TÁMOP-4.1.1.F-14/1/KONV-2015-0009
„A GÉPÉSZETI ÉS INFORMATIKAI ÁGAZATOK DUÁLIS ÉS MODULÁRIS KÉPZÉSEINEK KIALAKÍTÁSA A PÉCSI TUDOMÁNYEGYETEMEN „

Achs Ágnes–Szendrői Etelka

Programozás 2., II. kötet
Windows form alkalmazások

Pécs
2015

A tananyag a TÁMOP-4.1.1.F-14/1/KONV-2015-0009 azonosító számú, „A gépészeti és informatikai ágazatok duális és moduláris képzéseinek kialakítása a Pécsi Tudományegyetemen” című projekt keretében valósul meg.

PÉCSI TUDOMÁNYEGYETEM
UNIVERSITY OF PÉCS

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

TÁMOP-4.1.1.F-14/1/KONV-2015-0009
„A GÉPÉSZETI ÉS INFORMATIKAI ÁGAZATOK DUÁLIS ÉS MODULÁRIS KÉPZÉSEINEK KIALAKÍTÁSA A PÉCSI TUDOMÁNYEGYETEMEN „

Programozás 2., II. kötet

Achs Ágnes, Szendrői Etelka

Szakmai lektor: Szabó Levente

Nyelvi lektor: Veres Mária

Kiadó neve

Kiadó címe

Felelős kiadó:

ISBN szám

Pécsi Tudományegyetem

Műszaki és Informatikai Kar

Pécs, 2015

PÉCSI TUDOMÁNYEGYETEM
UNIVERSITY OF PÉCS

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

TARTALOMJEGYZÉK

TÁBLÁZATOK JEGYZÉKE

ÁBRÁK JEGYZÉKE

1. Grafikus vezérlőkomponensek, az eseményvezérelt programozás alapelvei

1.1. Elméleti háttér *(Szendrői E.)*

Eddig konzolalkalmazásokat készítettünk, melyekre az a jellemző, hogy az algoritmusban elkészített folyamatok (eljárások) vezérlik az alkalmazás működését.

A felhasználók által közvetlenül használt programok barátságosabb felületet kívánnak a konzolalkalmazásoknál, akár vastag kliens-, akár webes alkalmazásról van szó. Ma már követelmény, hogy grafikus felületeken (Graphical User Interface (GUI)) történjen a felhasználó és az alkalmazás közötti párbeszéd. A grafikus felületű alkalmazásokra jellemző, hogy a felhasználó tevékenysége irányítja a folyamatokat azáltal, hogy kattint a felület egy pontján vagy legördít egy menüt vagy bejelöl egy választógombot stb., vagyis, a felhasználó tevékenysége különböző eseményeket, és arra történő válaszokat generál.

1.1.1. Eseményvezérelt (GUI) alkalmazások

A GUI alkalmazások jellemzője, hogy a felhasználó határozza meg a program futásának menetét a beavatkozásaival. A felhasználó egy grafikus, akár több ablakból (űrlap, form) álló felületen keresztül kommunikál a programmal, s a rendelkezésére álló eszközökkel (billentyűzet, egér stb.) avatkozik be a program futásába, aminek jellemző viselkedése, hogy vár a felhasználó következő jelzésére, és arra reagál.

Események (Events)

Az esemény (akció) a felhasználó (a billentyűzet, egér stb.) vagy más komponens (pl. az operációs rendszer) által okozott beavatkozás, amely a rá jellemző adatokkal együtt üzenetként jelenik meg. Például egy egérkattintásnál melyik egérgombot, hányszor nyomták meg, és mi volt ekkor az egérkurzor pozíciója.

Az üzenet egy sajátos mechanizmus segítségével jut el a megfelelő alkalmazáshoz (eseménykezelő ciklus), majd annak aktuális szálához.

Az eseményhez olyan kezelőfüggvényt vagy függvényeket kapcsolhatunk, amelyeknek meghívása az esemény kiváltásának hatására történik. A programozó feladata, hogy ilyen eseménykezelő metódust készítsen. Az eseménykezelő olyan kód, ami értesítést kap, ha egy esemény bekövetkezett. Megkapja az eseményt kiváltó objektumot (sender) és az esemény kezeléséhez szükséges valamennyi információt.

Tehát az esemény a vele összefüggő információkat magába foglaló objektum. Az események mindig valamilyen objektumon keletkeznek, például nyomógombon, listapanelen, szövegmezőn stb.

Az eseménykezelő metódusok visszatérési típusa mindig void és két paraméterük van, az első az eseményt kiváltó objektum (sender) (pl. a gomb), ez mindig object típusú, a második pedig az EventArgs osztályból származtatott objektum, amely az esemény információit tartalmazza. Ez utóbbinak több specializált változata van (pl.: külön a billentyűzet és egér által kiváltott eseményekhez).

Grafikus felület

A grafikus felhasználói felület alkotóelemei a rajta látható objektumok, melyeket a .NET környezetben vezérlőelemeknek (controls) hívják. A felületre elhelyezni kívánt objektumok a grafikus felület elkészítését támogató osztályok példányai. Ezek az osztályok a *System.Windows.Forms* névtérben találhatók.

Az ablak, amellyel a felhasználó párbeszédet folytat, a **Form** osztály egy példánya. A Form osztály leszármazottja a *ContainerControl* osztálynak. A ContainerControl osztály leszármazottja a *ScrollableControl* osztálynak, amely leszármazottja a *Control* osztálynak.

A vezérlőelemek (controls), amelyek a komponens (*Component*) osztály leszármazottjai, jellegzetes alakú és funkciójú elemek, amelyeken események válthatók ki, és ezekhez kezelőfüggvényeket rendelhetünk.

Maga a **Form** osztály konténer típusú, (a *ContainerControl* osztályból származik), így tartalmazhat más vezérlőket is, ami azt jelenti, hogy a formoknak és a vezérlőknek sok közös tulajdonságuk és metódusuk van. Minden GUI vezérlőnek 80 *public* és 20 *protected* tulajdonsága van, melyet a Control osztálytól örökölt.

1. táblázat A Control osztály néhány publikus tulajdonsága

Tulajdonság	Leírás
Anchor	Beállítja / Megadja, hogy a vezérlő melyik élei rögzítettek.
BackColor	Beállítja / Megadja a vezérlő háttérszínét.
BackgroundImage	Egy háttérképet állít be/ ad meg a vezérlőnek.
CanFocus	Egy értéket ad vissza, amely megadja, hogy a vezérlő kaphat-e fókuszt.
CanSelect	Egy értéket ad vissza, amely megadja, hogy a vezérlő kiválasztható-e.
Enabled	Egy értéket állít be / ad meg, amely jelzi, hogy a vezérlő aktiválva van-e.
Focused	Egy értéket ad vissza, amely megadja, hogy a vezérlőnek van-e bemeneti fókusza.
Font	A vezérlő aktuális betűtípusát állítja be/adja meg.
ForeColor	A vezérlő előtérszínét állítja be/adja meg.

1. táblázat A Control osztály néhány publikus tulajdonsága (folytatás)

Tulajdonság	Leírás
Location	A vezérlő bal felső sarkának koordinátáit adja meg a konténerének bal felső sarkához viszonyítva.
Name	Beállítja / Megadja a vezérlő nevét.
Size	Beállítja / Megadja a vezérlő magasságát és szélességét.
Tabindex	Beállítja / Megadja a vezérlő tabulátor sorrendjét a konténerében.
TabStop	Egy értéket állít be / ad meg, amely jelzi, hogy a felhasználó Tab billentyűvel elérheti-e a vezérlőt.
Text	Beállítja / Megadja a vezérlőhöz kapcsolódó szöveget.
Visible	Egy értéket állít be / ad meg, amely jelzi, hogy a vezérlő látható-e.

A következő táblázatban a Control osztály néhány fontos metódusát soroljuk fel, amelyeket minden vezérlőelem örököl.

2. táblázat A Control osztály legfontosabb publikus metódusai

Metódus	Leírás
Focus()	Fókuszt ad a vezérlőelemnek.
Hide()	Elrejti a vezérlőt.
Select()	Aktiválja a vezérlőt.
Show()	Megjeleníti a vezérlőt (a láthatósági tulajdonságát *true*-ra állítja)

A Form

Korábban említettük, hogy a Form (ablak) a felhasználó és az alkalmazás közötti kapcsolat kulcseleme; jellemzője a mérete (méretezhetőség), helye (mozgatás), keret stílusa, címe, háttérszíne stb. A form tetején van a fejlécsor, amely a form címét mutatja meg. A fejlécsor jobb oldalán vannak az ablak vezérlőgombok, a *Kisméret/Teljes képernyő* és a *Bezárás* gombok. A fejlécsor alatt találjuk a *menüsort*, ha egyáltalán van az ablaknak menüsora. A menüsor alatt helyezkedhetnek el a form eszköztárai. A form fő területe, a felhasználói terület (client area), itt történik minden. A formra helyezett vezérlők együtt mozognak az ablakkal. Egy alkalmazásnak több ablaka is lehet, ezért fontos a formok egymáshoz való viszonya, melyik van fókuszban, milyen takarási sorrendben vannak, szülő–gyerek ablakról van-e szó, modális vagy nem modális. A Form osztálynak egy statikus tulajdonsága van, az

ActiveForm, amely az egész alkalmazás számára tartalmazza az aktuálisan aktív formot. Ha meg akarjuk határozni, hogy melyik ablakon van a fókusz (azaz a billentyűzetüzenetek irányítási pontja), használjuk az AktívForm tulajdonságot. A Form osztálynak is számtalan példánytulajdonsága van, és természetesen rendelkezik az őseitől megörökölt tulajdonságokkal is. A következő táblázatban a Form *néhány* publikus *példánytulajdonságát* soroljuk fel (a Control osztálytól örökölt tulajdonságok nem szerepelnek a táblázatban).

3. táblázat A Windows Form osztály néhány publikus példánytulajdonsága

Tulajdonság	Leírás
AcceptButton	Azt a gombot állítja be / adja meg a formon, amelyik lenyomódik, amikor a felhasználó Entert üt.
AllowDrop	Azt jelzi, hogy a form fogadhat-e olyan adatot, amelyet a felhasználó áthúz rá.
AutoScale	Azt jelzi, hogy a form átméretezheti-e magát és vezérlőit, hogy illeszkedjen a használt betűtípushoz.
AutoScroll	Azt jelzi, hogy a form végrehajt-e automatikus görgetést.
BackColor	A form háttérszínét adja meg.
Cancel Button	Azt a gombvezérlőt jelzi, amelyik akkor nyomódik le, ha a felhasználó ESC billentyűt üt.
ControlBox	Egy olyan értéket állít be, amely jelzi, hogy a vezérlődobozok megjelenjenek-e az ablak fejlécsorában.
Controls	A formon lévő vezérlők gyűjteményét adja meg / állítja be.
DialogResult	Megadja / beállítja, a formnak a párbeszédablak eredményét.
FormBorderStyle	A form szegélyének a stílusát adja meg /állítja be.
Height	A form magasságát adja meg /állítja be.
MaximumSize	A form lehetséges teljes méretét adja meg.
MaximizeBox	Egy olyan értéket állít be, amely jelzi, hogy a form fejlécsorában megjelenik-e a teljes méretre állító gomb.
MinimizeBox	Egy olyan értéket állít be, amely jelzi, hogy a form fejlécsorában megjelenik-e a kis méretre állító gomb.
Modal	Egy olyan értéket állít be, amely jelzi, hogy a form modálisan (a vezérlést megtartva) jelenik meg.
Name	A form nevét adja meg / állítja be.
Size	A form méretét adja meg.

3. táblázat A Windows Form osztály néhány publikus példánytulajdonsága (folytatás)

Tulajdonság	Leírás
StartPosition	A form futásidő alatti kezdőpozícióját adja meg.
Text	A form címét adja meg, amely a fejlécsorban jelenik meg.
Visible	Egy olyan értéket állít be, amely jelzi, hogy a form látható-e.
Width	A form szélességét adja meg /állítja be.

A formok a tulajdonságokon túl metódusokkal is rendelkeznek. A következő táblázatban a formok néhány, a programozási gyakorlataink során leggyakrabban használt, publikus példánymetódusait soroljuk fel.

4. táblázat A Windows formok néhány publikus példánymetódusa

Metódus	Leírás
Activate()	Aktiválja a formot (fókuszt ad neki és aktívvá teszi).
Close()	Bezárja a formot.
Dispose()	felszabadítja a form által lefoglalt erőforrásokat.
Focus()	Fókuszt ad a formnak.
Hide()	Elrejti a formot.
Show()	A Visible tulajdonság igazra állításával megjeleníti a formot.
ShowDialog()	A formot modális párbeszédablakként (a vezérlést magánál tartva) jeleníti meg.

A Windows formok különböző eseményeket kezelnek. Az események tudatják velünk, hogy valami történt a formmal, például a formra kattintáskor egy Click esemény következik be, vagy amikor a form bezárult, egy Closed esemény történik.

5. táblázat A Windows formok néhány publikus objektumeseménye

Esemény	Leírás
Activated	Akkor következik be, amikor a form a kódban vagy a felhasználó által aktiválódik.
Click	Akkor történik, amikor a formra kattintunk.
Closed	Akkor következik be, miután a form bezárult.
Closing	Akkor következik be, amikor a form bezárul.

5. táblázat A Windows formok néhány publikus objektumeseménye (folytatás)

Esemény	Leírás
Deactivate	Akkor következik be, amikor egy form elveszíti a fókuszt, és már nem ő az aktív form.
Enter	Akkor történik, amikor belépünk a formra.
GotFocus	Akkor következik be, amikor a form megkapja a fókuszt.
KeyDown	Akkor történik, amikor egy billentyűt leütöttünk, miközben a fókusz a formon van.
KeyPress	Akkor következik be, amikor egy billentyűt lenyomunk, miközben a formon van a fókusz.
KeyUp	Akkor következik be, amikor egy billentyűt felengedünk, miközben a formon van a fókusz
Load	Azelőtt történik meg, mielőtt egy form először megjelenne a felhasználó előtt.
LostFocus	Akkor történik, amikor a form elveszíti a fókuszt.
MouseDown	Akkor következik be, amikor az egérmutató a form felett van és az egér gombját lenyomtuk.
MouseEnter	Akkor következik be, amikor az egérmutató rámegy a formra.
MouseHover	Akkor történik meg, amikor az egérkurzor a form felett megállapodik.
MouseLeave	Akkor következik be, amikor az egérmutató elhagyja a formot.
MouseMove	Akkor történik meg, amikor az egérmutató a form fölött mozog.
MouseUp	Akkor következik be, amikor az egérmutató a form felett van és felengedjük az egérgombot.
MouseWheel	Akkor következik be, amikor az egér kereke megmozdul, mialatt a fókusz a formon van.

1.1.2. Grafikus felület szerkesztése

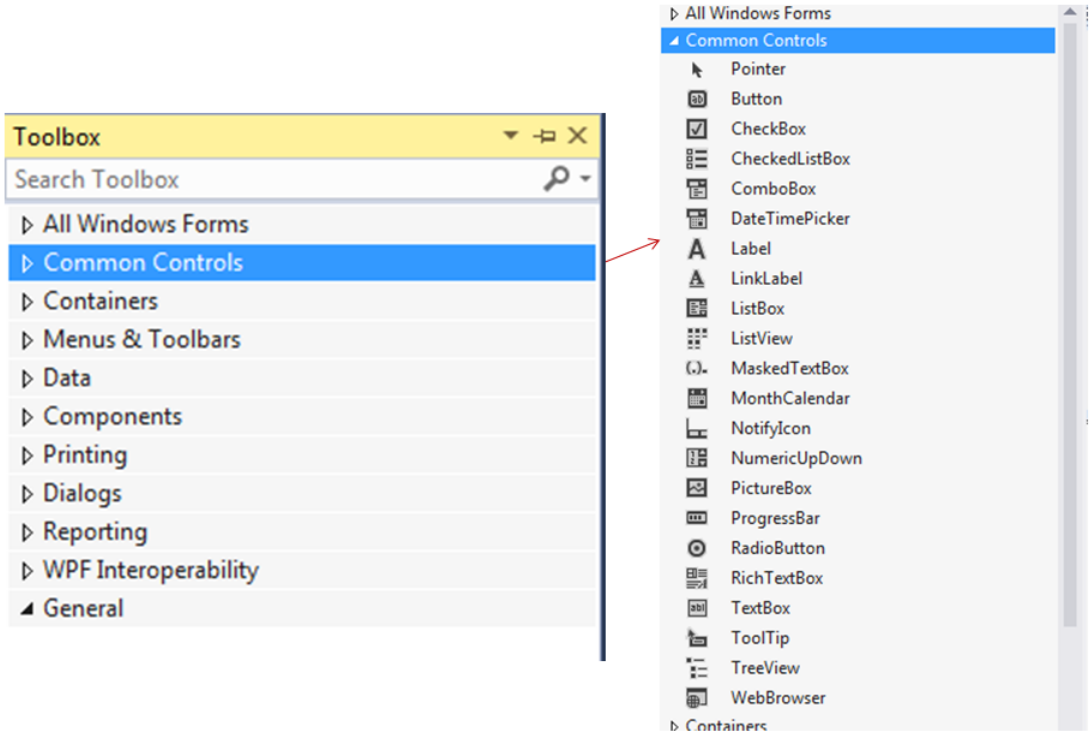
A fejlett integrált programfejlesztő eszközök egy felülettervezőt (szerkesztő, designer) biztosítanak a GUI alkalmazások készítéséhez. A felülettervezők támogatják

- a vezérlőelemek "drag&drop" technikával való formra helyezését, elrendezését
- a vezérlők kezdeti tulajdonságainak beállítását
- az eseményekhez eseménykezelők hozzárendelését.

A tervezés eredménye egy, a szerkesztő által generált, magas szintű programozási nyelvű kód lesz, amely beépül az alkalmazásunkba.

A Visual Studio ToolBox

A Visual Studio fejlesztőkörnyezete az eszközpaletta segítségével támogatja a fejlesztő munkáját a formok elkészítésében. Az egyes vezérlőelemek csoportokba vannak szervezve, így könnyebb eligazodni közöttük.



1. ábra A Visual Studio Toolbox ablaka

1.1.3. Windows Form alkalmazások végrehajtása

A Windows Form C# alkalmazásunk létrehozásakor automatikusan létrejön egy osztály, általában Form1 néven, amely arra szolgál, hogy biztosítsa számunkra a grafikus felhasználói felületet, amelyre vezérlőkomponenseket elhelyezve, vezérelni tudjuk az alkalmazás működését. A felhasználói interakció következményeként bekövetkező esemény hatására történik a program meghatározott részének a végrehajtása.

Az alkalmazás vezérlő osztálya a Form1 lesz, amely adattagokat, metódusokat, tulajdonságokat, eseményeket szolgáltat számunkra. Ahhoz, hogy a Form1 vezérlő osztályt használni tudjuk, objektumpéldányt kell létrehozni belőle, s majd ezen a példányon keresztül fogja ellátni feladatát. Kérdés, hol kell létrehozni ezt az objektumot? A válasz ugyanaz, mint a konzolalkalmazásoknál, a program belépési pontján, azaz a Main() metódusban. Tehát a Windows Form programok végrehajtása, hasonlóan a konzolalkalmazásokhoz, a Main() metódusban kezdődik. Minden C# alkalmazás tartalmaz egy Program.cs nevű fájlt, ahogy ezt már korábban tapasztaltuk, s ebben a fájlban találjuk meg a Program osztályt, benne a Main() metódussal, melynek feladata, hogy elindítsa az alkalmazásunkat.

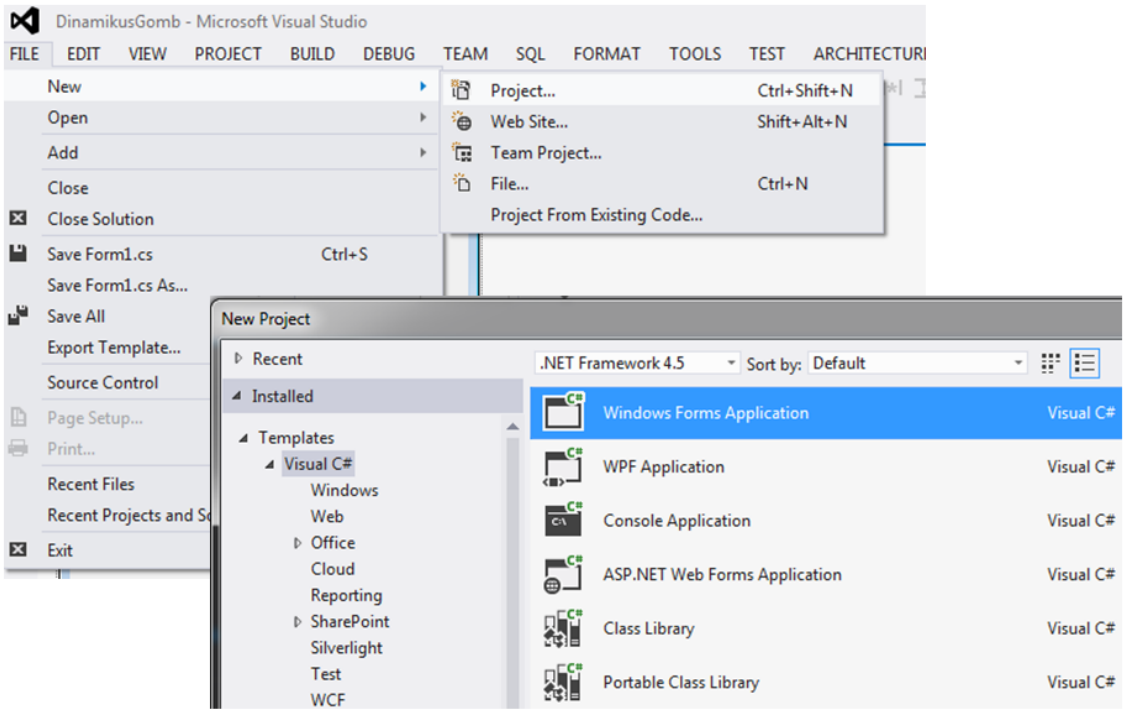
Windows Form alkalmazásokban a Main() metódus létrehoz egy Form objektumot, majd meghívja az **Application** osztály statikus **Run**() metódusát, melynek paramétere lesz a Form objektum. Az Application osztály szintén a *System.Windows.Form* névtérben van. Az *Application.Run()* metódushívást az alkalmazás kezdő ablakának megjelenítésére használjuk. A *Run()* metódus elindítja az alkalmazás futását, míg a statikus *Exit()* metódus befejezi azt.

```
0 references
static void Main() {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());

}
```

A további formokat, ha szükséges, a saját kódunkkal jeleníthetjük meg, a Show() vagy ShowDialog() metódushívásokkal.

