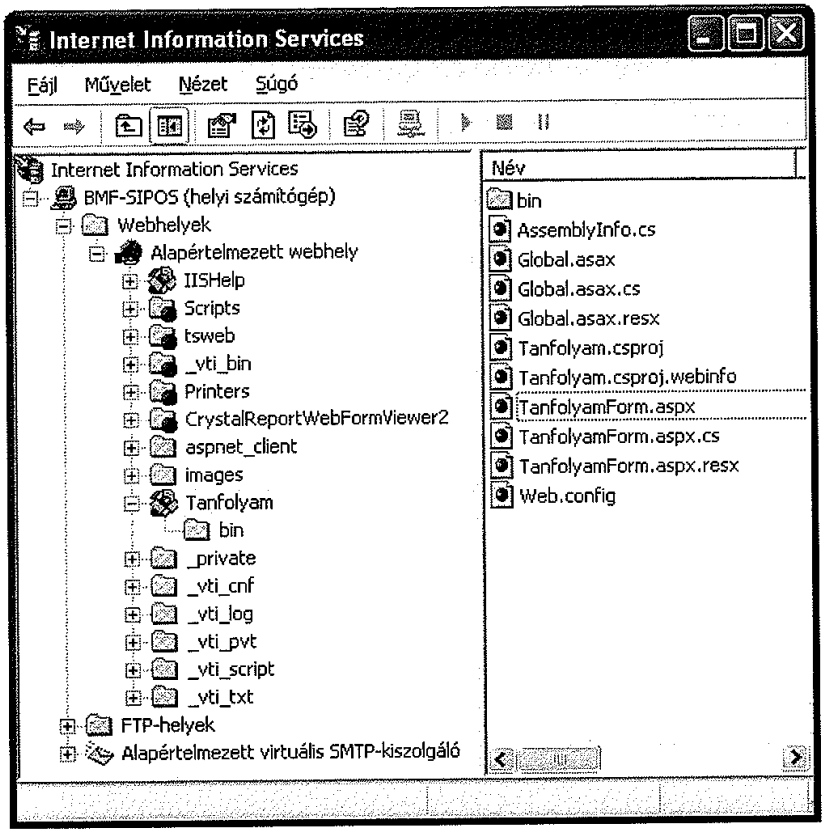
1. a Solution Explorerben WebForm1.aspx-ről TanfolyamForm.aspx-re!
2. a Class Viewban az osztály nevét!
3. a Kód megjegyzését!
4. a TanfolyamForm.aspx Properties ablakában (vagy a HTML kódban) a title tulajdonságot! Az oldal címe, ha azonnal nem látszik is, több helyen fontos az interneten. A keresőprogramok ezt szokták kiírni, de ha mi futtatjuk az oldalt a saját böngészőnkben, a Start menü tálcájára is az oldal címe (title) kerül. (Nem elég a frissítés, újrafordítás kell!)
5. a TanfolyamForm.aspx HTML nézetében form id="Form1" azonosítót!

A generálás során megadtuk a kód helyét. A http://localhost/Tanfolyam/TanfolyamForm.aspx címet a böngészőnkbe írva meg is nézhetjük a Web oldalt a Build Solution után. (Egyelőre üres.) A Start parancsot választva a felépítés után ki is nyitja nekünk a böngészőt az oldallal.

Fordítás/Futtatás

Hol tárolja a Visual Studio a projekt fájljait? A solution fájlt a Documents and Settings / Felhasználó név / Dokumentumok / Visual Studio Projects alatti projekt névvel megegyező alkönyvtárban. A Felhasználó név / VSWebCache / projekt neve alkönyvtár a fordítás helye. A kész fájlok publikálása, ahonnét elérhetjük őket a böngészővel, a C:\Inetpub\wwwroot könyvtár alatti, a projekt nevével megegyező alkönyvtárban történik. Ide másolja őket a VS.