1.1.4. Új Windows Form alkalmazás létrehozása

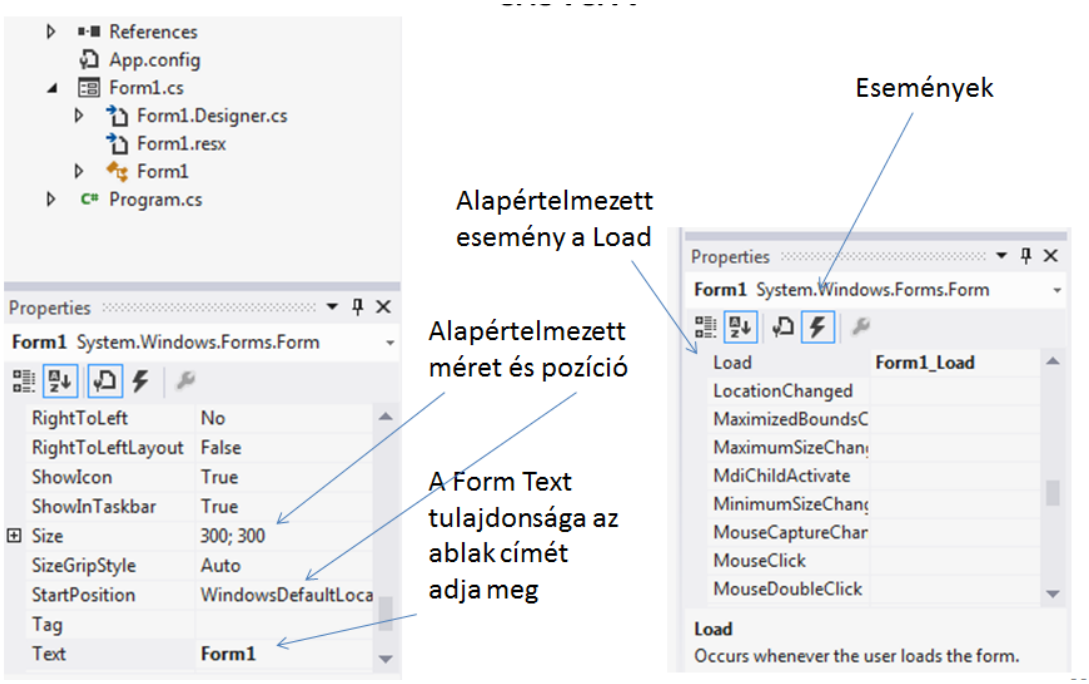
Új Windows Form alkalmazás létrehozása a Visual Studio New Project menujének kiválasztásával kezdődik. A megjelenő New Project ablakban kell megadnunk, hogy milyen nyelven és milyen típusú alkalmazást kívánunk létrehozni. Esetünkben a *Windows Forms Applicatio*n kell választanunk.



A Solution Explorer és a Properties ablak

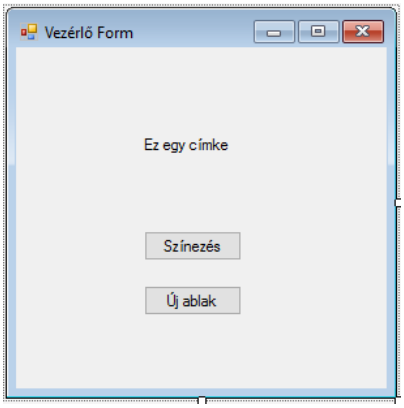
Miután kiválasztottuk, hogy Windows Form alkalmazást hozunk létre, megadtuk az alkalmazás nevét és helyét a gépünk mappastruktúrájában, megjelennek a fejlesztőfelület különböző ablakai, köztük a Solution Explorer és a Properties ablak, valamint a Form ablaka. A Solution Explorer ablakban az alkalmazás projektfájlai láthatók, jelen esetben a Form1.cs, és a hozzá tartozó kiegészítő fájlok, a Program.cs fájl, vagyis a Form és a Program osztályokat tartalmazó fájlok. Az app.config az alkalmazás paramétereit tartalmazó fájl.

A Properties ablakban megtekinthetjük és beállíthatjuk az éppen kijelölt vezérlő komponens tulajdonságait, s ha a villám ikonra kattintunk, akkor a Properties ablak eseményeket tartalmazó oldalát tekinthetjük meg. Az alábbi képen a Form Properties ablakának a tulajdonság és esemény oldalainak egy-egy részlete látható.



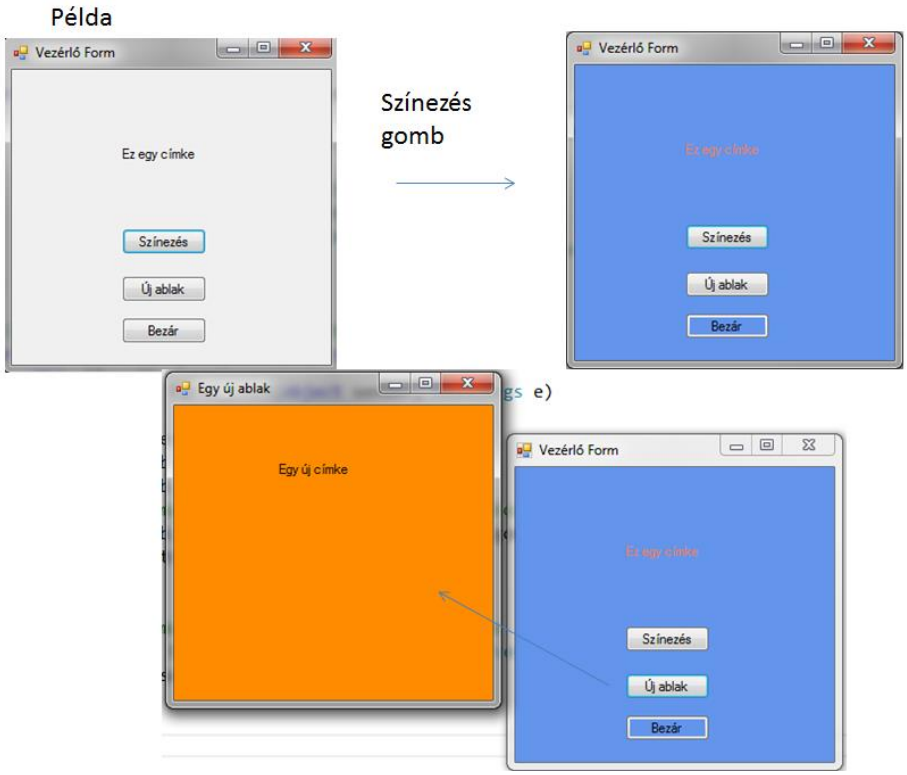
Tulajdonságok beállítása és objektum létrehozása futásidőben

A 3–5. táblázatban felsoroltuk a Form néhány publikus példánytulajdonságát, metódusát és eseményét. A következő mintapéldán azt mutatjuk meg, hogyan tudjuk ezeket a tulajdonságokat az alkalmazás futása közben megváltoztatni, beállítani.



A tervezéskor a képen látható vezérlőelemeket helyezzük el a Formra, két nyomógombot és egy label (címke) vezérlőt, az „Ez egy címke” tartalommal. Futásidőben változtatjuk meg az ablakon lévő címke betűszínét, módosítjuk az ablak helyét a képernyőn, majd megváltoztatjuk az ablak háttérszínét is. Az Új ablak feliratú nyomógombra kattintással egy új ablakot hozunk létre és egy címkét helyezünk rá, Új címke felirattal. A Form betöltésekor egy harmadik nyomógomb, Színezés is rákerül az ablakra, ennek segítségével változtatjuk meg az ablak színét.

Az alábbi képen a program futás közbeni állapotai látszanak.



A következő kép a fenti alkalmazás kódrészleteit mutatja.

```
static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main() {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```

A program osztály

Form1 példány létrehozása a konstruktor hívásával

A Form1 a Form osztály leszármazottja

Konstruktor

```
public partial class Form1 : Form
{
    public Form1() {
        InitializeComponent();
    }
```

```
private void InitializeComponent() {
    this.label1 = new System.Windows.Forms.Label();
    this.btnGomb = new System.Windows.Forms.Button();
    this.btnAblak = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(95, 68);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(72, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = "Ez egy címke";
```

A kódból az derül ki, hogy a Form1 osztály a Form osztályból származik, és Form1 egy parciális osztály, ami azt jelenti, hogy a kódnak egy része egy másik fájlban található. Ha megnézzük a konstruktort, azt látjuk, hogy egy *InitializeComponent()* nevű metódust hív

meg, a definícióját ennek a metódusnak azonban itt nem látjuk. Ezt a metódust a Visual Studio automatikusan generálja, a feladata pedig az, hogy beállítsa a vezérlők kezdeti tulajdonságait. Ha meg szeretnénk tekinteni a metódus tartalmát, akkor jobb egérgomb kattintás a metódushíváson és a felbukkanó menüből válasszuk ki a Go To Definition lehetőséget. A generált kód a Form1.Designer.cs fájlban található. Ha megnézzük a forráskódot, akkor látható, hogy ez a parciális osztályunk másik fele, s ebben találhatók a Form1 és a rajta tervezéskor elhelyezett vezérlőelemeinkkel kapcsolatos kódok.

Futás közben módosítsuk a vezérlőkomponensek tulajdonságait! Az alábbi képen a színezést végző gomb Click eseménykezelője látható. Az eseménykezelőben a címke ForeColor tulajdonságát módosítjuk, majd a form objektumot (this hivatkozás) CenterToSreen() metódushívással a képernyő közepére helyezzük. Ezután a form objektum háttérszínét is megváltoztatjuk, a BackColor tulajdonságának új értéket adva.

```
public partial class Form1 : Form
{
    public Form1() {

        InitializeComponent();
    }

    private void btnGomb_Click(object sender, EventArgs e) {

        label1.ForeColor = Color.Coral; //címke betűszín beállítás
        // ablak pozíciójának módosítása
        this.CenterToScreen();
        this.BackColor = Color.CornflowerBlue; //ablak háttérszínének módosítása

    }
```

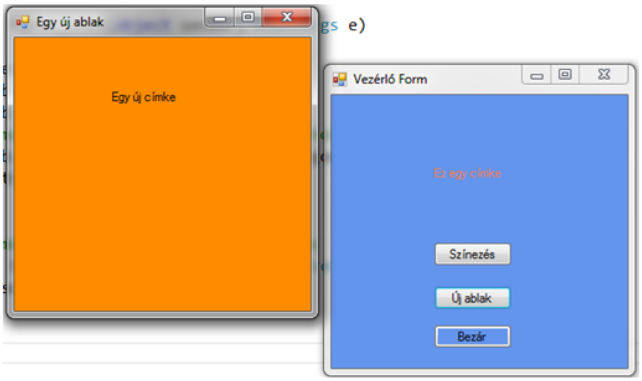
26

A következő kódrészletben az **Új ablak** nyomógomb eseménykezelőjének működése látható. Létrehozunk egy új Form objektumot, a Form osztály konstruktorának meghívásával, beállítjuk a háttérszínét (BackColor tulajdonság), megadjuk az ablak fejlécének, a címsornak feliratát (Text tulajdonság).

Ezt követően létrehozunk egy címke objektumot a *Label* osztály konstruktorával, megadjuk a címke feliratát a címke *Text* tulajdonságának értéket adva, majd beállítjuk a címke pozícióját az ablak bal felső sarkához viszonyítva. Ezt a beállítást a címke *Location* tulajdonságával végezzük el. A címke nevű *Label* objektumpéldány létrehozása és tulajdonságainak

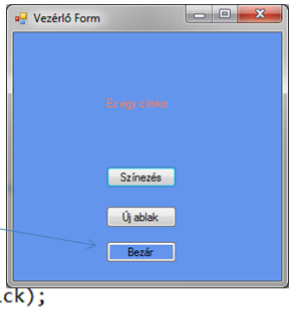
beállítása még nem jelenti azt, hogy készen vagyunk. Ahhoz, hogy a címkét láthassuk is a formon, hozzá kell adnunk a Form *Controls* gyűjteményéhez. Ehhez, a Controls gyűjtemény *Add*() metódusára lesz szükségünk. Az új form objektum megjelenítéséhez meg kell hívni a form objektum *Show()* metódusát.

```
private void btnAblak_Click(object sender, EventArgs e)
{
    // Új form létrehozása
    Form ujForm = new Form();
    ujForm.BackColor = Color.DarkOrange;
    ujForm.Text = "Egy új ablak";
    Label cimke = new Label();
    cimke.Text = "Egy új címke";
    cimke.Location = new Point(90, 50);

    ujForm.Controls.Add(cimke);

    ujForm.Show();
}
```



Láttuk, hogy tervezéskor csak két nyomógombot helyeztünk az alkalmazás ablakára. Amikor megjelenik a form a felhasználó előtt, szeretnénk, ha a harmadik nyomógomb is a felületen lenne, és használhatná a felhasználó. Ezt úgy oldhatjuk meg, hogy a form betöltésekor helyezzük fel a harmadik gombot a formra. Ehhez a form objektum *Load*() eseménykezelőjét kell elkészíteni. Ebben létrehozunk egy új nyomógomb objektumot, melynek neve esetünkben bezargomb. Ezután megadjuk a gomb objektum *Location* és *Text* (felirat) tulajdonságát. Ezután a gombra kattintás (Click) eseményét hozzá kell rendelni a gomb objektumhoz.

```
private void Form1_Load(object sender, EventArgs e)
{
    Button bezargomb = new Button();
    bezargomb.Text = "Bezár";
    bezargomb.Location = new Point(98, 220);
  // Gombnyomás esemény hozzárendelése a gombhoz
    bezargomb.Click += new EventHandler(bezargomb_Click);
    this.Controls.Add(bezargomb);
}