Ha megnézzük az **IIS** nyilvántartását, a saját gépünk Webhelyek / alapértelmezett webhely alatt megtaláljuk a Tanfolyam mappát. Az IIS-ben a Művelet menü Csatlakozás menüpontjával tudunk további számítógépekhez kapcsolódni. A Webhelyek adatai közt a Port információt is megtaláljuk.



11. Gyakorlat. 2. ábra Az **IIS** a Tanfolyam Web alkalmazással

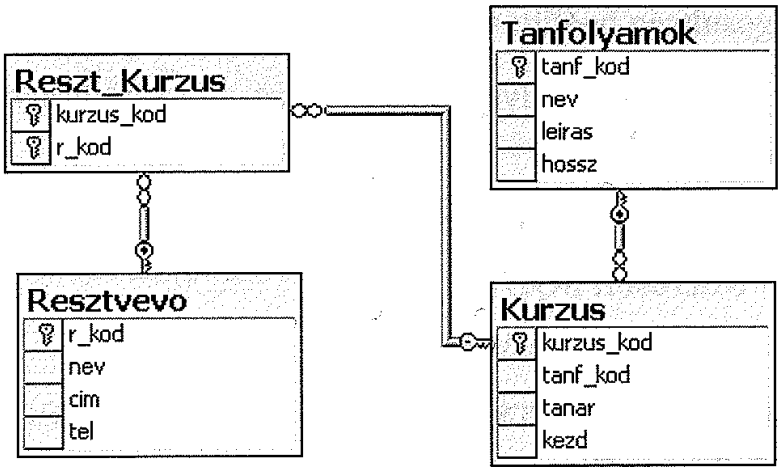
A fejlesztés során a web formot a böngészőben nézzük. Ha nem akarjuk a böngészőt minden alkalommal újraindítani és bezárni, célszerű nem a Visual Studioból, hanem önállóan elindítani. Így persze minden fordítás után frissíteni kell a böngészett oldalt. Ezt a honlap szerkesztői már megszokhatták, nem elég módosítani a kódot, csak akkor látjuk a változást, ha a böngészőben frissítünk.

A böngészőt a Visual Studio környezet elemeként is használhatjuk, ha a TanfolyamForm.aspx oldalon a gyorsmenüben a View in Browser menüpontot választjuk. A böngésző szokásos Web eszközsora a frissítő ikonnal a Visual Studio eszközsorai közt jelenik meg.

11.3. Az adatok megjelenítése

11.3.1. Az adatbázis létrehozása

Válasszuk a Server Explorerben a saját SQL Serverünket vagy az Accesst. Az új adatbázis neve legyen Tanfolyam! Táblái: Tanfolyamok, Kurzus, Reszt_Kurzus és Résztvevo. Az adatbázis szerkezete a következő:



11. Gyakorlat. 3. ábra A Tanfolyam adatbázis szerkezete

11.3.2. Az adatok megjelenítése

Tegyünk a TanfolyamForm.aspx Design nézetének felületére a Web Forms kategóriából egy Button vezérlőt! Az ID legyen fillButton, a Text tulajdonsága: Feltölt!

Húzzuk a TanfolyamForm.aspx Design nézetének felületére a Tanfolyamok táblát! Az objektumok neve legyen tanfConnection, tanfDataAdapter! Az adapterből generáljuk a TanfDataSet-et!

A Toolboxból húzzunk a formra egy DataGrid objektumot tanfDataGrid ID-vel! A DataSource tulajdonságához válasszuk ki a tanfDataSet1 objektumot! DataMember tulajdonsága legyen a Tanfolyamok tábla neve! A szerkesztő nézetben (Design) táblázat címsorában megjelennek a mezőnevek.

Feltölt gomb választása során töltsük fel a DataSet-et adatokkal, és csatoljuk az adatokat a táblázathoz!

```
private void fillButton_Click(object sender,
                                    System.EventArgs e)
{
 tanfDataAdapter.Fill(tanfDataSet1);
 tanfDataGrid.DataBind();
}
```

Fordítás/Futtatás/Tesztelés

Start választására megjelenik a Feltölt gomb a böngészőben, majd a gomb választása után a *Login failed for user 'SZERVER-NÉV\ASPNET'.* hibaüzenetet kapunk. Vagyis ezen a néven szeretnénk hozzáférni az adatbázishoz, de ilyen néven nincs hozzá jogosultságunk.

11.3.2.1. A jogosultság beállítása

Adjunk az SQL szerverünkben ilyen felhasználónéven jogosultságot!

Start menü

Minden program

Microsoft SQL Server

Enterprise Manager

Microsoft SQL Servers

SQL Server Group

local

Security

Logins jobbegér

New Login

General fül **Name** hárompontos gomb

Hozzáadandó név: SZERVER-NÉV\ASPNET

OK

Authentication: Windows Authetication

Domain: SZERVER-NÉV

Security access: Grant access

Database: Tanfolyam

Language: <Default>

OK

Ne csukjuk be az SQL Server Enterprise Managert!

Fordítás/Futtatás/Tesztelés

A hibaüzenet a következőre módosult:

Cannot open database requested in login 'Tanfolyam'. Login fails. Login failed for user 'BMF-SIPOS\ASPNET'

Tehát már konkrétan a Tanfolyam adatbázissal van a hozzáférési probléma.

Databases

Tanfolyam

Users jobbegér

New Database User

Login name: SZERVER-NÉV\ASPNET

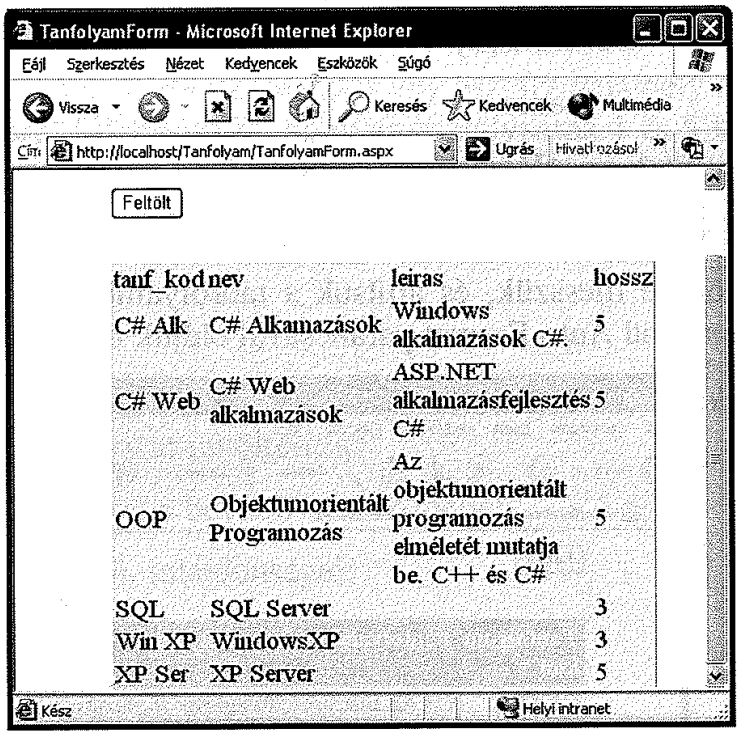
User name: SZERVER-NÉV\ASPNET

Database role membership: public, db_datareader

OK

Fordítás/Futtatás/Tesztelés

Most már hibátlanul megjelent a böngészőben az adatbázis tartalma a táblázatban.



11. Gyakorlat. 4. ábra A Tanfolyam adatbázis Tanfolyamok táblájának adatai a böngészőben

Megjegyzés:

Az olvasás jogot a Tanfolyam adatbázisra már a New Login létrehozásakor, a Database Access fülben is beállíthattuk volna.