// A gombnyomás hatására végrehajtandó metódus
private void bezargomb_Click(object sender, EventArgs e) {
    this.Close();
}
```



Ezt követően a gomb objektumot hozzá kell adni a form *Controls* gyűjteményéhez, az *Add()* metódus segítségével.

A *Load* eseménykezelőn kívül még szükségünk van egy metódusra, mégpedig az előbb létrehozott bezárás gomb *Click* eseménykezelőjére. Ez egyetlen utasítást tartalmaz, az aktuális form objektum *Close*() metódusát hívja meg, amellyel bezárja a formot, s befejeződik a program végrehajtása.

1.1.5. A Form életciklusa

A Form alapértelmezett eseménye a *Load*, amely akkor fut le, mielőtt a Form először megjelenik. A Form betöltése és bezárása között számos esemény zajlik. Ezeket összefoglaló néven a Form életciklusának nevezzük. Ennek az állomásai a következők:

- *HandleCreated*: egy eseménykezelő jön létre a formhoz, ez gyakorlatilag egy hivatkozás.
- *Load*: a form megjelenése előtt fut le. Akkor következik be ez az esemény, amikor a form objektum Show() vagy ShowDialog() metódusát először meghívjuk. A Load esemény eseménykezelő metódusa a legalkalmasabb arra, hogy a form által használt erőforrásokat lefoglaljuk, illetve az adattagok, tulajdonságok kezdeti értékét beállítsuk, ahogy a korábbi példában láthattuk. Fontos tudni, hogy **ez az esemény a form életciklusában csak egyszer**, az első betöltődés során következik be.
- *Shown* esemény akkor következik be, amikor a form megjelenik.
- *Activated* esemény akkor következik be, amikor a formra kerül a fókusz. Ez az esemény a form életciklusában többször is előfordul. Ezt az eseményt a form objektum Show() vagy ShowDialog() metódusa idézi elő, valamint akkor következik be, amikor a Activate() metódust hívjuk meg.
- *Deactivated* akkor következik be, amikor az ablakon nincs rajta a fókusz, azaz elveszti a fókuszt. A fókuszt a felhasználói interakciók során veszítheti el a form, vagy amikor meghívjuk a form objektum Hide() vagy Close() metódusát. Fontos tudni, hogy az Activated és Deactivated események csak akkor következnek be, ha a futó programon belül változik meg a fókusz. (Más alkalmazásba pl. Word stb. átlépés ezt nem befolyásolja.)
- *VisibleChanged* ez az esemény akkor következik be, ha a form *Visible* tulajdonságának értéke megváltozik. Így ez az esemény minden alkalommal bekövetkezik, amikor a form láthatóvá vagy éppen rejtetté válik. A form Show(), ShowDialog(), Hide() és Close() metódusai idézik ezt elő.
- *Closing* esemény akkor következik be, amikor kezdeményeztük az ablak bezárását, de az még nem fejeződött be, folyamatban van. A form objektum Close() metódusának hívásával, vagy az ablak jobb felső sarkában található bezárás gombra kattintással idézzük elő az eseményt.
- *Closed* esemény akkor következik be, miután bezártuk az ablakot.
- *HandleDestroyes*: az ablakhoz tartozó kezelő megsemmisül.

A form Show() és ShowDialog() és Hide() metódusainak működéséről még néhány dolgot meg kell említeni. Ahhoz, hogy egy ablakot láthatóvá tegyük meg kell hívni a form Show()

metódusát, ahogy ezt már korábban említettük. Ez a metódushívás azt eredményezi, hogy egy form objektumpéldány töltődik be a memóriába, majd megjelenik a számítógép képernyőjén, és megkapja a fókuszt. A form objektum *Visible* tulajdonságának értékét a Show() metódus **true** (igaz) értékre állítja. Ha egy form objektum már betöltődött a memóriába, és pillanatnyilag nem látható (mert a *Visible* tulajdonság értéke **false** értékre lett beállítva) és újból meghívjuk a *Show*() metódusát, akkor a hatása ugyanaz lesz, mintha a *Visible* tulajdonság értékét **true** értékre módosítanánk.

A form objektum *ShowDialog*() metódusa ugyanúgy működik, mint a Show() metódus, azzal a *lényeges különbséggel*, hogy a form modális dialógus ablakként jelenik meg, ami azt jelenti, hogy a formot be kell zárni, mielőtt bármilyen más form megkapná a fókuszt. Tehát az objektumpéldány megszűnik az ablak bezárása után, míg egy nem modális ablak esetében a fókusz elvesztése nem szünteti meg az objektumpéldányt.

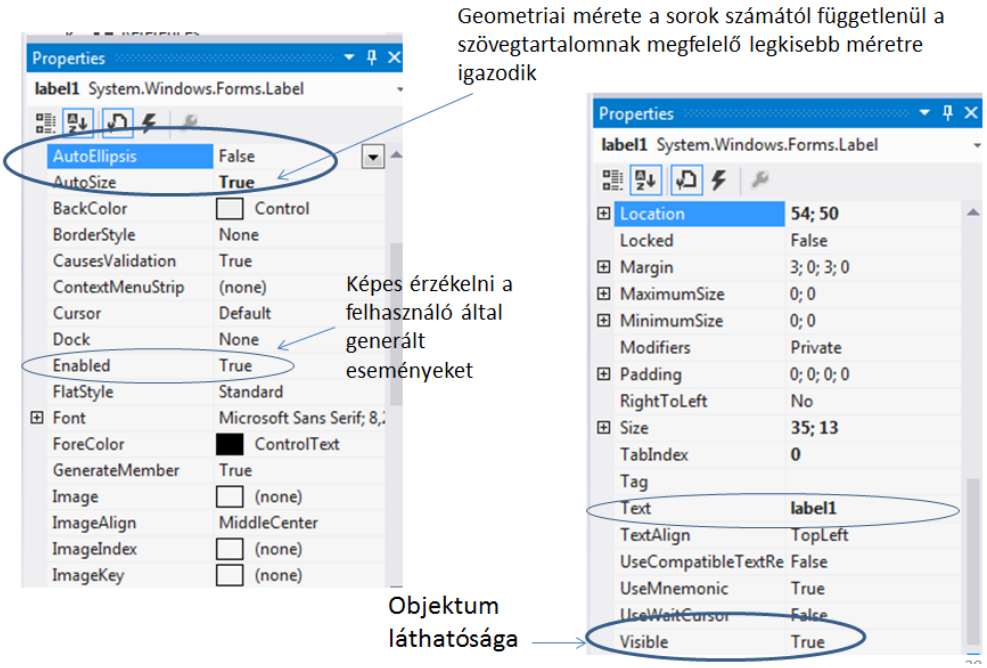
A *Hide*() metódus elrejti a formot, a képernyőn többé nem látható, azonban az objektum még létezik, benne van a memóriában. A *Show*() metódus hívásával, vagy a form objektum *Visible* tulajdonságának a kódban **true** értékre történő módosításával ismét megjelenik a form. A *Hide*() metódus hívásával a *Visible* tulajdonság értéke **false** értékre módosul.

1.1.6. Címkék, szövegdobozok használata

Label (Címke)

Feliratok elhelyezésére alkalmas vezérlőelem. Legfontosabb tulajdonsága a *Text*. Ez tartalmazza a megjelenítendő szöveget. A Text tulajdonságtartalma mind tervezési, mind futási időben változtatható kódszinten, de futás közben a felhasználó által interaktív módon semmiképp sem módosítható.

A Label vezérlő *nem interaktív* vezérlőelem, ezért **nem kerülhet rá a fókusz**. Az alábbi képen a Label Properties ablakának részleteit látjuk, néhány fontos tulajdonsággal.



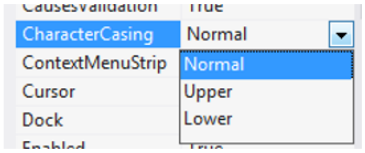
TextBox (Szövegdoboz)

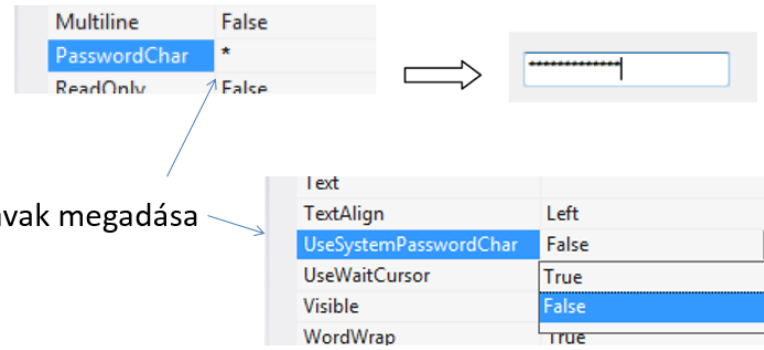
Szöveg megjelenítésére alkalmas vezérlőelem, input mezőként interaktív felhasználói kapcsolatra képes. A megjelenítendő szöveget vagy az input adatot a ***Text*** tulajdonsága tartalmazza. ***MultiLine*** tulajdonságának értékét *False* vagy *True* értékre állítva egy- vagy többsoros megjelenítési formáról dönthetünk. A ***ReadOnly*** tulajdonság *True*-ra állításával az adatbevitelt korlátozhatjuk, csak megjelenítésre lesz alkalmas a szövegdoboz.

A ***WordWrap*** tulajdonsága automatikusan megtöri egy szóköznél a szöveget, ha nem fér el a sorban. Csak *MultiLine=True* esetén működik. A ***ScrollBars*** tulajdonság igaz értékre állításával vízszintes vagy függőleges vagy mindkét irányú görgető sávot helyezhetünk el a szövegdobozban többsoros szövegek megtekintéséhez. *ScrollBars* tulajdonság is csak *MultiLine=true* esetén érvényes. A *CharacterCasing* tulajdonság a szövegdobozban megjelenő szöveg kisbetűssé vagy nagybetűssé alakítását teszi lehetővé, ahogy az alábbi ábrán látható.

Lehetőség van arra is, hogy a szövegdobozt jelszó bevitelére használjuk. Ehhez a ***PasswordChar*** tulajdonságértéket kell beállítani arra a kívánt karakterre, amit látni szeretnénk a jelszó gépelésekor a valódi jelszókarakterek helyett. Ha az operációs rendszer jelszó-helyettesítő karakterét kívánjuk megjeleníteni, akkor a ***UseSystemPasswordChar*** tulajdonságot ***True*** értékre kell beállítani.

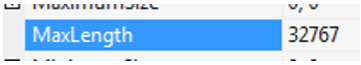
A *CharacterCasing* tulajdonsággal beállítható, hogy kisbetűvel (Lower), nagybetűvel (Upper), vagy ahogy beírta a felhasználó (Normal) jelenjen meg az információ.



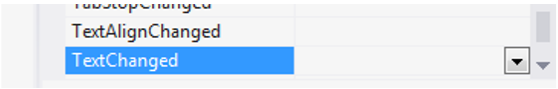


A megjeleníthető karaktersorozat hosszát is beállíthatjuk, amint az alábbi ábrán látható.

Megadható, hogy maximálisan hány karakter hosszú lehet a szövegdobozba írt karaktersorozat.



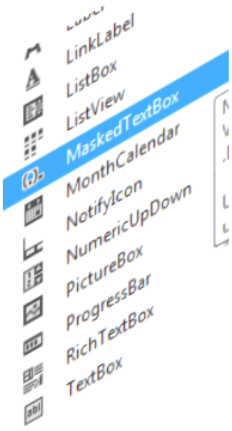
Alapértelmezett **esemény** a *TextChanged*.



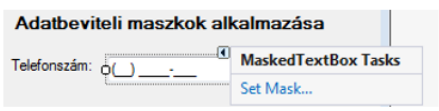
A szövegdoboz legfontosabb eseménye a TextChanged esemény, melynek kezelésével követni tudjuk a tartalom változását és reagálni tudunk rá.

Adatbevitel maszkolása (MaskedTextbox)

A MaskedTextbox módosított változata a szövegdobozoknak. Segítségükkel előre meghatározott formátumnak megfelelően adhatjuk meg az adatokat a program számára. A Mask tulajdonságértéke tartalmazza, hogy milyen karaktereket, és azokat opcionálisan vagy kötelező jelleggel kell-e megadni. Az alábbi ábrán látható táblázatban adtuk meg a maszkolásra használható karaktereket és leírásukat.



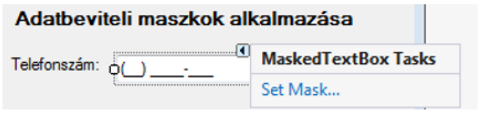
Maszk karakter	Leírás
0	0 és 9 közötti számjegy karaktert képvisel az adott pozíción.
9	0 és 9 közötti számjegy karaktert reprezentál. A karakter megadása az adott pozíción nem kötelező.(opcionális)
#	0 és 9 közötti számjegy vagy szóköz opcionális karaktert reprezentál. A plusz (+) és a (-) jel szintén megengedett.
&	Az adott pozíción kötelezően megadandó karaktert képvisel. Ha az *ASCIIOnly* tulajdonság értéke *True*, akkor úgy viselkedik mint a L eset.
:	Idő elválasztó számára fenntartott hely. A *FormatProvider* vezérlőben megadott idő elválasztó karakter jelenik meg a pozíción.
/	Dátum elválasztó számára fenntartott karakter pozíció. A *FormatProvider* vezérlőben megadott idő elválasztó karakter jelenik meg a pozíción.
\	Escape mask karakter. A \ karaktert követő karaktert literál karakterré alakítja. Ha magát a \ karaktert akarjuk megjeleníteni az adott pozíción, akkor duplán (\\) kell megadni.



Gyakran használunk adatbeviteli maszkot telefonszámok, dátumok megadásához. Az adatbevitel ily módon történő megvalósítása adatbevitel-ellenőrzést is lehetővé tesz.

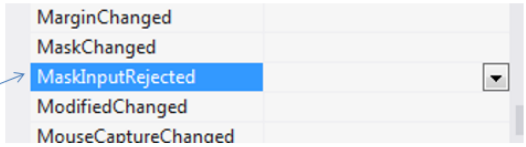
Adatbevitel ellenőrzése

Az ábrán látható, hogy nem megfelelő formátumú adatbevitel esetén bekövetkezik a ***MaskInputRejected*** esemény, amelynek, ha elkészítjük az eseménykezelőjét, reagálni tudunk a hibás adatbevitelre. Ha beállítjuk a *MaskedTextbox* ***BeepOnError*** tulajdonságát a **true** értékre, akkor hiba esetén hangjelzést is kapunk.

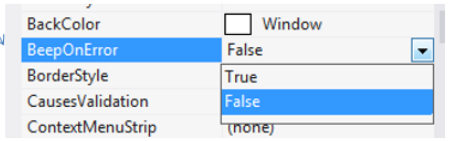


Alapértelmezett esemény

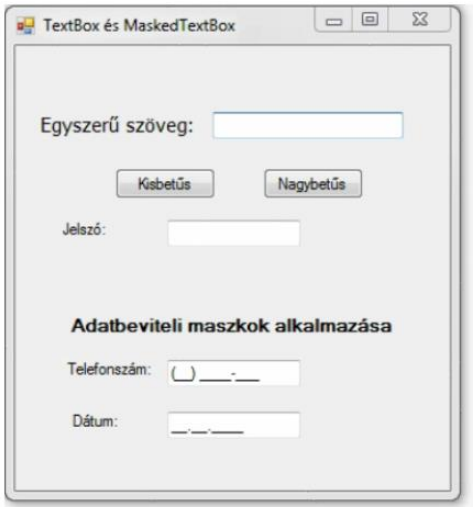
A maszk által nem engedélyezett karakterek bevitele során a vezérlőelem a MaskInputRejected eseményt váltja ki.



Ha a MaskedTextBox vezérlőelem BeepOnError tulajdonsága a True értékre van állítva, a program a System.Beep metódust is meghívja.

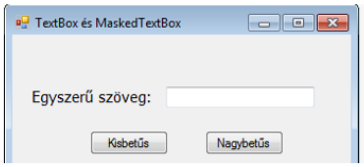


A következő mintaprogramban az eddig említett tulajdonságok használatát láthatjuk. Létrehozzuk a képen látható formot, amelyre két nyomógombot helyezünk el a szövegdoboz tartalmának kis- vagy nagybetűssé alakítására. A programban a formázott adatbevitelre is látunk példát a telefonszám és a dátum mező értékének megadásakor.

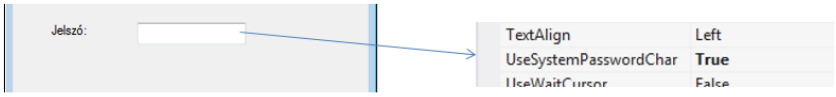


A képen a két nyomógomb Click eseménykezelőjének kódja látható. Az eseménykezelők ugyanazt az utasítást tartalmazzák, azzal a különbséggel, hogy egyik esetben a

CharacterCasing tulajdonságnak a Lower (kisbetűs), másik esetben az Upper (nagybetűs) értéket adják.



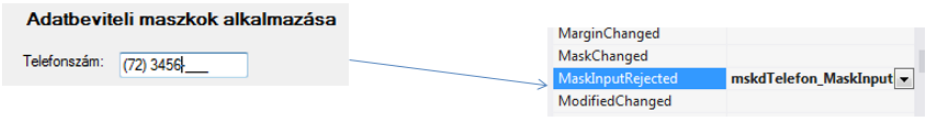

```
private void btnKisbetus_Click(object sender, EventArgs e) {
    txtSzoveg.CharacterCasing = CharacterCasing.Lower;
}

private void btnNagybetus_Click(object sender, EventArgs e) {
    txtSzoveg.CharacterCasing = CharacterCasing.Upper;
}
```




```
private void txtJelszo_TextChanged(object sender, EventArgs e) {
    lbJelszo.Font = new Font("Tahoma", 11);
    lbJelszo.Text = txtJelszo.Text;

}
```

A fenti ábrarészen a szövegdoboz jelszóbeviteli mezőként való alkalmazását látjuk, mégpedig a képen látható beállítással az operációs rendszer jelszókarakterét kívánjuk használni. Ezt követően a szövegdoboz TextChanged eseményére reagálva, ami mindig bekövetkezik, amikor változik a szövegdoboz tartalma, elkészítettünk egy eseménykezelőt, amelyben, egy címkében (lbJelszo) a szövegdoboz alá kiírjuk a begépelt jelszót.

A következő kódrészlet a formázott telefonszámmező kezelését mutatja be. Amikor hibásan adjuk meg a telefonszám valamelyik karakterét, bekövetkezik a ***MaskInputRejected*** esemény.




```
private void mskdTelefon_MaskInputRejected(object sender, MaskInputRejectedEventArgs e) {

    // A szöveget pirosra színezzük, jelezve a maszk megsértését
    ((MaskedTextBox)sender).ForeColor = Color.Red;
}
```

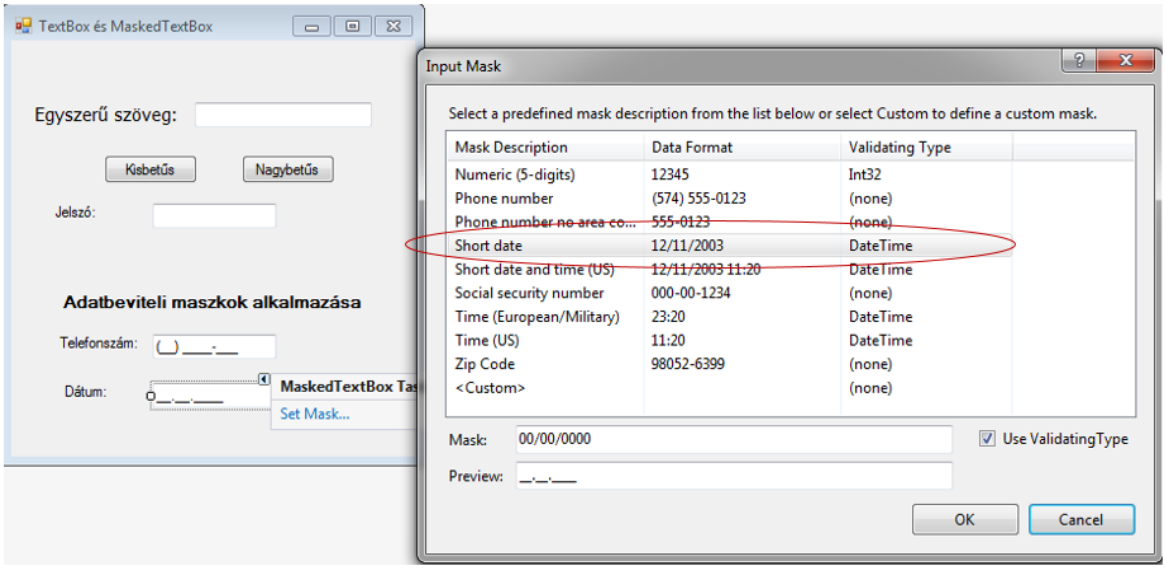

```
private void mskdTelefon_KeyDown(object sender, KeyEventArgs e) {
    // A szöveget visszaállítjuk az eredeti színre
    ((MaskedTextBox)sender).ForeColor = Color.FromKnownColor(KnownColor.WindowText);
    // vagy

    ((MaskedTextBox)sender).ForeColor = Color.FromName("WindowText");
```

Erre az eseményre készítettünk egy eseménykezelőt, amelyben piros színűre változtattuk az eseményt küldő (sender) objektumának, ami a *MaskedTextBox*, betűszínét. Készítettünk egy másik eseménykezelőt is arra az esetre, amikor lenyomja a felhasználó a billentyűt a hiba észlelése után, hogy kijavítsa azt, akkor visszaállíthassuk a beviteli mező értékét az eredeti alapszínre.

Van egy másik fontos tulajdonsága is a *MaskedTextBox* vezérlőelemnek, a ***RejectInputOnFirstFailure*** tulajdonság. Ha ezt a tulajdonságot **true** értékre állítjuk be, akkor hibás karakter bevitelekor az egész adatbevitelt visszautasítja a vezérlő. Ha ennek a tulajdonságnak az értéke **false**, akkor az összes olyan karaktert elfogadja, amely megfelel az előre megadott formátumnak.

Dátummaszkos mező

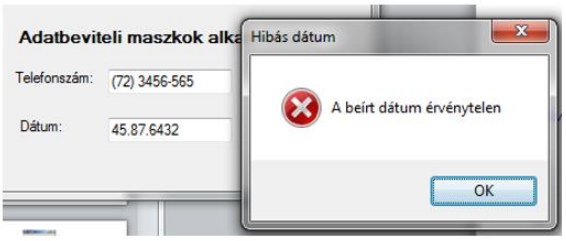
Az alábbi képen arra látunk példát, hogy előre elkészített mintákat is alkalmazhatunk adatbeviteli mező készítésekor. A MaskedTextBox vezérlőelem jobb felső sarkában látható úgynevezett smart tagre kattintva, majd a **Set Mask** menüt kiválasztva, közvetlenül az **Input Mask** ablakba jutunk, ahol előre elkészített beviteli formák közül választhatunk, vagy módosíthatjuk őket.



A következő kódrészletben arra látunk példát, hogy formázott adatbevitel esetén, ha elkészítjük a **TypeValidationCompleted** eseménykezelőt, akkor a kódban az adatbevitelt követően ellenőrizhetjük, hogy érvényes értéket vittünk-e be az adatmezőbe vagy sem. Érvénytelen input esetén egy felbukkanó ablakban értesítjük a felhasználót a hibáról. Ennek a részletei láthatók a következő ábrán.

Esemény


```
private void mskdDatum_TypeValidationCompleted(object sender, TypeValidationEventArgs e) {
    if (!e.IsValidInput)
    {
        MessageBox.Show("A beírt dátum érvénytelen", "Hibás dátum", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```



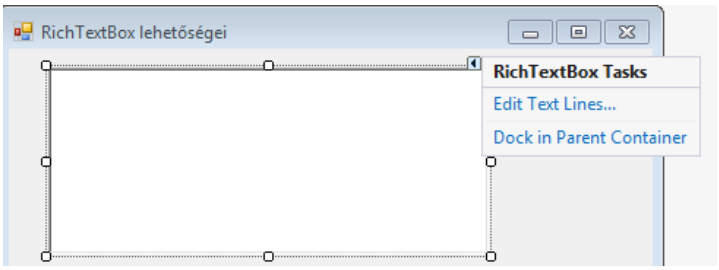
Rich TextBox vezérlő

Alapvető szövegszerkesztési lehetőségeket kínál. Számos metódusa biztosítja, hogy megfelelően változtassuk a tartalmát, illetve a tartalom formáját.

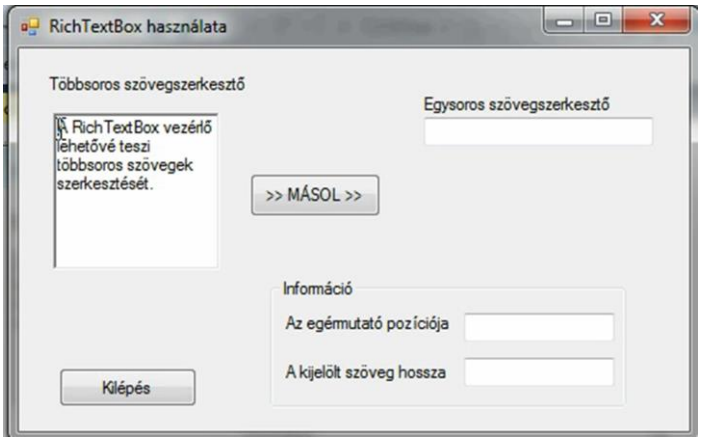
A RichtextBox legfontosabb tulajdonságait, metódusait az alábbi táblázatban soroltuk fel. A tulajdonságok és metódusok neve alapján könnyű következtetni arra, hogy mire szolgálnak, így ezeket most nem részletezzük, néhány példán mutatjuk be használatukat.

Tulajdonság	**Metódus**
Text	SaveFile()
TextLength	LoadFile()
SelectedRtf	Select()
SelectedText	SelectAll()
SelectionLength	Find()
SelectionBackColor	Clear()
SelectionType	Cut()
SelectionColor	Copy()
SelectionFont	Undo()
Lines	Redo()

Az alábbi képen látható, hogy tervezési időben a Smart tag elem segítségével a Text tulajdonságértékeként adhatjuk meg a szövegtartalmat, amelynek kinézetét futási időben tetszés szerint módosíthatjuk.



A következő programban bemutatjuk a Rich TextBox néhány hasznos tulajdonságát. A program lehetővé teszi, hogy a rich TextBoxba beírt szöveg egy részletét kijelöljük és azt a jobb oldalon látható szövegdobozba másoljuk. Közben információt kapunk arról, hogy mekkora a kijelölt szövegrész hossza és hol áll az egérmutató, vagyis hol kezdődik a kijelölés.



A Másol gomb eseménykezelőjében a kijelölt szöveget a rich TextBox ***SelectedText*** tulajdonságából kaptuk meg. A kijelölt szöveg hosszát a ***SelectionLengh*** tulajdonság adja. A ***SelectionStart*** tulajdonság a kijelölés kezdő pozícióját adja meg.

```
1 reference
private void Masol_Click(object sender, EventArgs e)
{
    SzovegPozicio = 0;
    KijeloltSzoveg = richTextBox1.SelectedText;
    SzovegHossza = richTextBox1.SelectionLength;
    if (SzovegHossza != 0)
    {
        SzovegPozicio = richTextBox1.SelectionStart;
    }
    txtMasolat.Text = KijeloltSzoveg;
    txtMasolKezdPoz.Text = SzovegPozicio.ToString();
    txtHossz.Text = SzovegHossza.ToString();
}
```

1.1.7. Fókusz és körbejárási sorrend

A program futása során az éppen aktív ablakban *egyszerre csak egy control* lehet abban a helyzetben, hogy a *felhasználói beavatkozást érzékelje*. Vagyis egy adott pillanatban csak akkor képes egy objektum a felhasználói interakciókat fogadni, ha ő a *fókusz* birtokosa.

Például, ha a fókusz egy TextBoxon van, akkor képes az *eseménykészletébe tartozó* minden környezeti változás (kattintás, szövegmódosítás, egérkurzor-mozgatás stb.), de csakis azok érzékelésére.

Egy objektum a következőképpen szerezheti meg a fókuszt:

- Az egérrel kiválasztjuk.
- A Tab billentyű lenyomásával választjuk ki.
- Egy billentyűkombináció segítségével választjuk ki.
- A forráskódban *a Focus()* metódus használatával tesszük a fókusz birtokosává.

Az objektumok csak az *Enabled* és *Visible* tulajdonságaik **True** értéke esetén képesek a fókusz fogadására. Néhány control *nem fogadhat* fókuszt, pl. Label, Panel, GroupBox.

Fontos tudni, hogy a Form is *csak akkor fogadhatja a fókuszt*, ha **üres**, illetve csak olyan vezérlőelemek vannak rajta, amelyek egyébként nem lehetnek a fókusz birtokosai.

A fókuszt nemcsak megszerezni, hanem elveszíteni is lehet, hiszen az alkalmazás futása során a felhasználói felület más-más része lesz aktív.

Két eseményt használhatunk a fókusz objektumok közötti vándorlásának követésére:

- **Enter**, amely a fókusz vételekor következik be.
- **Leave,** amely a fókusz elvesztésekor következik be.