11.3.2.2. A táblázat formázása

A Grid tulajdonságlapján:

BackColor: ControlLight

BorderColor: ControlDark

A Grid vezérlő gyorsmenüjében: Property Builder

A Columns fülben vegyük ki a Create columns automatically at run time dobozból a pipát, mert szeretnénk az oszlopok feliratait megváltoztatni! Az Available columns kategóriából tegyük át egyenként az oszlopokat a Selected columns kategóriába (esetleg a kód elhagyható)! Majd egyenként adjuk meg a Header text mezőben a kívánt fejlécnevet! A tanfolyam kódja legyen Read only!

A tanfolyam hosszát megadó napok számát szeretnénk középre rendezni!

Format

Columns

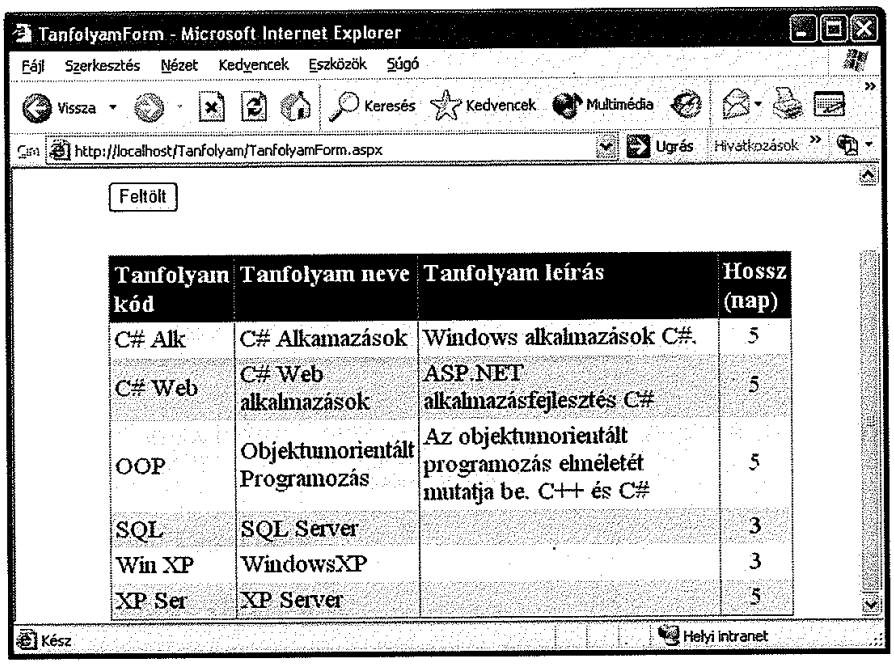
Columns[3]

Items

Horizontal alignment: Center

OK

A fejlécet fölfelé illesszük, és állítsuk a tanfolyamleírás szélességét! A színezést pedig a gyorsmenü Auto Format parancsával oldjuk meg.



11. Gyakorlat. 5. ábra A formázott Tanfolyamok tábla

! Ha elveszítenénk a localhost beállítást (nem tudjuk az aspx alkalmazásokat futtatni és nem tudunk web alkalmazást generálni), az IIS / Alapértelmezett webhely / Művelet / Tulajdonságok / Webhely / IP címet állítsuk a localhostnak megfelelő 127.0.0.1-re!

11.4. Feladatok

1. Készítsünk Web alkalmazást, mely két bevitt dátum esetén gombnyomásra megadja a közöttük eltelt napok számát! Adjuk meg az eltelt időt dátumértékként is, azaz év, hó, nap felbontásban!
2. Bővítsük a Tanfolyam feladatot úgy, hogy a kiválasztott tanfolyamhoz sorolja fel a kurzusokat, s az adott kurzus esetén lehessen megnézni a jelentkezettek névsorát!
3. Készítsünk egy Web Formot, melyen születési dátumadatokat kérünk be külön évszám, hónap, nap mezőkbe! Az évszám legyen nagyobb, mint 1900, és legfeljebb az idei évvel egyező legyen! A hónap-, napadatok a dátumnak megfelelőek legyenek! Adjunk részletes indoklású hibaüzeneteket! Használjunk Validator objektumokat az ellenőrzéshez!
4. **Osztályzatok.** Készítsünk Web alkalmazást egy osztály valamely tantárgyának osztályzatai bevitelére! Ellenőrizzük, hogy az osztályzatok 1 és 5 közötti egész számok legyenek, és ne lehessen ugyanazt a személyt kétszer felvinni!*

11.5.Megoldásötletek

4. Osztályzatok.

Készítsük el az adatbázist, a DataSet osztályt generáljuk le, és olvassuk be a dataSet objektumba az adatokat feltöltéskor!

Tegyünk ellenőrző vezérlőket a beviteli mezők mellé!

Egy RangeValidator vezérlő, tulajdonságai:

ID: jegyRangeValidator

ControlTo Validate: jegyTextBox

ErrorMessage: A jegy 1 és 5 közötti szám lehet!

MaximumValue: 5

MinimumValue: 1

Type: Integer

Ha a felvitelButton alapértelmezett CausesValidation tulajdonságát True értéken hagytuk, a gomb választásakor megtörténik az ellenőrzés. Ezt tesztelhetjük, ha a gombhoz egy üres kezelőmetódust rendelünk.

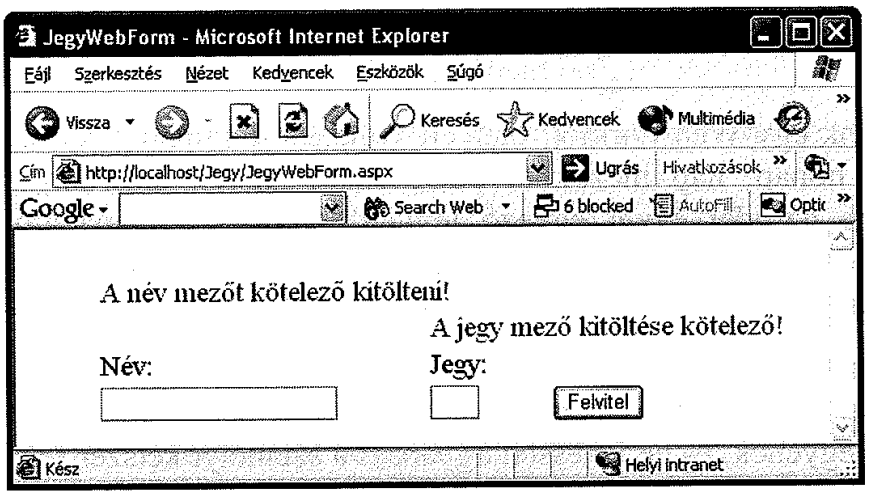
A tesztelés során kiderül, az ellenőrzés jól működik, kivéve, hogy üres mező esetén nem küld hibaüzenetet. Mivel egy mezőhöz több ellenőrző vezérlőt is rendelhetünk, húzzunk a felületre egy RequiredFieldValidatort!

ID: jegyRequiredFieldValidator

ControlTo Validate: jegyTextBox

ErrorMessage: A jegy mező kitöltése kötelező!

Hasonló módon a név mezőt se lehessen üresen hagyni!



11. Gyakorlat. 6. ábra A RequiredFieldValidator működés közben

Mivel megengedtük a kliens oldali ellenőrzést, az előző ellenőrzések a kliens oldalon történtek. Ezt jól láthatjuk, ha a felvitel gomb választását követően a státuszsorban az adatátvitel folyamatkijelzőjét figyeljük. Ha a felvitel gomb eseménykezelőjébe töréspontot teszünk láthatóan az eseménykezelő nem hívódik meg.

Annak ellenőrzését, hogy ugyanazt a személyt ne vihessük föl kétszer, már a szerver oldalon fogjuk megoldani. Ehhez egy CustomValidator vezérlőt használunk.

ID: nameCustomValidator

ControlTo Validate: nameTextBox

ErrorMessage: Ez a név már szerepel a nyilvántartásban!

Az események között a ServerValidate eseményt kezeljük!