Körbejárási sorrend

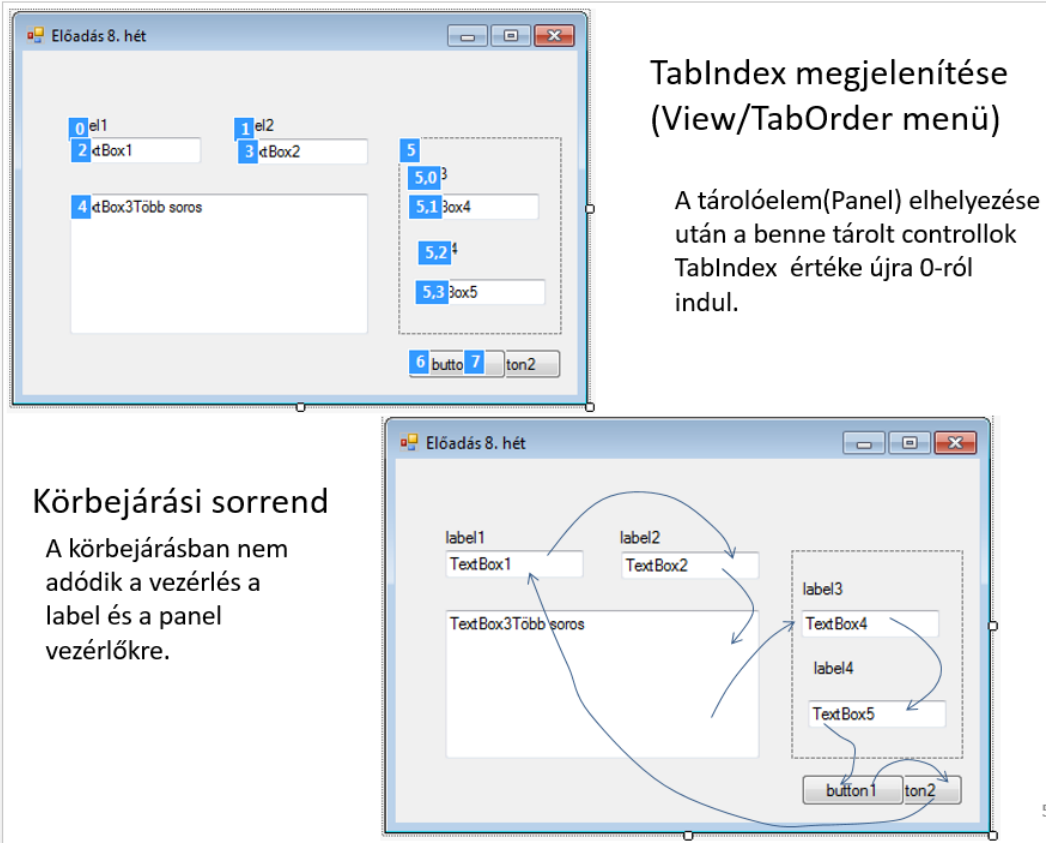
Alapértelmezés szerint a vezérlőelemek körbejárási sorrendje megegyezik a felületre helyezésük sorrendjével, feltéve, ha a Tab billentyű lenyomásával haladunk a felületen.

Tehát az alkalmazás indításakor az a control (vezérlőelem) kapja meg a fókuszt, amelyet először a felületre helyeztünk.

Minden control rendelkezik egy sorszámmal, amelynek értéke eredetileg megegyezik az adott vezérlőelem Formon való elhelyezésének sorrendjével.

Ez a sorszám az adott control ***TabIndex*** tulajdonságában tárolódik el, és onnan kiolvasható. A kezdősorszám a nulla. A következő ábrán a felületre elhelyezett vezérlőelemek TabIndex értékét láthatjuk. Fontos megjegyezni, hogyha ráteszünk a formra egy konténer típusú vezérlőt (konténer típusú vezérlőelemekre más vezérlőelemek elhelyezhetők), akkor a Tabindex számozása a konténeren belül újra nulláról indul. Az ábrán az is látható, hogy a

körbejárás a TabIndex értékének megfelelő sorrendben történik, de a bejárás nem érinti a Label és a Panel (konténer típusú) vezérlőelemeket, mert ezek nem fogadhatnak fókuszt.



Mind a tervezés, mind a kódírás során megváltoztathatjuk a TabIndex értékét. Arra is lehetőség van, hogy két vagy több control TabIndex értékét azonosra állítsuk, ha egymás fölötti rétegekben helyezzük el őket.

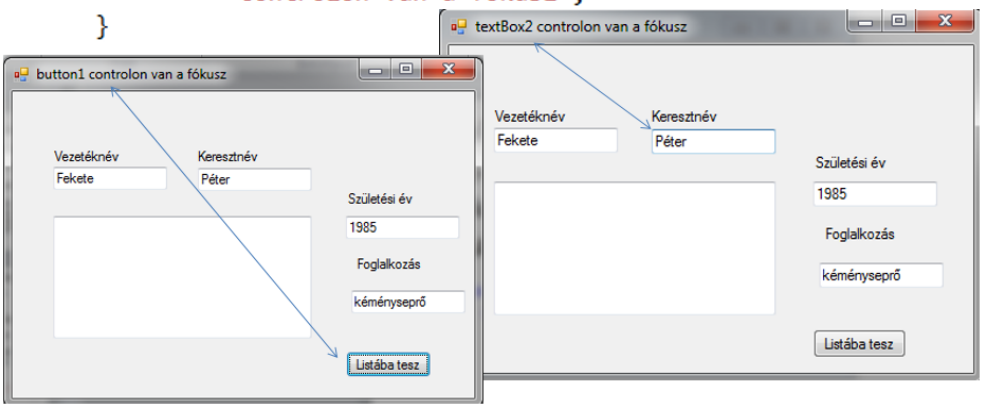
Eltérő TabIndex értékű vezérlők körbejárási sorrendje vízszintes irányú, míg azonos TabIndex értékű vezérlők esetén a körbejárási sorrend függőleges irányúvá válik. (Ekkor a vezérlők egymásra helyezési sorrendje dönti el, hogy melyik mikor szerzi meg a fókuszt).

A programunkban is módosíthatjuk a fókusz áthelyezését egyik vezérlőelemről a másikra, egyrészt a TabIndex értékének változtatásával, vagy az adott vezérlő Focus() metódusának hívásával (vízszintes irányban), vagy a BringToFront() metódusával (függőleges irányban). Egy vezérlőt ki is vehetünk a körbejárási sorrendből, amennyiben a TabStop tulajdonságának az értékét **False**-ra állítjuk.

A következő kódrészletben követjük a fókusz mindenkori helyzetét, s ezt a Form1 Click eseményéhez kötjük. A fókusz helyzetét az ablak címsorába fogjuk kiírni.

Ahova kattintunk, az a vezérlőelem objektum válik a fókusz birtokosává, vagyis a Form ActiveControl tulajdonsága tartalmazza azt a vezérlőt, amelyen éppen a fókusz van, s a Name tulajdonsággal a vezérlőelem nevét is megkapjuk.

```
private void Form1_Click(object sender, EventArgs e) {
    this.Text = this.ActiveControl.Name +
        " controlon van a fókusz";
}
```



1.1.8. Kivételkezelés

Az előző fejezetekben láttuk, hogy a szövegdobozok és a rich TextBoxok is arra szolgálnak, hogy a program működéséhez szükséges adatokat meg tudjuk adni. Mivel a felhasználó ott, ahol számot várunk, szám helyett karaktereket adhat meg vagy üresen hagyhatja a Textboxot, gondoskodnunk kell arról, hogy nem megfelelő adatbevitel esetén megfelelő üzenettel tájékoztassuk őt a hibáról.

A program futása közben fellépő hibákat kivételkezelés (Exception-handling) segítségével felügyelhetjük. Ha a programunkat nem készítjük fel a hibás esetek kezelésére, akkor a .NET alapértelmezett mechanizmusa érvényesül, vagyis megszakad a program végrehajtása.

A fent említett, a szövegdobozba történő helytelen adat beírása esetén hibaüzenetet kapunk, mégpedig egy **FormatException** kivétel dobódik. Ha erre felkészítjük a programunkat, akkor ezt a kivételt elkaphatjuk, és megfelelő tájékoztatást küldhetünk a felhasználónak. A hibák elkapására a try-catch szerkezetet használhatjuk. A szerkezet try ágába kerülnek a normális programműködés esetén végrehajtandó utasítások, míg a catch ágában a hibás működés miatt keletkezett kivételeket tudjuk kezelni.

```
try   // utasítások, amelyek normális működés során hajtódnak végre
{
            Ha hiba, throw valami    // Kivételt dob.
}
catch (valami)          // Elkapja a kivételt
{
            Megold, vagy hibát üzen.
 }
```

A kivételek is objektumok, minden kivételhez saját kivételosztály tartozik.

Az alábbi táblázatban soroltuk fel a System névtér néhány kivételosztályát.

6. táblázat: Az Exception (kivétel) osztályok

Kivételosztály	**Leírás**
Exception	A kivételek ősosztálya
FormatException	Szöveg számmá alakításakor fordulhat elő
ArithmeticException	Hiba az aritmetikai kifejezésben, pl. túlcsordulás stb.
OutOfMemoryException	Nincs elég memória egy objektum létrehozásához
NullReferenceException	Az objektumreferencia helyett null értéket talál a futtató rendszer.
System.IO.IOException	Valamilyen input/output hiba keletkezett
DevideByZeroException	0-val való osztás kísérlete
IndexOutOfRangeException	Tömb indexhatárainak túllépése

A **try**, **catch**, **finally**, valamint a **throw** kulcsszavakat használhatjuk a hibák kezelésére. Egy **try** blokkhoz több **catch** blokk is tartozhat. A catch blokkokat megfelelő sorrendben kell elhelyezni. A specifikusabb kivételtípusokat kell először megadni, mert a fordító hibaüzenetet ad, ha egy általánosabb catch blokk megelőz egy specifikusabb blokkot.

Az aktuális catch blokk végrehajtása után az utolsó catch blokkot követő utasítással folytatódik a program végrehajtása.

Ha a keletkezett kivételt egyik catch blokk sem azonosítja, akkor az a futtatórendszerhez továbbítódik és a program futása hibaüzenettel megszakad. A kivételek ősosztályai minden belőlük származtatott kivételt azonosítanak. A catch blokkok sorozatát egy paraméter nélküli catch zárhatja, mellyel minden kivételt elfoghatunk.

Összefoglalva: Ha a try blokk utasításainak végrehajtása során **nem dobódott kivétel**, a try-catch blokk végrehajtása **normálisan befejeződik**. Ha valamilyen hiba miatt **kivétel dobódott**, sorban megpróbáljuk **illeszteni** a keletkezett kivétel objektumot **a catch blokkok fejében szereplő kivétel osztályokra**. Ha **találunk illeszkedést**, az adott **catch blokkot végrehajtjuk**, és a try-catch blokk normálisan befejeződik, illetve ha a catch blokkban újabb kivétel dobódik, kivétellel fejeződik be. Ha **nincs illeszkedés**, a try-catch blokk **kivétellel**

fejeződik be és a program működése hibaüzenettel megszakad. Ezt elkerülendő célszerű egy paraméter nélküli catch blokkal az összes létező hibát elkapni.

Finally blokk: A try blokk futásának eredményétől függetlenül, akár elértük a blokk végét, akár kivételt dobtunk, a *finally* blokkot mindig végrehajtjuk. Ha egy try utasításban szerepel a dobott kivételnek megfelelő catch blokk is, akkor annak a finally blokk előtt történik meg a végrehajtása. A *Finally* blokkal biztosítható, hogy bizonyos kódrészek mindig lefutnak, akkor is, ha kivételt kapunk (pl. a fájl kapcsolat lezárása).

A ***throw*** utasítást kivételes helyzetek programból való előállítására használhatjuk. Mi magunk is dobhatunk hibát, ha szükségesnek tartjuk, amit aztán később egy catch blokkban el is kapunk. Formája:

```
throw new Exception();
```

Saját kivételosztályokat is származtathatunk:

```
class Sajátkivétel: System.Exception {}
…
…
throw new Sajatkivetel();
```

Egy példa saját kivétel dobására. Azt ellenőrizzük, hogy a honap nevű változó értéke érvényes hónap sorszámot tartalmaz-e vagy sem. Ha az érték a tartományon kívül esik, kivételt dobunk, amit később elkapunk.

```
if ((honap <=0) || (honap>12))

      throw new SytemException("Nincs ilyen hónap");
```

Mindegyik kivételnek van *Message* tulajdonsága, amit a kivétel kezelésekor kiírathatunk.

A try blokkok egymásba ágyazhatók. A belső által dobott kivételt a külső is elkaphatja. A try blokkból hívott függvények is dobhatnak kivételt. Egy függvénynél mindig felsorolja a súgó a dobott kivétel típusát. Ezt kapjuk el, és kezeljük az osztály által szolgáltatott lehetőségek kihasználásával.

A következő kódrészleten a try-catch szerkezetre látunk példát. Egy listába szeretnénk beszúrni egy új nevet egy megadott pozícióba. A program azt ellenőrzi, hogy a megadott pozíció értéke nem nagyobb-e mint a lista elemeinek a száma, vagy nem kisebb-e mint nulla. Ha igen, akkor saját kivételt dobunk, az **ArgumentOutOfRangeException**-t, amit el is kapunk a catch ágban. A kivétel üzenet tulajdonságának értékét is kiíratjuk.

A program azt is ellenőrzi, hogy a felhasználó az adott szövegdobozba számértéket írt-e. Ezt úgy ellenőrizzük, hogy elkapjuk az esetleges **FormatException** kivételt, amit a rendszer futáskor dob.

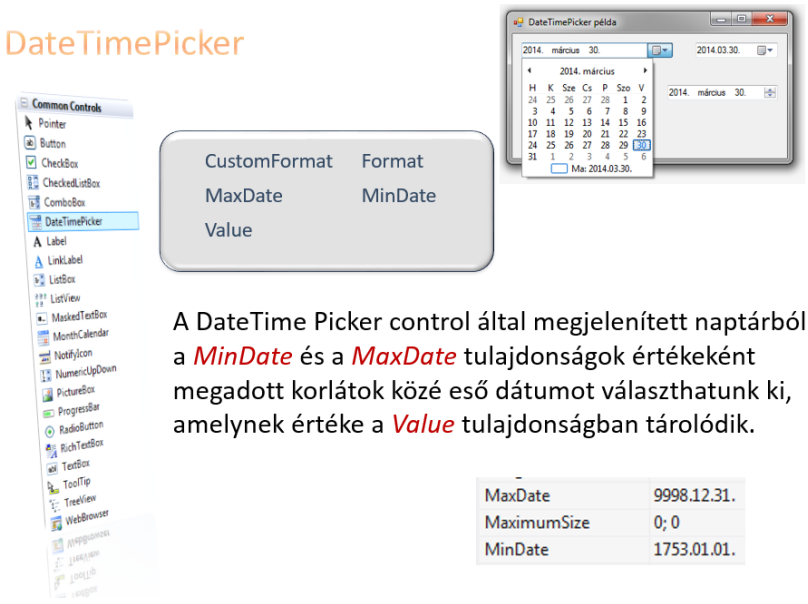
```
private void btnBeszur_Click(object sender, EventArgs e)
{
    int hova;
    if (txtNev.Text == ""){
        MessageBox.Show("Nem adott meg nevet", "Hiba");
        txtNev.Focus();
    }
    else if (txtHova.Text == ""){
        MessageBox.Show("Nem adta meg hova!", "Hiba");
        txtHova.Focus();
    }
    else {
        try {
            hova = int.Parse(txtHova.Text);
            if (hova >= 0 && hova < lstNevsor.Items.Count){
                lstNevsor.Items.Insert(hova - 1, txtNev.Text);
                txtNev.Text="";
                txtHova.Text="";
                txtInfo.Text=lstNevsor.Items.Count.ToString();
                txtNev.Focus();
            }
            else {
                throw  new ArgumentOutOfRangeException("Nincs a tartományban");
            }
          }
        catch (FormatException){
            MessageBox.Show("Csak egész szám lehet");
        }
        catch(ArgumentOutOfRangeException ex)
        {
          MessageBox.Show(ex.Message);
        }
      }
    }
```

1.1.9. Dátum és idő kezelését támogató vezérlők

A programok készítése során gyakran van szükség arra, hogy dátumértékeket adjunk meg. A grafikus felületeken a dátumok megadását külön vezérlők támogatják. A továbbiakban ezeket a vezérlőelemeket tekintjük át.

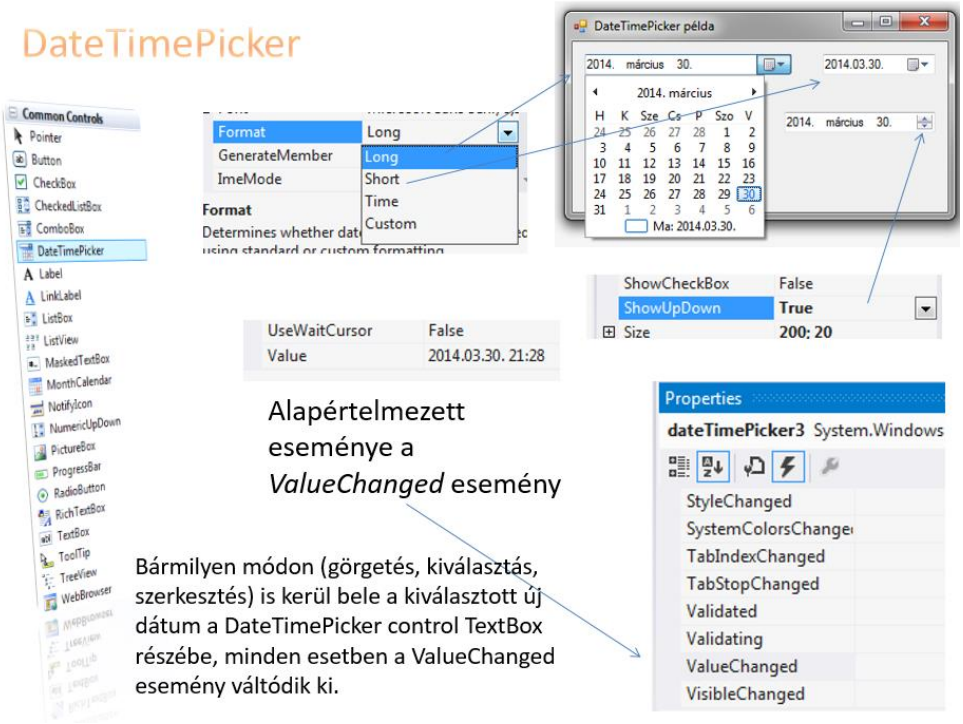
DateTime Picker

A **DateTime Picker** vezérlőben a dátumot és az időt egy naptárnézetben állíthatjuk be, illetve tekinthetjük meg. Egyszerűen a kinyíló naptárban egy kattintással állíthatjuk be a kívánt értéket. A dátum-idő kiválasztóban kijelölhető dátumot és időt a **MinDate** és a **MaxDate** tulajdonságokkal szabályozhatjuk. A **Format** tulajdonság segítségével adható meg, hogy milyen formában jelenjen meg a dátum és idő a vezérlő szövegdobozában. Az úgynevezett rövid és hosszú dátum között választhatunk, ami abban különbözik egymástól, hogy a hónap sorszáma (short érték) vagy a neve (Long érték) jelenjen meg a mezőben. Választhatjuk azt is, hogy csak az idő jelenjen meg, dátum nélkül (Time érték). A Custom tulajdonságérték választásakor mi adhatjuk meg a formát. Az alábbi képen a DateTime Picker legfontosabb tulajdonságai láthatók.



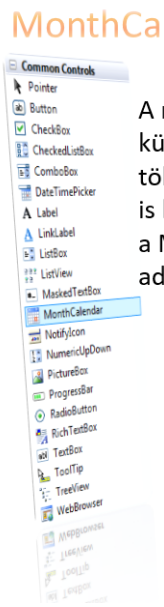
Fontos tudnunk, hogy a kiválasztott dátumérték a vezérlő Value tulajdonságában van, innét kell kivennünk, ha szükségünk van rá. Amikor a felhasználó kiválaszt egy dátumot, akkor a **ValueChanged** esemény történik. A Value tulajdonságban tárolt érték egy DateTime típusú objektum. A DateTime típusokra hamarosan kitérünk.

Az alábbi képen a Format tulajdonság különböző értékeinek választásától függő dátum megjelenési formák láthatók. Ha a dátum-idő értékek igazításához függőleges vezérlőt szeretnénk, állítsuk a ShowUpDown tulajdonságot true-ra. A képen a vezérlő Properties ablakából a DateTimePicker néhány eseménye is látható.

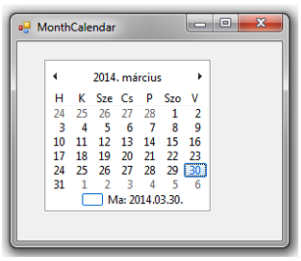


A MonthCalendar (Naptár) vezérlő

A naptárban a felhasználó kattintással választhatja ki a dátumot és időt. A kiválasztható értéktartományt a **MinDate** és **MaxDate** tulajdonságok határozzák meg. Amikor a felhasználó egy új dátumot választ ki, egy **DataSelected**, amikor pedig a dátum megváltozik, egy **DataChanged** esemény jön létre.



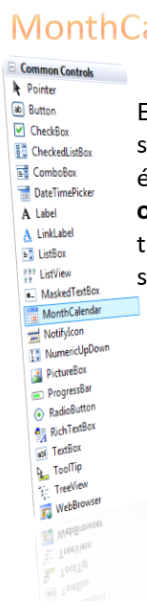
A megjelenésen kívül abban is különbözik a DateTimePicker-től, hogy több, egymást követő nap kijelölésére is lehetőség van. A kijelölések számát a MaxSelectionCount tulajdonságban adhatjuk meg.



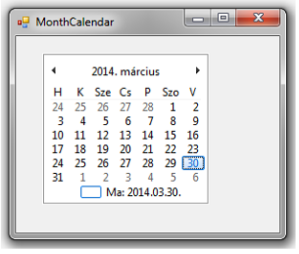
A dátumot a *Value* tulajdonság tartalmazza.

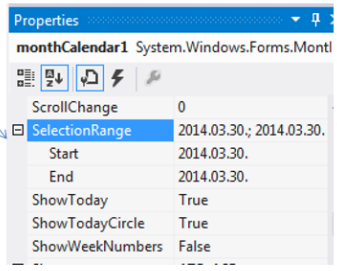
BoldedDates	FirstDayOfWeek
MaxDate	MinDate
MaxSelectionCount	SelectionEnd
SelectionRange	SelectionStart

A MonthCalendar vezérlő dátumtartomány kijelölését is lehetővé teszi. A kijelölt napok számát a **MaxSelectionCount** tulajdonság adja meg. A SelectionRange tulajdonság egy SelectionRange objektumot ad vissza, amelynek két hasznos tulajdonsága van, a Start és az End, amelyek a kiválasztott dátum elejének és végének megfelelő DateTime objektumot adnak vissza.



Egy kiválasztott időtartam napjainak számát a SelectionRange tulajdonság értékeként kapott **SelectionRange objektum** *Start* és *End* tulajdonságértékek különbsége szolgáltatja.



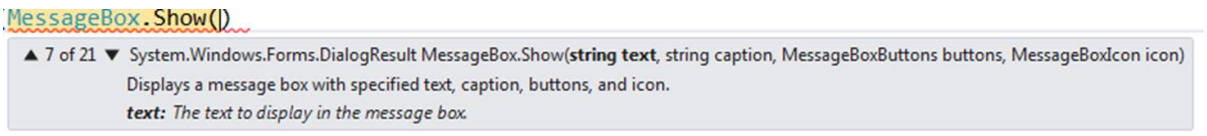


Új dátum kiválasztásakor a *DateChanged* esmény generálódik, egy tartomány kijelölésekor pedig a *DateSelected* esemény váltódik ki.