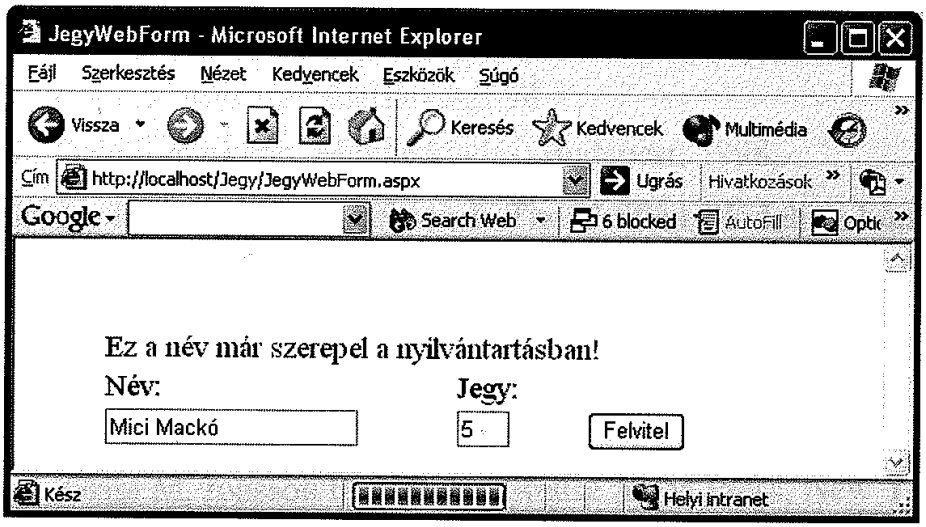
```
private void nameCustomValidator_ServerValidate(
 object source,
 System.Web.UI.WebControls.ServerValidateEventArgs args)
{
 args.IsValid=!jegyDataSet1.Osztályzatok.Rows.Contains(
                                        nameTextBox.Text);
}
```

Ha az Osztályzatok tábla sorai tartalmazzák a beírt nevet, akkor hamis legyen az érvényesség!

Most már nincs más tennivalónk, mint megírni a felvitelt!

```
private void felviszButton_Click(object sender,
                                        System.EventArgs e)
{
 if (nameCustomValidator.IsValid)
 {
    object[] values = {nameTextBox.Text,
                       jegyTextBox.Text, "Programozás"};
    jegyDataSet1.Osztályzatok.BeginLoadData();
    jegyDataSet1.Osztályzatok.LoadDataRow(values,false);
    jegyDataSet1.Osztályzatok.EndLoadData();
    nameTextBox.Text="";
    jegyTextBox.Text="";
    osztalyzatokSqlDataAdapter.Update(jegyDataSet1);
 }
}
```

Bár a LoadDataRow metódus önmagában is garantálja, hogy kulcsmezőt nem vihetünk föl többször, mégis a legegyszerűbb, ha meg sem hívjuk a felvitelt ismételt név esetén.



11. Gyakorlat. 7. ábra A CustomValidator figyelmeztetése

A CustomValidator szerver oldali ellenőrzését az adatátvitel kijelzéséből is látjuk a státuszsorban.

Irodalomjegyzék

1. Albert István, Balássy György, Charaf Hassan, Erdélyi Tibor, Horváth Ádám, Levendovszky Tihamér, Péteri Szilárd - Rajacsics Tamás: A .NET Framework és programozása, Szak kiadó, 2004.
2. A tantárgyak előadásaihoz használt segédletek. www.nik.hu/aii/oktatok/siposmarianna.html
3. Baga Edit: Delphi másképp, Magán kiadás, Budapest, 1998.
4. Booch Grady, Rumbaugh J, Jacobson I: The Unified Modeling Language User Guide, Addison Wesley Longman Inc., Reading Massachusetts, 1998.
5. Bradley L. Jones: C# mesteri szinten, Kiskapu Kft, 2004.
6. Gamma Erich, Helm R, Jonson R, Vlissides J: Design Patterns, Addison Wesley Longman Inc., Reading Massachusetts, 1995.
7. Jacobson I, Griss M, Jonsson P: Software Reuse ACM Press, New York, 1997.
8. Jacobson Ivar, Rumbaugh J, Booch G,: The Unified Software Development Process, Addison Wesley Longman Inc., Reading Massachusetts, 1999.
9. Jutta Eckstein, Mary Lynn manns, Helen Sharp, Marianna Sipos: Teching from Differnet Perspectives, EuroPLoP'03, Irsee, 2003. 165-183. old.
10. Madsen Ole Lehrmann Madsen, Henric Ron, Kresten Krab Thorup & Mads Torgersen, A Conceptual Approch to Teaching Object-Orientation to C