1.1.10. MessageBox dialógusablak

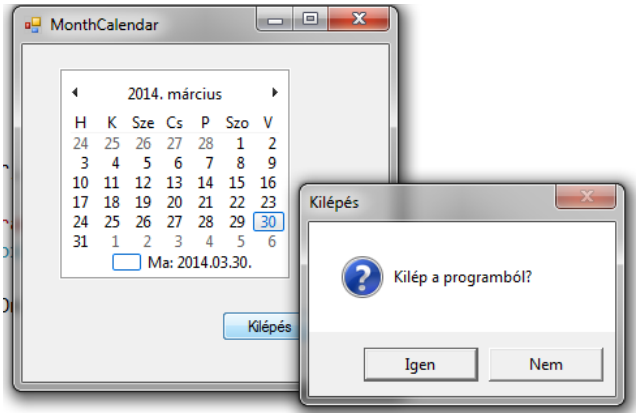
A MessageBox ablakot arra használjuk, hogy értesítéseket küldjünk a felhasználóknak bizonyos események megtörténtekor.

A MessageBox osztály többszörösen túlterhelt Show() metódusát alkalmazzuk az üzenetek megjelenítésére. Például hibák jelzésére, vagy valamilyen tevékenység megerősítésére stb.



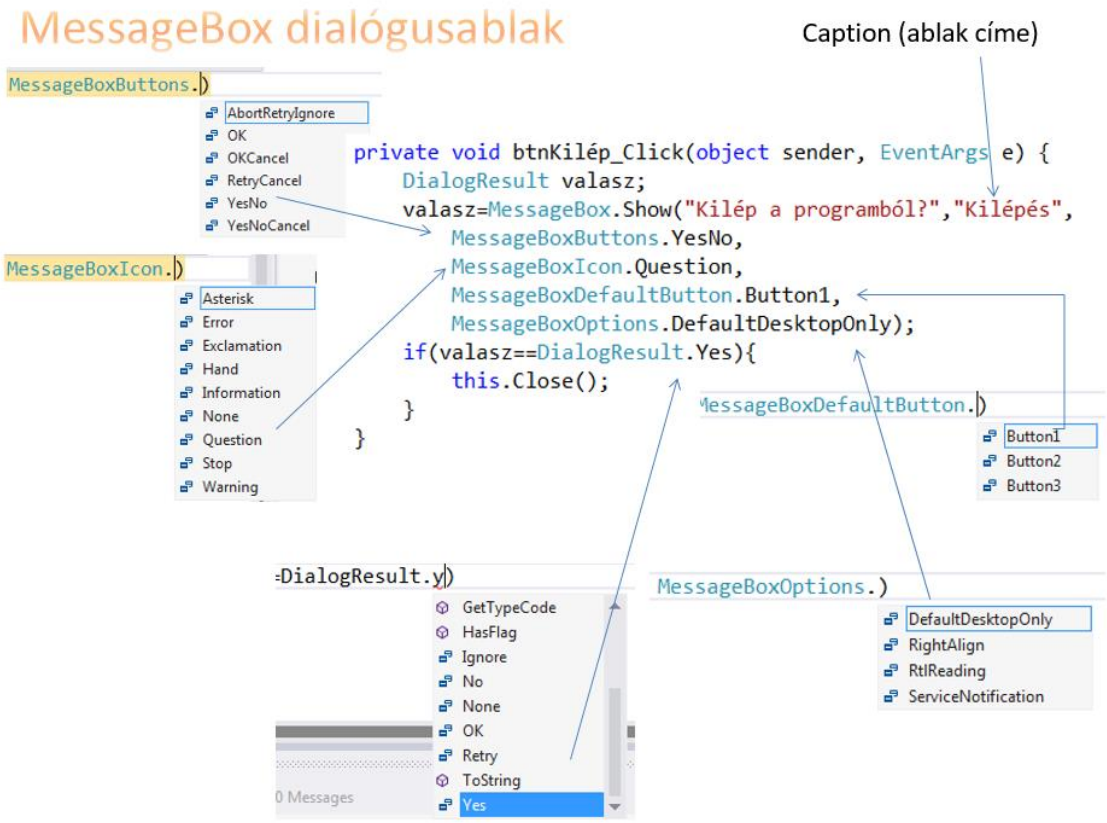
A **MessageBox** dialógusablakban, ha egynél több nyomógombot kívánunk megjeleníteni, akkor mindenképpen figyelni kell a felhasználó válaszreakcióját. Annak megállapítására, hogy éppen melyik gombot választotta a felhasználó a dialógusablak bezárásához, egy **System.Windows.Forms.DialogResult** típusú változót kell deklarálni, amely a **MessageBox.Show()** metódusának visszatérési értékét kapja meg.

Nézzünk egy példát! Egy programban azt szeretnénk, hogy a **Kilépés** gomb megnyomásakor jelenjen meg egy MessageBox, amelyben megkérdezzük a felhasználót, hogy biztosan ki akar-e lépni a programból. A felhasználó lehetséges válaszai, az **Igen** vagy a **Nem** lesz. Ennek megfelelően paraméterezzük a MessageBoxot, jelen esetben két nyomógombot helyezünk el rajta a két lehetséges válasz megadására. Ha a felhasználó válasza az igen gomb lenyomása lesz, akkor kilépünk a programból. Ennek kódja a következő:



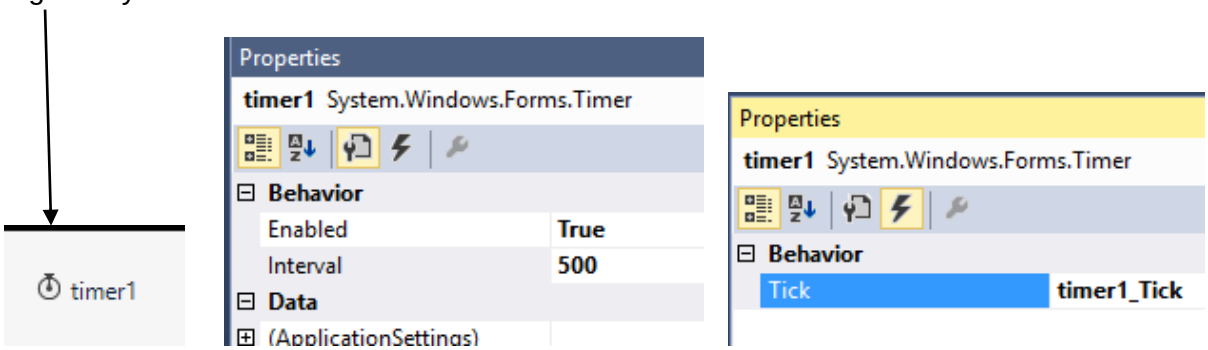

```
private void btnKilép_Click(object sender, EventArgs e) {
    DialogResult valasz;
    valasz=MessageBox.Show("Kilép a programból?","Kilépés",
        MessageBoxButtons.YesNo,MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button1,
        MessageBoxOptions.DefaultDesktopOnly);
    if(valasz==DialogResult.Yes)
    {
        this.Close();
    }
}
```

A programkódban látszik, hogy a MessageBox Show() metódusának a paraméterlistájában a nyomógombokon kívül megadtuk az ablakban megjelenő ikont, valamint az alapértelmezett nyomógombokat. Az alábbi ábráról leolvasható a paraméterek beállítási lehetősége.



1.1.11. A Timer komponens

Az időzítő (Timer) osztály fontos vezérlő, mert segítségével periodikus eseményeket hozhatunk létre. Pontosabban az időzítő már nem is vezérlő, hanem komponens, amely futásidőben nem jelenik meg az ablakban. Tervezés alatt a form alatti komponens tálcán foglal helyet.



Két fontos tulajdonsága van, az Enabled és az Interval. Az Enabled tulajdonság beállítja vagy megadja, hogy az időzítő fut-e. Az Interval tulajdonság az időzítő intervallumát állítja be milliszekundumokban. Legfontosabb metódusai a Start() és a Stop(), melyek elindítják, illetve

megállítják az időzítőt. A timer objektum legfontosabb eseménye a Tick esemény, ami akkor következik be, amikor az időzítő intervalluma lejár.

A Timer időzítőt animációk készítésére is fel tudjuk használni. A beállított intervallumnak megfelelően cserélgetve képeket, filmszerű lejátszást tudunk elérni. A képen egy ilyen animáció kódrészlete látható. Az ImageList gyűjtemény képeit cserélgetjük minden tick esemény bekövetkeztekor. A képeket egy PictureBox vezérlőben jelenítjük meg.

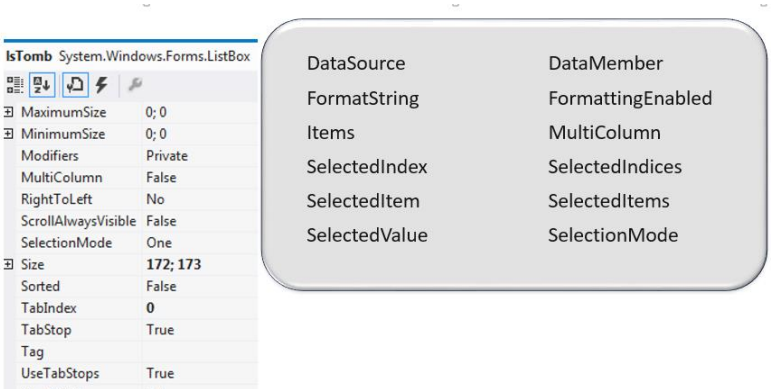
```
private void timer1_Tick(object sender, EventArgs e)
{
   pictureBox1.Image= imageList1.Images[i];
   i = ++i % 4;

}
```

Fontos tudni, hogy a Windows-időzítőket egyszálú (nem pedig többszálú) környezetre tervezték. A .Net Frameworkben több időzítő is rendelkezésünkre áll. A **System.Threading** névtérben található **Timer** osztály az **aszinkron** metódusok használatát támogatja.

A **System.Windows.Forms.Timer** névtér Timer osztályt a Windows Form alkalmazásokhoz használhatjuk. A **System.Timers.Timer** névtérben lévő Timer osztály metódusait, tulajdonságait felhasználói felületeken és munkaszálakban egyaránt felhasználhatjuk.

1.1.12. ListBox vezérlő

Az egyik legegyszerűbb listamegjelenítő vezérlőelem. Egy vagy több elem kijelölését teszi lehetővé. Tartalma futás közben dinamikusan változhat. A már korábban megismert listák tartalmának megjelenítésére kiválóan alkalmas. Az **Items** tulajdonsága egy gyűjtemény, ebben tárolódnak a lista elemei. Az **Items** legfontosabb tulajdonsága a **Count** tulajdonság, ami az elemek számát adja meg. Az **Items** tulajdonság **Add**() metódusával tudunk a listához elemeket hozzáadni, a **Remove**()-val pedig eltávolítani. Néhány további fontos tulajdonság: a SelectedIndex tulajdonsága a lista aktuálisan kijelölt elemének az indexét adja vissza. A SelectedItem tulajdonság beállítja/megadja a listadobozban kijelölt elemet. Többes kiválasztás esetén a kiválasztott elemeket a SelectedIndeces, SelectedItems tulajdonságokkal tudjuk kezelni.



A ListBox vezérlő legfontosabb eseménye a **SelectedIndexChanged**, amely akkor következik be, amikor a **SelectedIndex** tulajdonság értéke megváltozik.

Néhány kódrészlet a ListBox kezelésére, új elem hozzáadása a listához.

```
lstNevsor.Items.Add(nev);
txtNev.Clear();
txtInfo.Text = "Nevek száma: " + lstNevsor.Items.Count.ToString();
```

Elem törlése a listaboxból:

```
txtInfo.Text = "A törölt név:" + lstNevsor.SelectedItem.ToString()
    + "\r\n" + lstNevsor.SelectedIndex.ToString();
lstNevsor.Items.Remove(lstNevsor.SelectedItem);
```

1.1.13. DateTime és String típusok kezelése

A korábbi példákban láttuk, hogy a dátum-idő megadására, kezelésére külön vezérlőelemek állnak rendelkezésünkre. Azt is tudjuk, hogy a szövegdobozokban karakteres, szöveges típusú értékekkel dolgozhatunk. Itt az ideje, hogy részletesebben áttekintsük a DateTime és a String osztály tulajdonságait, metódusait.

DateTime típus

A System.DateTime osztály statikus és példány tagokkal egyaránt rendelkezik.

A DateTime osztály objektumtulajdonságai: Year, Month, Day, Hour, Minute, Second, Millisecond, DayOfWeek (ennek ToString()metódusa adja a tényleges nevet), DayOfYear, TimeOfDay

Statikus tulajdonságok a következők:

- **DateTime.Today**: Mai dátum 00:00:00 idővel.
- **DateTime.Now**: aktuális dátum és idő.
- **DateTime.UtcNow**: mostani UTC idő.
- **DateTime.MinValue** : legkisebb dátum/idő adat, amit tárolni képes.
- **DateTime.MaxValue**: legnagyobb dátum/idő adat, amit tárolni képes.

A DateTime osztály példánymetódusai a következők: AddYears(), AddMonth(), AddDays(), AddHours(), AddMinutes(), AddSeconds().

A dátum-idő értékek megjelenítése során szükség van arra, hogy szabályozzuk a megjelenési formát. A DateTime osztály is rendelkezik a **ToString**() metódussal, amit az object osztálytól örökölt. Az alábbiakban a ToString() metódusban alkalmazható formázási paramétereket mutatjuk be:

• ”d”	rövid dátum	pl. 2007.10.10.
• ”D”	hosszú dátum	pl. 2007. október 10.
• ”F”	teljes dátum, idő	pl. 2007. október 10. 14:12:23
• ”g”	általános dátum, idő	pl. 2007.10.10. 14:12
• ”t”	rövid idő	pl. 14:12
• ”T”	hosszú idő	pl. 14:12:23
• nincs	rövid dátum, hosszú idő	pl. 2007.10.10. 14:12:23

A String osztály

Unikód karakteres szöveget tárol (karakterek gyűjteménye). A String referenciatípus. A String osztály lezárt (sealed) osztály.

Amikor egy létező string objektumnak új értéket adunk, akkor nem az eredeti példány módosul, hanem egy teljesen új objektum keletkezik a memóriában. Bármely objektum értékét stringgé alakíthatjuk a ToString() metódus segítségével. Számjegyekből álló stringen nem végezhetők matematikai műveletek, a + jel a konkatenálás operátora. A string Indexelhető, első karaktere a 0. pozíciójú helyen van.

A String osztály néhány metódusa:

String.Compare (stringa, stringb) – a két string összehasonlításának eredménye:

- 0, ha egyenlő,
- –1 ha az első kisebb mint a második,
- +1 ha nagyobb.

Contains (karaktersorozat) – tartalmazás

- True értékkel tér vissza, ha a karaktersorozat benne van a stringben.

Replace () – az első paraméterének megfelelő karaktereket lecseréli a másodikra.

```
string.Replace('t', 'k')
```

Format() a megadott vezérlőkaraktereknek megfelelő formátumban jelennek meg az adatok.

Példa stringek összehasonlítására:

```
static void Main(string[] args)
{
    string a = "egyik";
    string b = "másik";

    int x = String.Compare(a, b);

    if (x == 0)
    {
        Console.WriteLine("A két string egyenlő");
    }
    else if (x < 0)
    {
        Console.WriteLine("Az 'a' a kisebb");
    }
    else
    {
        Console.WriteLine("A 'b' a kisebb");
    }

    Console.ReadKey();
}
```

Karaktersorozat beszúrása adott pozíciótól az **Insert**() metódussal történhet:

```
st="Varga";
```

```
s2=st.Insert(0,"Dr. ") ⇒ Dr. Varga
```

Karaktersorozat eltávolítása stringből (Remove() metódus):

`st.Remove(2,4)` – a megadott stringből a 2. indexű karaktertől kezdve 4 karakter hosszú karaktersorozatot távolít el.

A ***Concat()*** metódussal két, három vagy több stringet egyetlen stringgé fűzhetünk össze.

```
string Concat(string str0, string str1)
```

```
string.Concat("A", "BC", "DEF")        // ABCDEF
```

Karakter vagy string első, illetve utolsó előfordulásának kezdőpozíciója. Ha nincs találat, –1-et ad vissza

```
IndexOf(char), IndexOf(string)
```

```
LastIndexOf(char), LastIndexOf(string)
```

Karaktersorozat kivágása egy adott stringből:

```
Substring(int kezdőpozíció, int hossz)
```

`Substring(int kezdőpozíció)` ⇒ az utolsó karakterig kiveszi a karaktereket.

A ***PadLeft()*** és a ***PadRight()*** metódusokkal adott hosszú (*totalWidth*) stringet hozunk létre, balról, illetve jobbról kitöltve szóközökkel vagy a megadott *paddingChar* karakterrel.

TrimStart(), **TrimEnd()**, **Trim()** a fehér szóközöket (tab, soremelés) szedi ki a szöveg elejéről, végéről, mindkettőről.

Összehasonlítás az == és a != operátorokkal is lehetséges

Length-tulajdonság a string hosszát adja meg, vagyis a karakterek számát.

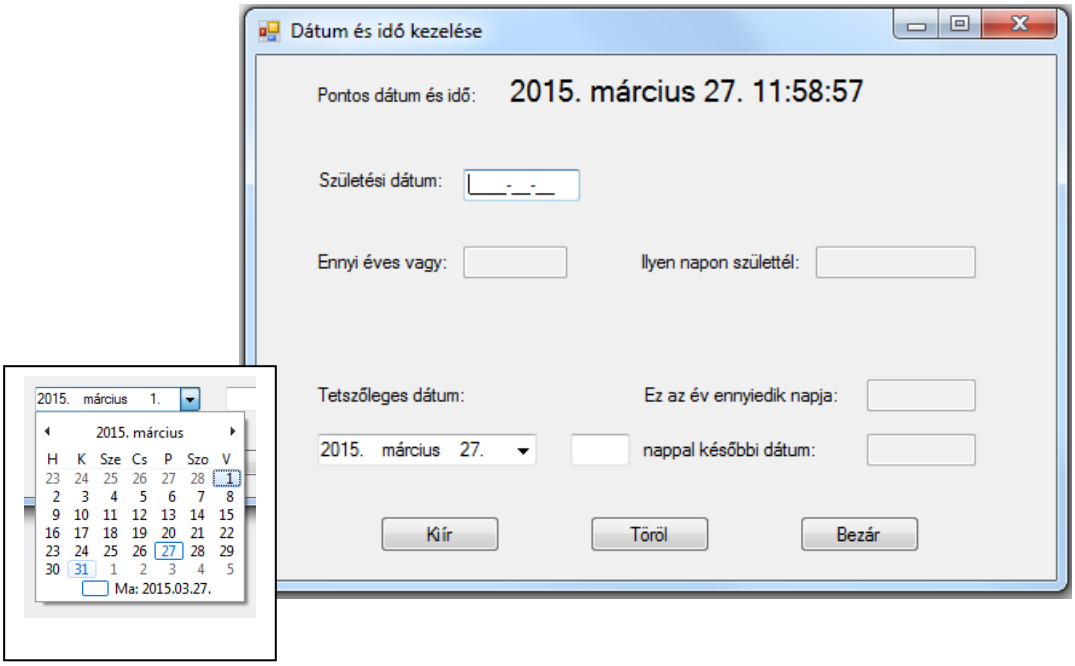
A String osztálynak van egy nagyon fontos metódusa, a Split() metódus. *Visszatérési értéke* egy *karaktertömb*. Segítségével egy hosszú stringet fel tudunk darabolni, ha megadjuk az elválasztó karaktert. A példában az adatsor nevű string vesszővel elválasztott számadatokat tartalmaz. A Split() metódus eredményül egy karaktertömböt ad, amelynek elemeit átkonvertáljuk egész típusúvá és egy egésztípusú tömbben tároljuk.

```
// Szétszedjük
string[] sszam = adatsor.Split(',');
// Áttesszük egész tömmbbe
int[] szam = new int[sszam.Length];
for (int i = 0; i < sszam.Length; i++)
    szam[i] = Convert.ToInt32(sszam[i]);
```

1.2. Gyakorlati példa *(Achs Á.)*

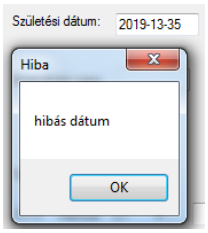
1.2.1. Játék a dátumokkal

A következő fejezetben vezérlőkomponensek használatával kapcsolatos feladatokat oldunk meg. Célunk egy ilyen szerkezetű űrlap létrehozása:

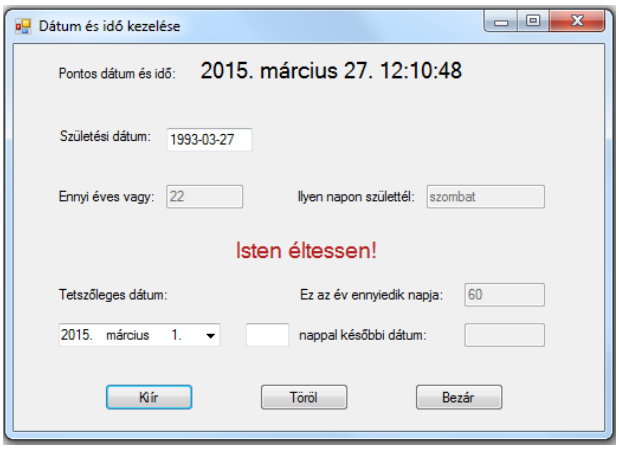


Legfelül egy címkében az **aktuális dátum és idő**, másodpercenként frissülve.

Alatta egy maszkolt szövegmezőbe bekérjük a **születési dátumot**.



Oldjuk meg, hogy a program indulásakor ez legyen az aktuális mező, és addig ne lehessen kilépni belőle, amíg helyes dátumot nem írunk. A helyes azt jelenti, hogy megfelelő formátumú, és az aktuális dátumnál nem későbbi időpont.



Kilépéskor értelemszerűen írjon ki egy gratulációt, ha aznap van a születésnapja. Ha nem aznap van, akkor a **gratulációs címke** tartalma: „Boldog hétköznapot!” legyen.

A **Kiír** gomb hatására jelenjen meg, hogy hány éves az illető, milyen napon született, és már ekkor lehessen látni azt is, hogy a tetszőleges dátum az év hányadik napja.

Figyeljen rá, hogy a napnevet magyarul írjuk ki.

A **tetszőleges dátum** felirat alatt

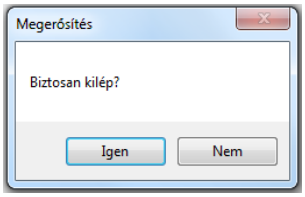
kiválaszthatunk valamilyen dátumot. Minden egyes újabb választáskor jelenjen meg, hogy ez a dátum az év hányadik napja.

Azt is kérdezhessük meg, hogy valahány nappal később (vagy előbb) milyen dátum lesz (volt). Amikor megváltozik a beviteli mező értéke, azonnal írja ki a megváltozott dátumot is. Adjon hibajelzést, ha nem egész szám kerül ebbe a mezőbe.

(Kicsit gondolkoztató rész: hogyan lehet megoldani, hogy negatív számot is gépelhessünk, és azt, hogy ne kiabáljon teljes visszatörléskor.)

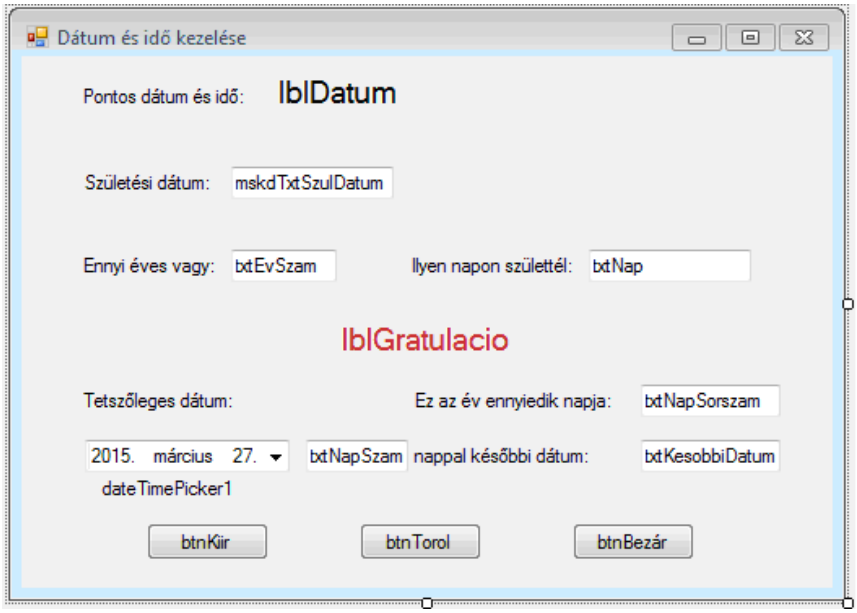
A **Töröl** gomb törli az összes szövegmező és a gratulációs címke tartalmát.

A **Bezár** gomb az ablak bezárására szolgál, de bezárás előtt kérjen megerősítést:



Lehetséges megoldásrészletek:

A megoldásban az ábrán látható elnevezéseket használjuk (mivel az alkalmazásban egyetlen dátumválasztó doboz van, ezért ezt most nem neveztük át):



Mivel magyar dátumformátumot szeretnénk, ezért a maszk megadásakor nekünk kell beállítani az elvárt 0000-00-00 formátumot.

1. Aktuális dátum kiíratása:

A feladat azt kéri, hogy már induláskor lássuk az aktuális dátumot. Azokat az utasításokat, amelyeket a program indulásakor azonnal el kell végezni, a form betöltésének (`Load`) eseményéhez rendeljük. (Ez a form alapértelmezett eseménye, ezért a formra duplán kattintva is elérhető.)

Mielőtt megírnánk az eseményhez tartozó megoldást, még egy dolgot gondoljunk végig:

A megoldás során több eseménykezelő is használja a mai dátum, a születési dátum és a kiválasztott dátum értékét is. Ezért ezeket a változókat globálisan, mezőként célszerű definiálni.

```
private DateTime ma = DateTime.Now;
private DateTime szulDatum;
private DateTime valasztottDatum;

private void Form1_Load(object sender, EventArgs e) {
    lblDatum.Text = ma.ToString("F");
    lblGratulacio.Text = "";
    valasztottDatum = dateTimePicker1.Value;
}
```

A választott dátumot azért célszerű már az elején is beállítani, hogy azonnal értelmezni tudja a dátumválasztóban lévő értéket.

Több jó megoldás is lehet arra, hogy a két megadott dátum hol kapjon értéket. Kaphatnak deklarációkor, a form betöltésekor, de kaphatnának a konstruktoron belül is.

2. Másodpercenkénti frissülés:

Azt is meg kell oldanunk, hogy az aktuális dátum kiírása másodpercenként frissüljön. Ahhoz, hogy így legyen, egy `Timer` típusú komponensre van szükségünk. Ha a tervezési módban felrakjuk a formra, akkor azt tapasztaljuk, hogy a komponens lekerül a tervezési felület alatti mezőbe – ide egyébként közvetlenül is be lehetne rakni. Ebbe a mezőbe kerülnek az olyan komponensek, amelyek nem jelennek meg látható módon a felületen, de hozzá kapcsolódnak.

A `Timer` működése során bizonyos időközönként végrehajtja az általunk beállított utasításokat. A komponens két tulajdonságát kell beállítanunk: Az `Enabled` tulajdonságot `true`-ra, és az `Interval` tulajdonságot annyi millisecundumra, amilyen időközönként végre akarjuk hajtatni az utasításainkat. Esetünkben ez 1000 ms.

A `Timer` egyetlen eseménye a `Tick`, ennek kezelésekor hajtja végre a megadott utasításokat (mivel most csak ez az egyetlen timer a programban, nem neveztük át):

```
private void timer1_Tick(object sender, EventArgs e) {
    lblDatum.Text = DateTime.Now.ToString("F");
}
```

Megjegyzés: Jogosan vetődhet fel a kérdés, hogy miért nem használtuk a `ma` nevű változót, hiszen egyszer már beállítottuk az értékét. Próbálja ki, mi történik ekkor! Azt tapasztalja, hogy nem lesz változás, mégpedig azért nem, mert a `ma` nevű változóban nem a pillanatnyi gépidő szerepel, hanem az, amennyi a program indulásakor volt. Ez az érték teljesen jó a többi feladatrészhez, ott nincs szükség ezred-másodpercenkénti frissülésre, vagyis nincs értelme, hogy állandóan lekérdezzük a gépidőt, ennél a feladatrésznél azonban ez kikerülhetetlen.

3. Születési dátum bekérése:

Azt szeretnénk, ha a születési dátum megadása után bárhova kattintunk (csak aktív vezérlőelemre tudunk kattintani), akkor azonnal jelenjen meg a születésnapi gratuláció szövege. Ez azt jelenti, hogy a mező elhagyásakor kell kiírnunk az üzenetet, vagyis akkor, amikor a fókusz máshova kerül. Ez a `Leave` esemény (a `Focus` események közül, de ha névsorban keresi az eseményeket, akkor persze névsor szerint találja meg ☺):

Írjuk meg az eseményhez tartozó metódust. Első nekifutásra evvel a kódrészlettel próbálkozhatunk:

```
private void mskdSzuldatum_Leave(object sender, EventArgs e) {
    szulDatum = DateTime.Parse(mskdTxtSzulDatum.Text);

    if (szulDatum.Month == ma.Month && szulDatum.Day == ma.Day) {
        lblGratulacio.Text = "Isten éltessen!";
    }
    else {
        lblGratulacio.Text = "Boldog hétköznapot!";
    }
}
```

Vagyis, ha a születési dátum hónapja és napja azonos a mai dátum hónapjával és napjával, akkor írhatjuk a köszöntő szöveget, egyébként pedig jelezhetjük, hogy nem ma van a születésnap.

Csakhogy örömünk még korai. Ha ugyanis „gonoszkodó” módon kipróbáljuk pl. az 1234-56-78 „dátumot”, akkor programunk azonnal elszáll, olyannyira, hogy még lelőni sem lesz könnyű. (Általában szokja meg, hogy programját nem csak „normál” adatokkal próbálja ki, hanem „extrémekkel” is, hiszen ekkor derül ki, hogy valóban gondoltunk-e minden lehetőségre.) Másrészt hiába ígérte a dokumentáció azt, hogy a maszkba írt 0 kötelező számkitöltést jelent, a futtatókörnyezet mégis elfogadja, ha egy-egy számjegy helyét üresen hagyjuk. Emiatt ellenőriznünk kell, hogy igaz-e a maszkolt szövegdoboz `MaskCompleted` tulajdonsága.

Ha egy programot ilyen könnyen „meg lehet ölni”, akkor ebben nem a felhasználó, hanem a programozó hibás. Nekünk kell gondolni az elkövethető hibákra, és kivédeni őket amennyire csak lehet. Egy áramszünetet vagy rendszerhibát nyilván nem tudunk kivédeni, de a fent említett „elgépelést” igen. Erre nagyon jól használható eszköz a kivételkezelés. A végrehajtandó kódrészletet a `try` blokkba írjuk, és normál körülmények között ugyanúgy le is fut, mint eddig. Viszont írunk egy `catch` blokkot is, amelyben azt írjuk le, hogy mi történjen, ha valami kezelhető hiba, azaz kivétel keletkezik a `try` blokkban megírt kódrészletek futásakor.

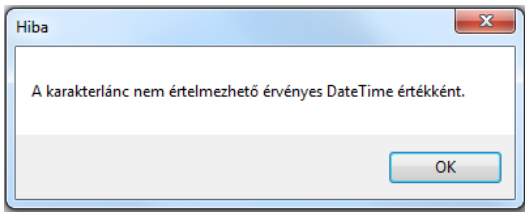
A jelzett hibát így lehet kivédeni:

```
private void mskdSzuldatum_Leave(object sender, EventArgs e) {
    try {
        if (!mskdTxtSzulDatum.MaskCompleted) throw new FormatException();
        else {
            szulDatum = DateTime.Parse(mskdTxtSzulDatum.Text);
            if (szulDatum.Month == ma.Month && szulDatum.Day == ma.Day) {
                lblGratulacio.Text = "Isten éltessen!";
            }
            else {
                lblGratulacio.Text = "Boldog hétköznapot!";
            }
        }
    }
    catch (FormatException) {
        MessageBox.Show("hibás dátum", "Hiba");
    }
}
```

Ha most próbáljuk ki az 1234-56-78 „dátumot”, akkor hibaüzenetet kapunk, amelynek tudomásulvétele után ki is javíthatjuk a hibánkat, míg az előbb, kivételkezelés nélkül csak leállítani tudtuk a programunkat.

Megjegyzés: A `catch` ágban elvileg kiírathatnánk a kivételobjektum eredeti üzenetét is:

```
catch (FormatException ex) {
    MessageBox.Show(ex.Message, "Hiba");
}
```



A közölt változat egyetlen szépséghibája, hogy a rendszerüzenet nem mindig világos egy felhasználó számára, pláne, ha esetleg nem is magyarul, hanem angolul írja ki. Emiatt írtunk saját hibaüzenetet, viszont fontos megemlíteni azt, hogy fejlesztéskor mindig a rendszerüzeneteket kérjük le, hiszen fejlesztőként pontosan ezekre az üzenetekre van szükségünk az esetleges programhibák javításához.

Amikor saját üzenetet írunk a rendszerüzenet helyett, akkor elvileg nincs is szükségünk a `catch` blokk paraméterében szereplő változóra, csak a kivétel típusára. A C# lehetőséget ad rá, hogy ilyenkor elhagyjuk a változót. (Sőt arra is van lehetőség, hogy elhagyjuk a kivétel típusát is, de ezt nem javasoljuk, mert fontos, hogy lássuk, milyen kivételt kezelünk.)

Még egy dolgot beszéljünk meg: nemcsak az a fontos, hogy helyes dátumformátumot adjunk meg, hanem az is, hogy születési dátumként ne gépelhessünk be az aktuális dátumnál későbbi időpontot. Azt, hogy a megadott születési dátum későbbi időpont-e a mainál, már csak akkor tudjuk vizsgálni, ha kiderült, helyes formátumú a begépelt dátum. De ha ez későbbi a mai dátumnál, akkor ismét hibaüzenetet kellene küldenünk. Ez persze, továbbra is megoldható `if-else` szerkezettel, de sokkal elegánsabb és kényelmesebb is, ha ezt is kivételkezeléssel valósítjuk meg. Ehhez az kell, hogy akkor, ha a begépelt dátum későbbi, mint a mai, szintén kivétel keletkezzen. Kivételt mi magunk is generálhatunk (a szokásos

szóhasználat: kivételt dobunk) a `throw` parancs segítségével, amelyet majd a `catch` blokkban kapunk el. Elvileg ugyan ekkor is dobhatnánk `FormatException` kivételt, csak nem lenne logikus a programunk, hiszen most nem formai hibáról van szó.

Szerencsére azonban a `catch` blokkban egymás után több kivételt is el lehet kapni, csak figyelni kell a kivételek hierarchiájára, így feladatunk logikusan is megoldható:

```
private void mskdSzuldatum_Leave(object sender, EventArgs e) {
    try {
        if (!mskdTxtSzulDatum.MaskCompleted) throw new FormatException();
        else {
            szulDatum = DateTime.Parse(mskdTxtSzulDatum.Text);
            if (szulDatum > ma) throw new ArgumentOutOfRangeException();
            if (szulDatum.Month == ma.Month && szulDatum.Day == ma.Day) {
                lblGratulacio.Text = "Isten éltessen!";
            }
            else {
                lblGratulacio.Text = "Boldog hétköznapot!";
            }
        }
    }
    catch (FormatException) {
        MessageBox.Show("hibás dátum", "Hiba");
    }
    catch (ArgumentOutOfRangeException) {
        MessageBox.Show("Túl késői dátum", "Hiba");
    }
}
```

Természetesen ez az igazán jó megoldás. Időnként azonban „kényelmesek” vagyunk, és nem akarjuk ennyire részletezni az egyes kivételeket. Most is gondolhatjuk azt, hogy ha valaki meglátja a „Hibás dátum” üzenetet, akkor nemcsak arra jön rá, hogy hibás formátumot írt, hanem talán arra is, hogy az aktuálisnál későbbit. Vagyis, ha „lusták” vagyunk, akkor most akár össze is vonhatjuk a két hibaüzenetet, és elég, ha csak az általános kivételt (`Exception`) figyeljük. Az így változtatott metódus:

```
private void mskdSzuldatum_Leave(object sender, EventArgs e) {
    try {
        if (!mskdTxtSzulDatum.MaskFull) throw new Exception();
        else {
            szulDatum = DateTime.Parse(mskdTxtSzulDatum.Text);
            if (szulDatum > ma) throw new Exception();
            if (szulDatum.Month == ma.Month && szulDatum.Day == ma.Day) {
                lblGratulacio.Text = "Isten éltessen!";
            }
            else {
                lblGratulacio.Text = "Boldog hétköznapot!";
            }
        }
    }
    catch (Exception) {
        MessageBox.Show("hibás dátum", "Hiba");
    }
}
```

A feladat azt is kéri, hogy induláskor a születési dátum megadására szolgáló mezőben villogjon a kurzor. Ezt legegyszerűbben úgy tehetjük meg, ha a mező `TabIndex` tulajdonságát 0 értékre állítjuk (még a tervezési nézetben). Bár kicsi az esélye, hogy valaki itt gonoszkodna, de elvileg megtehető az is, hogy több különböző vezérlőelem `TabIndex` értékét is ugyanakkorára állítjuk, ekkor nincs rá garancia, hogy a több egyforma közül épp melyik lesz az aktuális. Ezért, ha garantáltan biztosak akarunk lenni a dolgunkban, akkor a form betöltésekor célszerű még ezt az utasítást is kiadni: `ActiveControl = mskdTxtSzulDatum;`

További kérés, hogy addig ne is lehessen kilépni, amíg nem gépeltünk helyes dátumot. (Nem túl barátságos egy ilyen program, de időnként szükség van az ehhez hasonló erőszakra.)

Ezt úgy oldhatjuk meg, hogy az előbb tárgyalt kódrészlet `catch` blokkját így alakítjuk:

```
catch (Exception) {
    MessageBox.Show("Hibás dátum!", "Hiba");
    mskdTxtSzulDatum.Focus();
}
```

Bár a feladat azt kérte, hogy az esemény a mező elhagyásához legyen kötve, de adatmegadáshoz általában hozzá szokás kötni (vagy legalábbis illik, még akkor is, ha nem minden éles program csinálja így) azt is, hogy Enter megnyomásával is érvényesíthessük a begépelt adatot. Oldjuk meg ezt is.

A mezőhöz hozzá kell rendelnünk a `KeyPress` eseményt is, és meg kellene oldanunk, hogy az Enter lenyomásakor is ugyanaz fusson le, mint amit az előbb megírtunk. Természetesen NEM copy-paste-tel oldjuk meg a duplázást, hanem úgy, hogy erről a közös részről külön metódust írunk, és mindkét esemény kezelésekor ezt hívjuk meg.

Azt, hogy melyik billentyűt nyomtuk meg, a `KeyPressEventArgs` típusú `e` nevű objektum `KeyChar` tulajdonságának vizsgálatával oldhatjuk meg:

```
private void mskdTxtSzulDatum_KeyPress(object sender, KeyPressEventArgs e) {
    if (e.KeyChar == 13) Gratulacio();
}
```

(13 az Enter kódja.)

Megjegyzés: Az entergépelést a `KeyDown` esemény figyelésével is megoldhatjuk:

```
private void mskdTxtSzulDatum_KeyDown(object sender, KeyEventArgs e) {
    if (e.KeyCode == Keys.Enter) {
        Gratulacio();
    }
}
```

(Ilyen módon sokkal több kódot tudunk figyelni.)

A hivatkozott `Gratulacio()` metódus:

```
private void Gratulacio() {
    try {
        if (!mskdTxtSzulDatum.MaskFull) throw new Exception();
        else {
            szulDatum = DateTime.Parse(mskdTxtSzulDatum.Text);
            if (szulDatum > ma) throw new Exception();
            if (szulDatum.Month == ma.Month && szulDatum.Day == ma.Day) {
                lblGratulacio.Text = "Isten éltessen!";
            }
            else {
                lblGratulacio.Text = "Boldog hétköznapot!";
            }
        }
    }
    catch (Exception) {
        MessageBox.Show("hibás dátum", "Hiba");
        mskdTxtSzulDatum.Focus();
    }
}
```

A `Leave` esemény ennek megfelelően átalakított kezelése:

```
private void mskdTxtSzulDatum_Leave(object sender, EventArgs e) {
    Gratulacio();
}
```

4. Kiir gomb:

Ehhez egyelőre nem kell túl sok magyarázatot fűzni:

```
private void btnKiir_Click(object sender, EventArgs e) {
    txtEvSzam.Text = (ma.Year - szulDatum.Year).ToString();
    txtNap.Text = szulDatum.DayOfWeek.ToString();
    txtNapSorszam.Text = valasztottDatum.DayOfYear.ToString();
}
```

Legföljebb annyit említhetnénk meg, hogy az évszám így nem teljesen pontos, azt adja meg, hogy ebben az évben hány éves az illető, függetlenül attól, hogy az év mely napján született.

Van viszont egy megbeszélni való, mégpedig az, hogy a nap nevét angolul kapjuk meg. Kicsit talán meglepő, hogy ha a hónap nevét magyarul közli a rendszer, akkor vajon a hét nevét miért nem, de most ne ezen ógjunk-mógjunk, hanem javítsuk ki a hibát.

Mielőtt tiltakozna, hogy ez nem is hiba, vagy hogy sok helyen lehet így látni, hadd jegyezzük meg, hogy elég baj az, hogy sok helyen tapasztalhatunk hasonlót. Ahol ilyen a kiírás, ott azon is el lehet gondolkozni, hogy vajon mennyire érdemes használni a programot, hiszen szemmel láthatóan lusta (és igénytelen) a programozója. Szokjon hozzá, hogy egy felületen nem keverjük a nyelveket (hacsak nem indokolt, mert pl. szótár programot ír).

A feladatot pillanatok alatt meg lehetne oldani egy egyszerű kis saját metódus segítségével (a metódus minden angol szóhoz hozzárendelné a magyar párját), de most egy más, általánosabb lehetőséget mutatunk be, mégpedig a `CultureInfo` osztály használatát. Nézzen

utána, mi mindent tud ez az osztály, most csak annyit, hogy ennek segítségével beállíthatjuk egy-egy nyelvi kultúra specialitásait, pl. a hét neveit is.

A konstruktorban állítsuk be az aktuális kultúrát:

```
Thread.CurrentThread.CurrentCulture = new CultureInfo("hu-HU");
```

A kiíratást így módosíthatjuk:

```
private void btnKiir_Click(object sender, EventArgs e) {
    int evek = ma.Year - szulDatum.Year;
    txtEvSzam.Text = evek.ToString();
    string nap = CultureInfo.CurrentCulture.DateTimeFormat.GetDayName(szulDatum.DayOfWeek);
    txtNap.Text = nap;
    txtNapSorszam.Text = valasztottDatum.DayOfYear.ToString();
}
```

Mindez csak akkor működik, ha az elején beállítjuk a névtereket is, vagyis:

```
using System.Globalization;
using System.Threading;
```

5. Dátumválasztás:

A `dateTimePicker1` értékváltozás eseményének bekövetkeztekor ki kell írnunk, hogy a választott dátum az év hányadik napja:

```
private void dateTimePicker1_ValueChanged(object sender, EventArgs e) {
    valasztottDatum = dateTimePicker1.Value;
    txtNapSorszam.Text = valasztottDatum.DayOfYear.ToString();
}
```

6. Későbbi dátum kiírása:

Még egy lényegi feladatunk van, ki kell íratni a választott dátumhoz képest valahány nappal későbbi (vagy korábbi) dátumot. Azt, hogy hány nappal tér el, a `txtNapSzam` nevű mezőben adhatjuk meg. A feladat szerint: mihelyt megváltozik a mező tartalma, azonnal lehessen látni az új dátumot. Ez meglehetősen egyszerű feladat:

```
private void txtNapSzam_TextChanged(object sender, EventArgs e) {

    try {
        int nap = int.Parse(txtNapSzam.Text);
        txtKesobbiDatum.Text = valasztottDatum.AddDays(nap).ToString("d");
    }
    catch (Exception) {
        MessageBox.Show("Nem számot írt", "Hiba");
    }
}
```

(Ne felejtse el, hogy a választott dátum kezdőértékét már a form betöltésekor megadtuk.) Nem is lenne mit magyarázkodni a megoldáson, ha közben nem merülne fel egy probléma, illetve, ha jól számoljuk, akár kettő is:

1. Az első számot begépelve még jól működik, de ha visszatöröljük, hogy beírjuk a következőt, a visszatörlés hatására keletkező üres mező miatt hibajelzést kapunk.