Programmers, Proceedings, Educator's Symposium, OOPSLA'98, Vancouver, ACM, New York, 1998.

11. Microsoft: Developing Microsoft.NET Applications for Windows Visual C#.NET, Microsoft Corporation 2002, Material No: 2555A
12. Microsoft: Developing Microsoft ASP.NET Web Applications Using Visual Studio .NET, Microsoft Corporation, 2002, Material No. 2310.
13. Microsoft: Introduction to C# Programming for the Microsoft.NET Platform, Microsoft Corporation, 2001, Material No. 2124B.
14. Microsoft: Querying Microsoft SQL Server 2000 with Transact-SQL, Microsoft Corporation, 2002, Material No 2071BCP
15. Microsoft .NET Framework SDK Documentation v1.1, Microsoft Corporation, 2002.
16. MSDN Library for Visual Studio .NET 2003 Microsoft Corporation, 2003.
17. MSDN Online: http://msdn.microsoft.com
18. Nyékyné Gaizler Judit: Java2 Útikalauz programozóknak I. ELTE TTK Hallgatói Alapítvány, Budapest, 2000.
19. Platt David S.: Bemutatkozik a Microsoft .NET, Szak Kiadó Kft, Bicske, 2001
20. Pethő Ádám: abC, Számalk Könyvkiadó, Budapest, 1991.
21. Rumbaugh James, Booch G, Jacobson I: The Unified Modeling Language Reference Manual, Addison Wesley Longman Inc., Reading Massachusetts, 1998.
22. Rebecca M. Riordan: Microsoft ADO.NET Step by Step, Microsoft Press, Redmond, Washington, 2002.
23. Sipos Mariann: Objektumorientált programozás a C++ nyelv lehetőségeivel, Gábor Dénes Főiskola, főiskolai jegyzet, Budapest, 2000.
24. Sipos Marianna: A Visual C++ és az MFC, második kiadás Budapest, 2001.
25. Visual C# Language 2004. 09. 28.: http://msdn.microsoft.com/library/en-us/cscon/html/vcoriCStartPage.asp
26. Visual Studio.NET C# 2001, Software Offline, Animare Software Kft, 2002.
27. Visual Studio.NET C# 2002, Software Offline, Animare Software Kft, 2003. www.SoftwareOnline.hu
28. Wikipedia the free encyclopedia http://wikipedia.org

Könyvismertető

Bradley L. Jones

C# mesteri szinten 21 nap alatt

Mindössze 21 nap alatt elsajátíthatjuk a C# nyelv alapjait, sőt, általános és webes alkalmazások készítésére egyaránt képesek leszünk. A könyv a C# szabványos változatára épül, így a megszerzett tudást nem csak a Microsoft Windowson, hanem a nyelvet támogató más operációs rendszereken, így Linuxon és FreeBSD-n is alkalmazhatjuk.

Nincs szükség előzetes C# vagy .NET tapasztalatokra. A 21 gondosan megszerkesztett leckét követve bárki megtanulhatja a C# használatát, aki legalább alapszintű programozói tudással rendelkezik.

Saját tempónkban tanulhatunk. Haladhatunk sorban a leckéken, hogy biztosan megértsünk minden fogalmat és eljárást, de választhatjuk azt is, hogy csak a számunkra újdonságot jelentő témaköröket vagy fogásokat taglaló fejezeteket tanulmányozzuk. Két-három órát szánva az egyes leckékre mindenki megtanulhatja a C# nyelvet.