2. De akár már az első adat begépelésekor is kaphatunk hibaüzenetet, ha negatív számot gépelünk.

Persze, ha kellően kitartóak vagyunk, és a hibaüzenet ellenére folytatjuk a gépelést, akkor működik a program (negatív szám esetén korábbi dátumot ír ki), de az mégsem fogadható el, hogy közben hibaüzenetet kapjunk.

A megoldás az, hogy számoljuk a begépelt karaktereket. Csak akkor kezdi el figyelni, hogy helyes formátumú-e a begépelt adat, ha legalább egy karakter már van benne, sőt, ha ez a karakter a negatív előjel, akkor csak a második karaktertől kezdve figyeli:

```
private void txtNapSzam_TextChanged(object sender, EventArgs e) {

    int hossz = txtNapSzam.Text.Length;
    if ((hossz >= 2 && txtNapSzam.Text.ElementAt(0) == '-') ||
        (hossz >= 1 && txtNapSzam.Text.ElementAt(0) != '-')) {
        try {
            int nap = int.Parse(txtNapSzam.Text);
            txtKesobbiDatum.Text = valasztottDatum.AddDays(nap).ToString("d");
        }
        catch (Exception) {
            MessageBox.Show("Nem számot írt", "Hiba");
        }
    }
}
```

Gyakorlatilag készen vagyunk. Persze, hátra van még két gomb, illetve a hozzájuk rendelt események.

7. Törlés

A törlést nyilván meg tudná oldani önállóan is, de hadd mutassak meg egy ügyesebb megoldást. (Persze, nem kizárt, hogy már eleve erre a megoldásra gondolt. ☺)

```
private void btnTorol_Click(object sender, EventArgs e) {
    foreach (var item in this.Controls) {
        if (item is TextBox) ((TextBox)item).Clear();
    }
    lblGratulacio.Text = "";
    mskdTxtSzulDatum.Clear();
}
```

8. Bezárás

Nyilván a bezárást is meg tudná oldani, de még nézzük meg, hogyan kérdezhetjük meg a felhasználótól, hogy tényleg ki akar-e lépni.

Ehhez felhasználjuk azt, hogy a `MessageBox Show` metódusa mindig visszaad valamilyen választ (elemi esetben azt, hogy megnyomtuk az OK gombot). Ez egy `DialogResult` típusú adat, és a `MessageBox` bezárásakor megvizsgálható, hogy melyik gombot nyomtuk meg. Esetünkben:

```
private void btnBezár_Click(object sender, EventArgs e) {
    DialogResult valasz = MessageBox.Show("Biztosan kilép?", "Megerősítés",
                                          MessageBoxButtons.YesNo);
    if (valasz == DialogResult.Yes) this.Close();
}
```

1.2.2. Játék a dátumokkal – ráadás

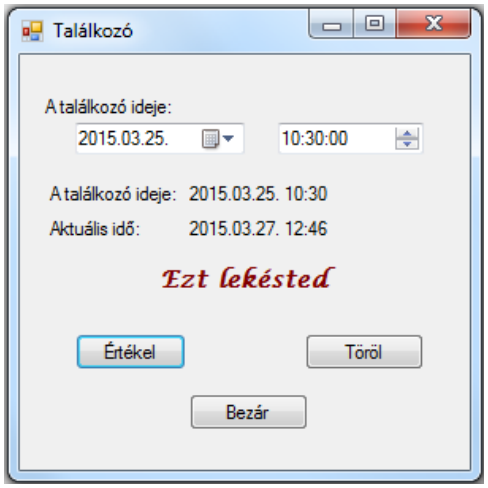
Az előzőek ráadásaképpen nézzünk meg még egy kis alkalmazást.

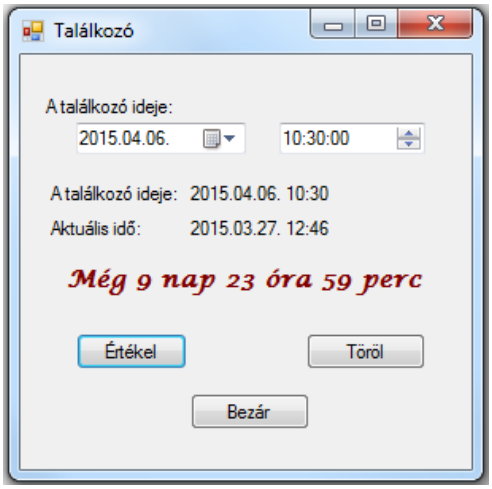
Kérjük be két DateTimePicker segítségével egy találkozó **dátumát és idejét**! A másodikat úgy állítsuk be, hogy az időt mutassa, és a le-föl nyilak segítségével lehessen beállítani az órát, percet, másodpercet.

Az **Értékel** gomb megnyomására az alábbiak történhetnek: Ha ez egy jövőbeni találkozó, akkor írjuk ki egy címkébe az addig **visszalevő időt** (napok száma és óra, perc). Ha már elmúlt a találkozó időpontja vagy pont most van, akkor a címkébe az „Ezt már lekésted!” szöveget írjuk. A címke kerüljön középre (vízszintesen).

Az ellenőrizhetőség kedvéért írassuk ki a találkozó idejét és az aktuális időt is.

A másik két gomb hatása a szokásos.

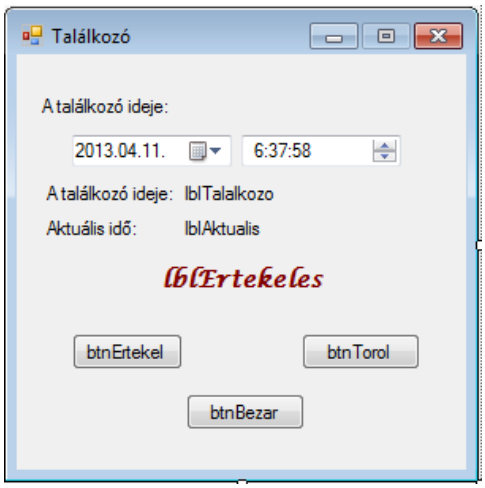




Lehetséges megoldásrészletek:

A megoldásban az ábrán látható elnevezéseket használjuk:

A két dátumválasztó neve: `dtTmPckrDatum` és `dtTmPckrIdo`.



Néhány formai megjegyzés:

1. Ha azt szeretnénk, hogy ne lehessen átméretezni az alkalmazást, akkor a form tulajdonságai közül a `FormBorderStyle` tulajdonságot kell megváltoztatni valamelyik `Fixed...` stílusra. Ekkor nem lehet majd átméretezni, de még fel lehet növelni a méretét teljes képernyősre. Ha ezt sem szeretnénk, akkor a `MaximizeBox` tulajdonságot kell `false`-ra állítani.

2. Ha azt szeretnénk, hogy az értékelő felirat az alkalmazás közepén jelenjen meg, akkor az `lblErtekeles` címkét így kell beállítani:

Az `AutoSize` tulajdonságát `false`-ra állítjuk, ekkor változtatható lesz a címke mérete. Úgy állítjuk be, hogy a címke az alkalmazás egyik szélétől a másikig tartson. A `TextAlign` tulajdonságot `MiddleCenter`-re állítjuk (kiválasztható egy kínálatból). Ez azt határozza meg, hogy a felirat hova kerüljön a címkén belül.

3. A `dtTmPckrIdo` beállítása: Állítsuk be a `Format` tulajdonság értékét `Time`-ra, a `ShowUpDown` tulajdonságot pedig `true`-ra.

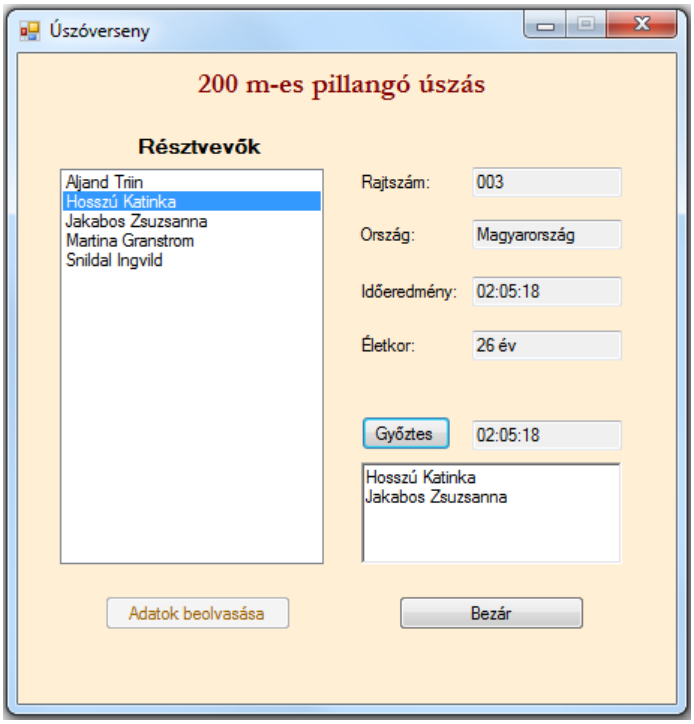
Ezek után már gyerekjáték a megoldás.

```
private DateTime ma;

private void Form1_Load(object sender, EventArgs e) {
    ma = DateTime.Now;
    //lblAktualis.Text = ma.ToString();
    lblAktualis.Text = ma.ToString("g");
}

private void btnErtekel_Click(object sender, EventArgs e) {
    DateTime datum, ido, talalkozo;
    datum = dtTmPckrDatum.Value;
    ido = dtTmPckrIdo.Value;
    talalkozo = datum.Date + ido.TimeOfDay;

    // Csak próbaképpen:
    //lblTalalkozo.Text = datum + "--" + ido;

    lblTalalkozo.Text = datum.ToShortDateString() + " " + ido.ToShortTimeString();
    if(talalkozo <ma){
        lblErtekeles.Text = "Ezt lekésted";
    }else{
        TimeSpan hatraLevo = talalkozo - ma;
        lblErtekeles.Text = "Még " + hatraLevo.Days + " nap " +
                                     hatraLevo.Hours + " óra " +
                                     hatraLevo.Seconds + " perc.";
    }
}
```

1.2.3. Úszóverseny

Készítsünk programot egy úszóversenyen részt vevők adatainak és eredményeinek megjelenítésére. A versenyzők adatait egy szöveges fájlból olvassuk be (*uszok.txt*). A fájl egy-egy sorában pontosvesszővel elválasztva a versenyző egyedi rajtszáma, neve, születési dátuma, melyik ország versenyzője és időeredménye szerepel.

Olvassuk be a versenyzők adatait, majd a versenyzők nevét listázzuk ki egy listadobozban. A versenyző többi adata csak olvasható szövegdobozokban jelenjen meg, amikor kiválasztunk egy-egy versenyzőt a listából. Az alkalmazás (nem átméretezhető) űrlapja a következő legyen:



Az adatok ne kerülhessenek be többször a listába, és névsorba rendezve jelenjenek meg (a listadoboz megfelelő tulajdonságának beállításával). A „Győztes” gomb hatására jelenjen meg a győztes időeredmény és a győztes vagy győztesek neve egy `RichTextBox` típusú szövegdobozban.

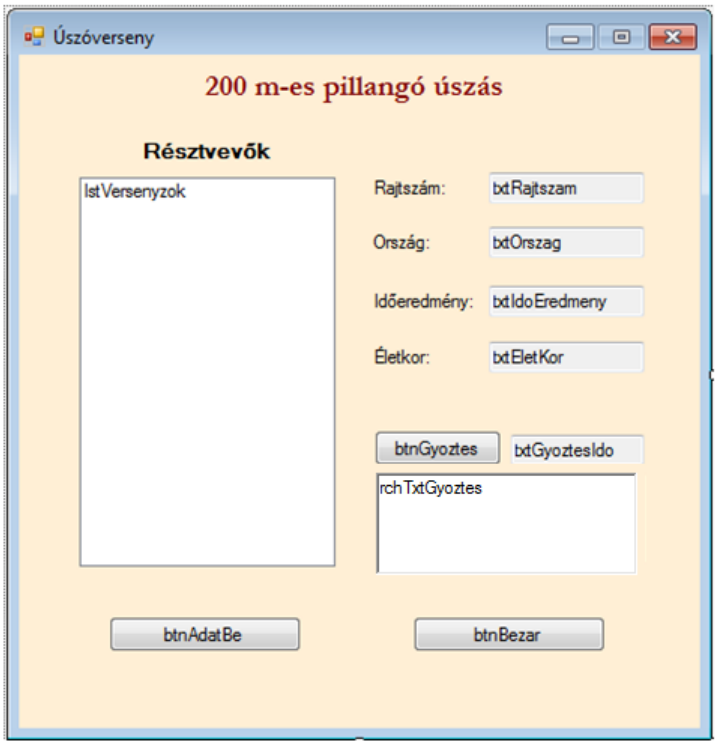
Lehetséges megoldásrészletek:

A megoldáshoz létre kell hoznunk a `Versenyzo` osztályt, illetve a form kezeléséért felelős `VersenyForm` osztályt.

A `Versenyzo` osztály:

```
class Versenyzo {
    public string Rajtszam {get; private set;}
    public string Nev { get; private set; }
    public DateTime SzulDatum { get; private set; }
    public string Orszag { get; private set; }
    public TimeSpan IdoEredmeny { get; private set; }

    // konstruktor
    public Versenyzo(string rajtszam, string nev, DateTime szulDatum,
                                      string orszag, TimeSpan idoEredmeny) {
        this.Rajtszam = rajtszam;
        this.Nev = nev;
        this.SzulDatum = szulDatum;
        this.Orszag = orszag;
        this.IdoEredmeny = idoEredmeny;
    }

    public override string ToString() {
        return Nev;
    }
}
```

A megoldásban az ábrán látható elnevezéseket használjuk:



A vezérlőelemek felrakásakor célszerű beállítani a `btnGyoztes` gomb `Enable` tulajdonságát `false`-ra, hogy még csak véletlenül se lehessen hamarabb megnyomni, mint az adatbevitel gombot.

Először azt gondoljuk végig, hogyan olvasnánk be az adatokat a megadott adatfájlból, és mit kezdenénk velük.

Pontosan ugyanazt, mint amit konzolos alkalmazáskor tettünk. Egyetlen egy különbség van, mégpedig az, hogy a korábbi példákban az adatfájl soronként egyetlen adatot tartalmazott, most viszont egy-egy versenyző adatai vannak egy-egy sorban.

A sorokat hajszálpontosan ugyanúgy olvassuk be, mint eddig, csak a beolvasott sort most még fel kell darabolnunk, hogy tényleges adatok legyenek belőle.

Kérdés még, hogy mit tegyünk a beolvasott adatokkal. Hajszálpontosan ugyanazt, mint konzolos alkalmazás esetén, azaz az adatok alapján hozzunk létre egy versenyző példányt, és tegyük a versenyzőket tartalmazó listába.

Ezt valóban csinálhatnánk így is, de akkor még el kellene érnünk azt, hogy ezek a lista-elemek meg is jelenjenek a listadoboz felületén. Ezért kényelmesebb az a megoldás, hogy a versenyző példányokat nem külön listába rakjuk, hanem azonnal a listadobozhoz tartozó adatlistába. Ezt az adatlistát speciális módon nevezik: ez a listadoboz `Items` tulajdonsága. (Vagyis az `Items` a listadobozhoz tartozó objektumok listája.)

```
private void AdatBeolvasas() {
    StreamReader olvasoCsatorna = new StreamReader("uszok.txt");

    string adat = olvasoCsatorna.ReadLine();
    while (adat != null) {
        Feldolgoz(adat);
        adat = olvasoCsatorna.ReadLine();
    }
    olvasoCsatorna.Close();
}

private void Feldolgoz(string adat) {
    string rajtSzam, nev, orszag;
    DateTime szulDatum;
    TimeSpan idoEredmeny;
    Versenyzo versenyzo;
    string[] tordelt = adat.Split(';');
    // 001;Jakabos Zsuzsanna;1989.04.03;Magyarország;2:05.18

    rajtSzam = tordelt[0];
    nev = tordelt[1];
    szulDatum = DateTime.Parse(tordelt[2]);
    orszag = tordelt[3];
    idoEredmeny = TimeSpan.Parse("0:" + tordelt[4]);
    versenyzo = new Versenyzo(rajtSzam, nev, szulDatum, orszag, idoEredmeny);
    lstVersenyzok.Items.Add(versenyzo);
}
```

Már csak annyi teendőnk van, hogy hívjuk meg ezt a metódust, és jelenítsük meg a beolvasott adatokat. Ezt konzolos alkalmazásban úgy oldottuk meg, hogy a metódust egy indító metódusból hívtuk meg, és a versenyzők adatait a konzolos képernyőre írtuk.

Jelenleg egy esemény hívja meg a metódust – a mostani feladat szerint egy gombnyomás esemény, de szólhatna úgy is a feladat, hogy induláskor azonnal jelenjenek meg az adatok, ekkor a form-load eseményhez kellene rendelnünk a metódushívást.

De hol jelenjenek meg az adatok? Közvetlenül a formra nem rakhatjuk őket, no meg határozottan ronda is lenne, ha oda tennénk. Az adatok megjelenítésére szolgál a listadoboz, azaz egy `ListBox` típusú vezérlőkomponens. Ennek esetünkben ugyanaz a funkciója, mint konzolos alkalmazás esetén a képernyő szerepe volt. De ha csak ennyi lenne, akkor az adatokat egy többsoros szövegdobozba vagy egy `RichTextBox` vezérlőkomponensbe is tehetnénk. És ha valóban csak annyit szeretnénk, hogy megjelenítsük az adatokat, akkor tényleg szövegdobozba kellene raknunk őket, ahogy a győzteseket is szövegdobozba írjuk majd.

Most azonban egy kicsit többet is szeretnénk, mégpedig azt, hogy ki is tudjunk választani egy-egy versenyzőt. Ezért van szükségünk listadobozra, mert ez a vezérlőelem ilyen lehetőségekkel is rendelkezik. (Be is szúrhatunk újabb elemet, illetve meghatározhatjuk a kiíratási sorrendet stb.)

A listadobozba többféle módon kerülhetnek az elemek. Avval, hogy adatbeolvasáskor a listadoboz `Items` listájába raktuk be az adatok alapján létrehozott példányokat, azok azonnal meg is jelennek.

Az adatbeolvasás után célszerű inaktívvá tenni a gombot, hiszen többé már nem akarunk beolvasni, ugyanekkor aktívvá kell tennünk a győztes megjelenítésére szolgáló nyomógombot.

Az „Adatok beolvasása” feliratú gombnyomás hatása:

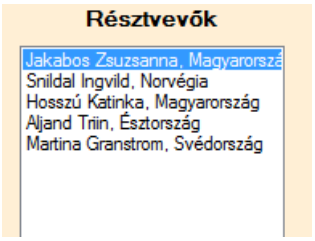
```
private void btnAdatBe_Click(object sender, EventArgs e) {
    lstVersenyzok.Items.Clear();
    AdatBeolvasas();
    btnAdatBe.Enabled = false;
    btnGyoztes.Enabled = true;
}
```

Hogy pontosabbak legyünk, a listadobozban nem a lista elemei jelennek meg, hanem azok `ToString()` reprezentációja.

Ahhoz tehát, hogy olvasható formában láthassuk a versenyzők adatait, meg kellett írnunk a `Versenyzo` osztály `ToString()` metódusát.

Megjegyzés:

Próbálja ki különböző `ToString()` metódusokkal is, és érje el az alábbi külalakokat:



A listadobozhoz tartozó alapértelmezett esemény a listából kiválasztott elem indexének változása (vagyis akkor következik be, ha egy másik indexű elemet választunk). Mi is ezt használjuk most ahhoz, hogy ki tudjuk íratni a kiválasztott versenyző adatait.

Az elem(ek) kiválasztására a listadoboz Selected… tulajdonságai alkalmazhatók: `SelectedIndex`, `SelectedIndices`, `SelectedItem`, `SelectedItems`.

A kiválasztott index alapján így írathatjuk ki egy versenyző adatait:

```
private void lstVersenyzok_SelectedIndexChanged(object sender, EventArgs e) {
    Versenyzo versenyzo = versenyzok[lstVersenyzok.SelectedIndex];
    txtRajtszam.Text = versenyzo.Rajtszam;
    txtOrszag.Text = versenyzo.Orszag;
    txtIdoEredmeny.Text = new DateTime(versenyzo.IdoEredmeny.Ticks).ToString("mm:ss.ff");
    txtEletKor.Text = (DateTime.Now.Year - versenyzo.SzulDatum.Year) + " év";
}
```

A feladat közvetlenül az elem kiválasztásával is megoldható. A listadobozból azonban csak `Object` típusú elemet tudunk kiválasztani, ezért típuskényszerítést kell alkalmaznunk. De vigyázat! A típuskényszerítés mindig veszélyes, ha ugyanis nem olyan típusra akarjuk kényszeríteni, amilyen ténylegesen, akkor elszáll az egész program!

Az ilyen elvű megoldás:

```
private void lstVersenyzok_SelectedIndexChanged(object sender, EventArgs e) {
    Versenyzo versenyzo = (Versenyzo)lstVersenyzok.SelectedItem;
    txtRajtszam.Text = versenyzo.Rajtszam;
    txtOrszag.Text = versenyzo.Orszag;
    txtIdoEredmeny.Text = new DateTime(versenyzo.IdoEredmeny.Ticks).ToString("mm:ss.ff");
    txtEletKor.Text = (DateTime.Now.Year - versenyzo.SzulDatum.Year) + " év";
}
```

A típuskényszerítés így is írható:

```
private void lstVersenyzok_SelectedIndexChanged(object sender, EventArgs e) {
    Versenyzo versenyzo = lstVersenyzok.SelectedItem as Versenyzo;
    txtRajtszam.Text = versenyzo.Rajtszam;
    txtOrszag.Text = versenyzo.Orszag;
    txtIdoEredmeny.Text = new DateTime(versenyzo.IdoEredmeny.Ticks).ToString("mm:ss.ff");
    txtEletKor.Text = (DateTime.Now.Year - versenyzo.SzulDatum.Year) + " év";
}
```

Mindhárom megoldás jó, bármelyiket választhatja.