Tudásunkat ellenőrizhetjük is. Minden fejezet végén kérdéseket és válaszokat, illetve a további tanulmányokat szolgáló gyakorlatokat találunk. Egyes kiegészítő fejezetek hosszabb példaprogramokat mutatnak be, amelyeket szintén kipróbálhatunk.

Adatok:

ISBN: 963 9301 73 6

Ára: 6930 Ft

Oldalszám: 768 oldal

Témakör: programozás, C#

Felhasználói szint: kezdő–profi

Web: kiado.kiskapu.hu/81

Melléklet: CD-ROM

Tárgymutató

Cs

D

E,É

F

G

H

I,Í

J

K

L

M

N

Ny

O,Ó

P

R

S

Sz

T

U,Ú

Ü,Ű

V

W

Y

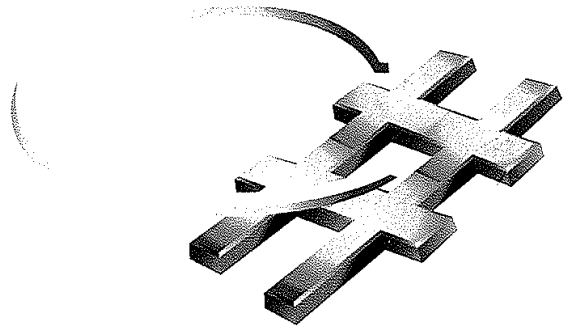
Sipos Marianna

PROGRAMOZÁS ÉLESBEN

.net

A kötet a .NET Windows alkalmazások fejlesztését mutatja be C# programozási nyelven.
A könyv címében szereplő "élesben" nemcsak a C# nyelv angol fordítására utal, hanem arra is, hogy a programozást nem feltétlen kell a kézzel begépelt kód, és a fekete-fehér konzolos képernyő világában kezdeni. Már az első pillanattól használhatjuk a grafikus felületet, az objektumorientált kódot az eseményvezérelt alkalmazásokhoz. A feladatok bár egyszerűek, mégis a felhasználó számára ismerős felületet és technikákat biztosítanak. A kötet forgatása során egyre mélyebben ismerjük meg a megoldások hátterében alkalmazott filozófiát, az objektumorientált programozástól a komponensalapú fejlesztésig. Kitekintést kapunk az adatbázis-feldolgozásra és a Web alkalmazások fejlesztésére.

9 789632 166520



A könyv újszerű szemléletén kívül a szerző előző köteteiben bevált szerkezeti tagolásával is támogatja a tanulást. Az első rész az elméleti fogalmakat, és a felhasználásukhoz szükséges mélyebb háttérismereteket tárgyalja. A második rész az azonos fejezetszámú elmélethez kapcsolódó részletes feladatmegoldást, valamint önálló feldolgozásra szánt feladatsort tartalma megoldásötletekkel. Ez a szerkezet lehetővé teszi a csoportos és az önálló tanulás támogatását egyaránt. Haladhat sorban, vagy választhatja az újdonságot jelentő fejezeteket. A kulcsszavas keresővel gyorsan megtalálhat egy fogalmat, vagy annak további előfordulásai A könyv feladatmegoldásai olyan részletesek, hogy a forráskódra nincs szükség, de letölthető a http://www.nik.hu/aii/oktatok/siposmarianna.html honlapró

Sipos Marianna
- a Visual C++ és az MFC könyv szerzője -
a BMF Neumann Informatikai Karán programozást és szoftvertechnológiát tanít. Jelentős szakmai és oktatásmódszertani tapasztalatokkal rendelkezik. Doktori disszertációját programozásoktatás témakörber írta. A könyv feladatait a BMF NIK műszaki informatika szakos és az ELTE IK programozó matematikus hallgatóival próbálta ki.

Felhasználói szint: Kezdő-haladó
Kategória: Programozás, C#
ISBN: 963 216 652 3