A győztes időt pontosan ugyanúgy határozhatjuk meg, mint konzolos alkalmazás esetén. Az összehasonlítandó időeredmények `TimeSpan` típusú objektumok. Objektumokat elvileg az objektumot definiáló osztályban megírt `CompareTo()` metódussal tudunk összehasonlítani (elvileg `DateTime` objektumokat is), de a dátumot, időt számmá alakítva kezeli a C#, ezért ezeket a szokásos kisebb/nagyobb relációkkal is összehasonlíthatjuk.

Az eltérés most is a megjelenítésben van. Ha konzolos alkalmazást írnánk, akkor a győztes időeredmény kiszámítása után végig kellene mennünk a versenyzők listájának adatain, és kiírni mindenkit, akinek időeredménye azonos a győztes idő értékével. Ugyanezt tesszük itt is, csak az adatokat nem konzolra írjuk, hanem az `rchTxtGyoztesek` nevű, `RichTextBox` típusú szövegdobozba.

Tervezési fázisban ezt a szövegdobozt is csak olvashatóvá tesszük, és ha fehér hátteret szeretnénk, akkor azt is beállítjuk.

Mivel a listadobozhoz tartozó `Items` lista `Object` típusú elemeket tartalmaz, ezért most is típuskényszerítést kell alkalmaznunk.

A „Győztes” feliratú gomb hatása:

```
private void btnGyoztes_Click(object sender, EventArgs e) {
    TimeSpan min = (lstVersenyzok.Items[0] as Versenyzo).IdoEredmeny;
    foreach (var item in lstVersenyzok.Items) {
        if ((item as Versenyzo).IdoEredmeny < min) {
            min = (item as Versenyzo).IdoEredmeny;
        }
    }
    txtGyoztesIdo.Text = new DateTime(min.Ticks).ToString("mm:ss:ff");
    rchTxtGyoztes.Clear();
    foreach (var item in lstVersenyzok.Items) {
        if ((item as Versenyzo).IdoEredmeny == min) {
            rchTxtGyoztes.AppendText(item + "\n");
        }
    }
}
```

Megjegyzések:

1. Ha a listadobozban névsorba rendezve akarjuk megjeleníteni az adatokat, akkor a `Sorted` tulajdonságát `true`-ra kell állítani. De vigyázat, ez a `ToString()`-ben megadott `string`-nek megfelelően rendez, és nem feltétlenül a nevek alapján.

2. Ha eredmény alapján szeretnénk rendezni a versenyzőket, akkor valószínűleg kényelmesebb előbb egy `Versenyzo` típusú elemekből álló listába átrakni a versenyző példányokat, ezt a listát rendezni, majd rendezés után visszaírni őket a listadobozba. De arra mindenképpen figyelnünk kell, hogy ha a listadoboz `Sorted` tulajdonsága `true`-ra van állítva, akkor hiába rendezzük a versenyzőket, a kiíratásra mégis a `ToString()`-nek megfelelő sorrend lesz mérvadó.

2. A leggyakrabban használt komponensek

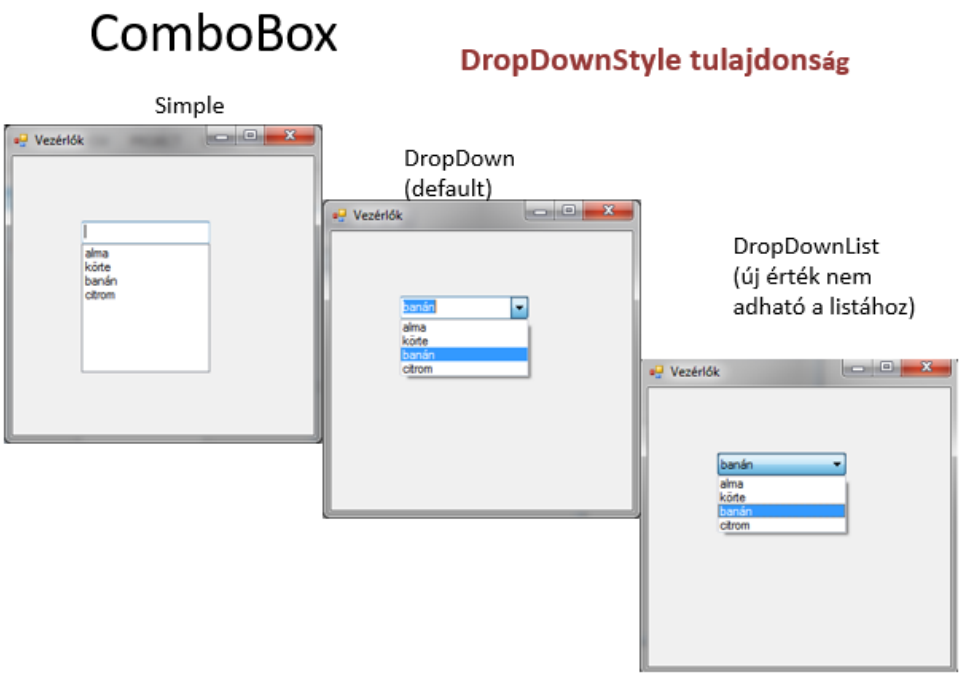
2.1. Elméleti háttér *(Szendrői E.)*

Az előző fejezetben megismerkedtünk néhány vezérlővel. Ebben a fejezetben a választást támogató vezérlőkomponensekkel ismerkedünk meg.

ComboBox vezérlő

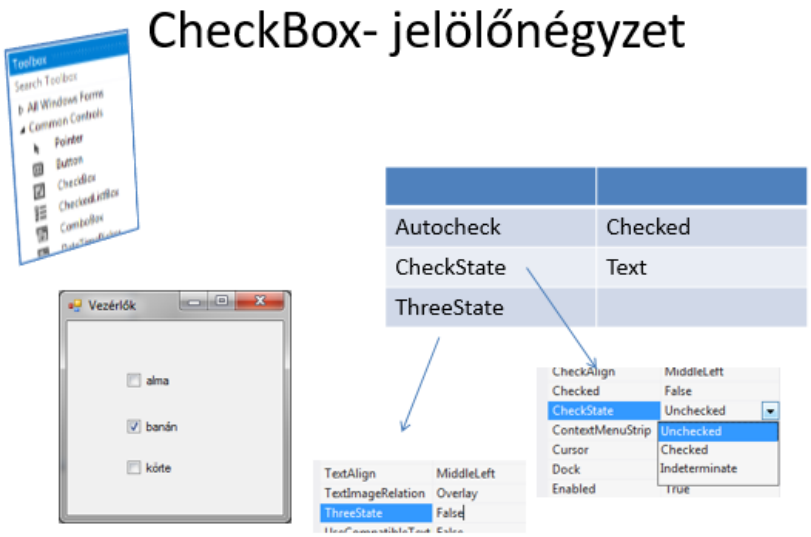
A ComboBox egy szövegdoboz és egy listamező kombinációja. A ListBox osztályból származik. Annyiban más, mint az őse, hogy nemcsak kiválasztást tesz lehetővé, hanem a vezérlő szövegdoboz részében új elemeket is megadhatunk, ha az engedélyezett, növelve a választékot.

A ComboBoxnak is számos publikus tulajdonsága van, ebből az egyik fontos a **DropDownStyle**, amely a legördülő lista stílusát állítja be. Az alábbi képen a különböző stílusbeállítások láthatók. Simple érték esetén nincs legördülő rész, csak a listát látjuk. A default, alap esetben legördíthető a lista kínálata és a szövegmező részbe beírt új elemmel bővíthető a lista. A *DropDownList* tulajdonságérték letiltja a listabővítési lehetőségét, csak a legördítés lehetséges. A többi tulajdonsága a ListBoxnál már megismert *SelectedIndex*, *SelectedItem*, *Items* stb. A *DropDownHeight* tulajdonsággal megadhatjuk a legördülő lista magasságát.



Választóvezérlők – CheckBox

A CheckBox vezérlő egy érték beállítására alkalmas. Rendszerint valamilyen logikai érték megjelenítésére használjuk. A formra több checkBox objektum is elhelyezhető. Legfontosabb tulajdonsága a Checked, amely megmutatja, hogy be van-e állítva vagy sem. A Text tulajdonsága a checkBoxhoz tartozó feliratot tartalmazza.



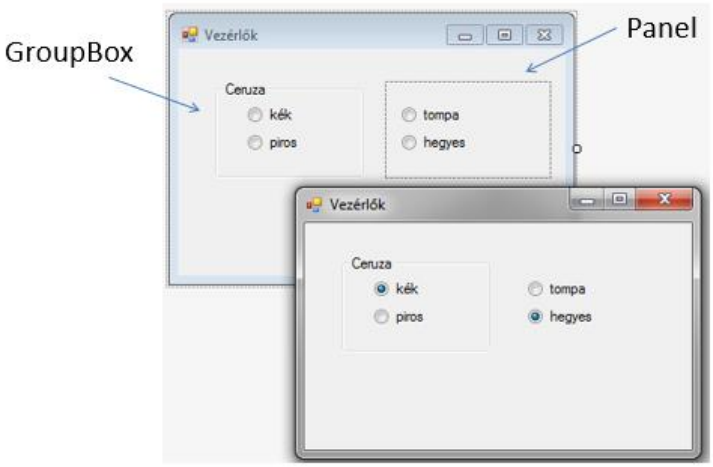
A CheckState tulajdonsággal három állapot beállítására van lehetőség Checked, Unchecked és Intermediate, ha a ThreeState tulajdonság értéke True, vagyis megengedett a három állapot. Legfontosabb eseménye a **CheckedChanged** esemény.

A RadioButton vezérlő

Általában több RadioButton vezérlőt helyezünk a formra, ez több felkínált lehetőség közötti választást tesz lehetővé. Csak egy RadioButtont lehet választani. Az alábbi képen látható a működése. Legfontosabb tulajdonsága a Text, ami a felirata és a Checked, ami jelzi, hogy ki van választva.

Fontos eseménye a CheckedChanged esemény, amely jelzi, hogy megváltozott a kijelölése.

Abban az esetben, ha többféle választékból kell egyet-egyet kiválasztanunk, akkor csoportosítani kell a RadioButton vezérlőket, és valamilyen konténer típusú vezérlőre kell felhelyeznünk. Ilyen konténer típusú vezérlő a formon kívül a **Panel** és a **Group** vezérlő.



A Group vezérlőnek adhatunk feliratot, Text tulajdonság, s ez futás közben látható. A Group vezérlőnek nincs AutoScroll tulajdonsága, tehát nem gördíthető. A Panel vezérlőnek nem adhatunk feliratot, nincs Text tulajdonsága, viszont a BorderStyle, keretstílust beállíthatjuk, ami futás közben látszik. A Panel vezérlőn görgetősávot be lehet kapcsolni.

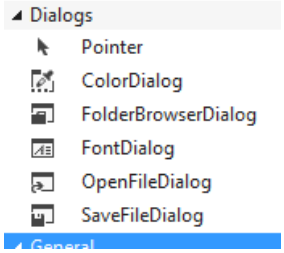
Button (nyomógombvezérlő)

A leggyakrabban használt vezérlőelem. A Control osztály minden tulajdonságát megörökli. A Text tulajdonsága a gomb feliratát tartalmazza. Előtér, háttérszíne beállítható. Az Image tulajdonsága segítségével kép is elhelyezhető rajta. Legfontosabb eseménye a Click esemény.

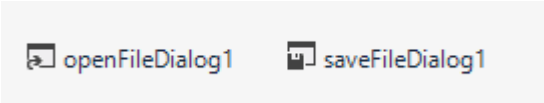
Párbeszéd, dialógusablakok

A .NET környezetben több beépített dialógusablak van, amelyek megkönnyítik a fejlesztők munkáját. A fájlok kezelésénél láttuk, hogy meg kell adnunk a fájl nevét, és ha nem a projekt Bin/Debug mappájában helyeztük el a fájlt, akkor az elérési utat is meg kellett adnunk. Sokkal könnyebb a falhasználó dolga, ha a fájlokat és mappákat egy párbeszédablakban adhatja meg.

A dialógusablakok a ToolBox Dialogs csoportjában találhatók meg, ahogy a kép mutatja.



Amikor kiválasztunk egy dialógusablakot és ráhúzzuk a formra, az a komponens tálcára kerül, nem látszik a formon.



OpenFileDialog és SaveFileDialog

Amikor a programból meg akarunk hívni egy fájl Megnyitás vagy Mentés ablakot, akkor előbb egy objektumpéldányt kell készíteni belőle.

Az objektum létrehozása a konstruktorhívással történik:

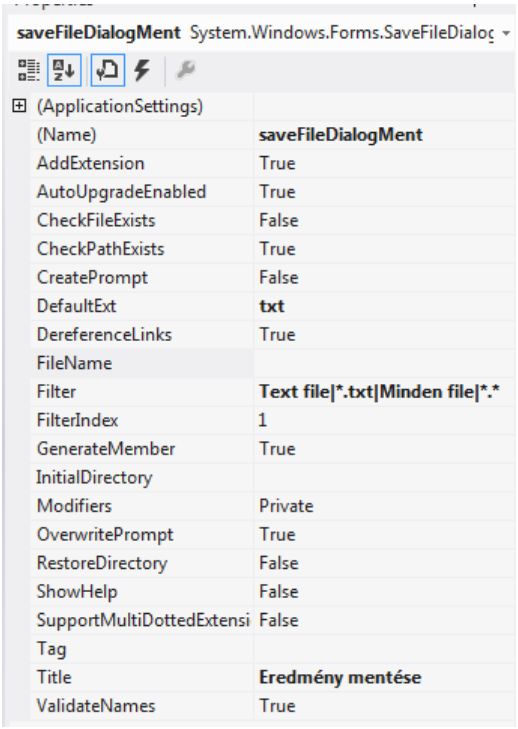
OpenFileDialog *változónév* ***= new OpenFileDialog*** *();*

SaveFileDialog *változónév* ***= new SaveFileDialog*** *();*

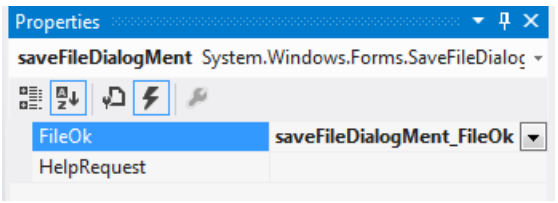
Az alábbiakban az OpenFileDialog és SaveFile Dialog legfontosabb tulajdonságait gyűjtöttük össze:

AddExtension	***true*** – ha nem adnak meg kiterjesztést, a **DefaultExt** tulajdonságban megadott kiterjesztéssel kezeli a fájlt, ***false*** – nincs automatikus kiterjesztéskezelés
DefaultExt	alapértelmezett fájlkiterjesztés
FileName	a kiválasztott fájl neve
Filter	szűrő a fájltípusok megadására, az elemeket a pipe (cső) szimbólummal (| -vel) elválasztva
InitialDirectory	kezdeti könyvtár beállítása
Title	a fejléc szövege
ValidateNames	***true*** – ellenőrzi, csak érvényes karakterek szerepelnek-e a fájlnévben ***false*** – nincs ellenőrzés.

Az alábbi képen a SaveFile Dialog tulajdonságai láthatók



Legfontosabb eseménye a FileOK esemény, amely akkor következik be, amikor a felhasználó az Open vagy a Save gombra kattint.



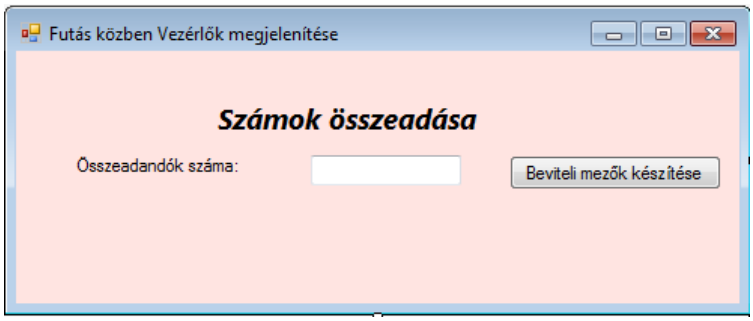
A párbeszédablakok a **ShowDialog**() metódus hívásával jeleníthetők meg, vagyis modális ablakként fut. A metódushívás visszatérési értéke a *DialogResult,* amely megadja, milyen nyomógombot választottunk a párbeszéd lezárásához.

Vezérlők dinamikus kezelése

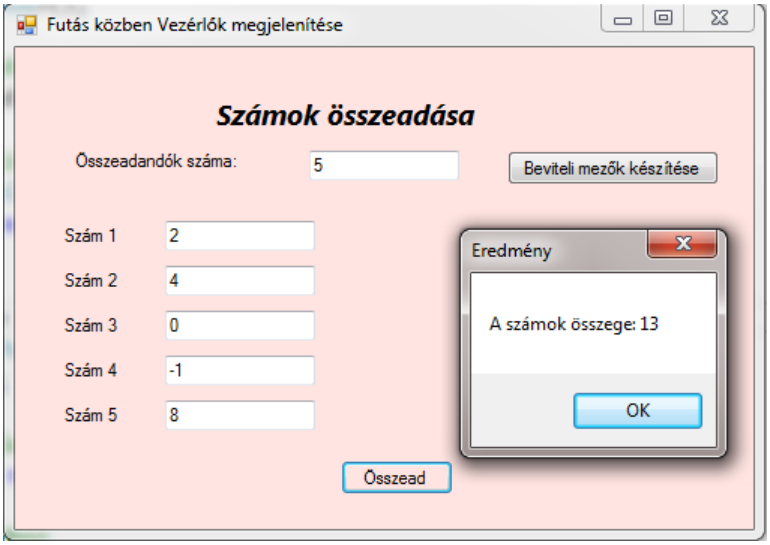
A következő egyszerű példában bemutatjuk, hogyan készíthető olyan alkalmazás, ahol futás közben tesszük a formra a vezérlőelemeket.

Készítsünk egy programot, amelyben a felhasználó megadja az összeadandók számát. A nyomógombra kattintást követően felkerül annyi szövegdoboz az űrlapra, ahány értéket

össze szeretne adni a felhasználó. A **Számol** gombra kattintáskor megjelenik az eredmény is. Az induló felület a következő:



Egy futási eredmény a következő ábrán látható



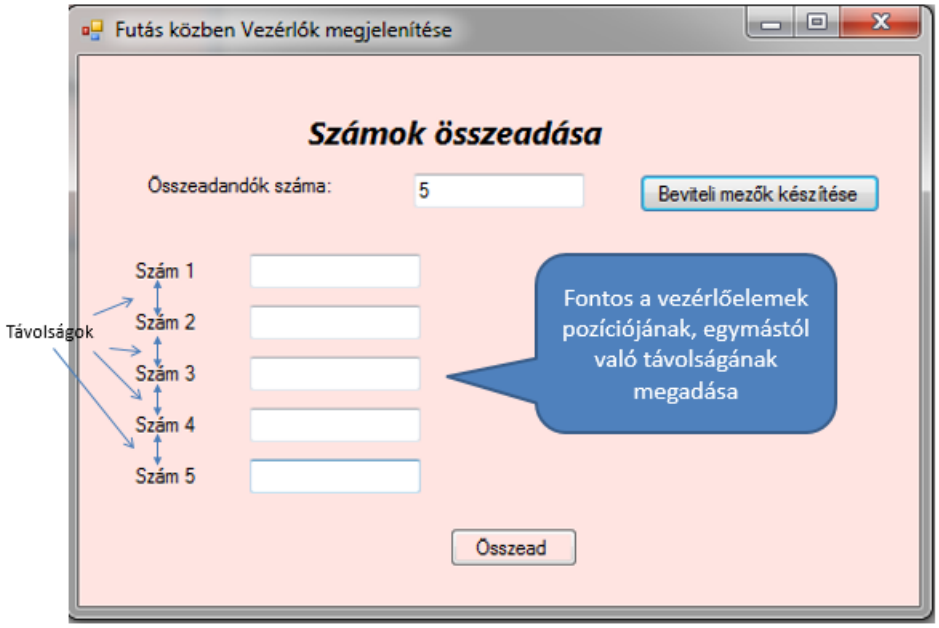
A Form kezdeti tulajdonságait a konstruktorban állítjuk be, így az átméretezhetőség engedélyezését, a kezdőpozíciót. Osztályszinten deklarálunk egy TextBox típusú listát az összeadandó számok beviteléhez.

```
public partial class Form1 : Form
{
    public Form1() {
        InitializeComponent();

        this.AutoSizeMode = AutoSizeMode.GrowAndShrink;
        this.AutoSize = true;
        this.Padding = new Padding(0, 0, 20, 20); // az ablak jobb oldalától
        // és aljától való távolság
        this.StartPosition = FormStartPosition.CenterScreen;
    }
    // Az adatbeviteli szövegdobozok listája
    private List<TextBox> inputTextBoxes;
```

A következő kódrészletben ellenőrizzük, hogy helyesen adta-e meg a felhasználó az összeadandók számát. Az ellenőrzés while ciklusban történik, s mindaddig nem lépünk ki a ciklusból, amíg helyes értéket nem kapunk.

```
private void btnFelrak_Click(object sender, EventArgs e) {
    //Bekérjük a szövegdobozok számát a generáláshoz
    int beSzamDarab;
    while (!int.TryParse(txtOsszadandokSzama.Text, out beSzamDarab) || beSzamDarab<=0)
    {
        MessageBox.Show("Pozitív egész számot adjon meg!", "Hiba");
        txtOsszadandokSzama.Focus();
        txtOsszadandokSzama.Clear();
        return;
    }
```

Fontos kérdés a Vezérlőelemek megfelelő pozicionálása. Pontosan egymás alatt, megfelelő függőleges távolságban, az ablak bal szélétől ugyanolyan távolságban kell elhelyezni őket futásidőben.



A vezérlőelemek futásidőben történő formra helyezését az alábbi kódrészletben követhetjük nyomon.

Az ábrán a kommentekből jól követhető, hogy mit csinál a program. Először a konstruktor meghívásával létrehozzuk a szövegdobozokat tartalmazó listaobjektumot. Ezt követően egy for ciklusban létrehozunk annyi címkét és szövegdobozt, amennyi a felhasználó által megadott összeadandók száma. Beállítjuk a tulajdonságaikat, különös tekintettel a Location tulajdonságra. Majd hozzáadjuk a TextBox vezérlőelemeket a textboxok listájához és azután a textboxokat és a címkéket hozzáadjuk a form vezérlőelemeinek Controls gyűjteményéhez.

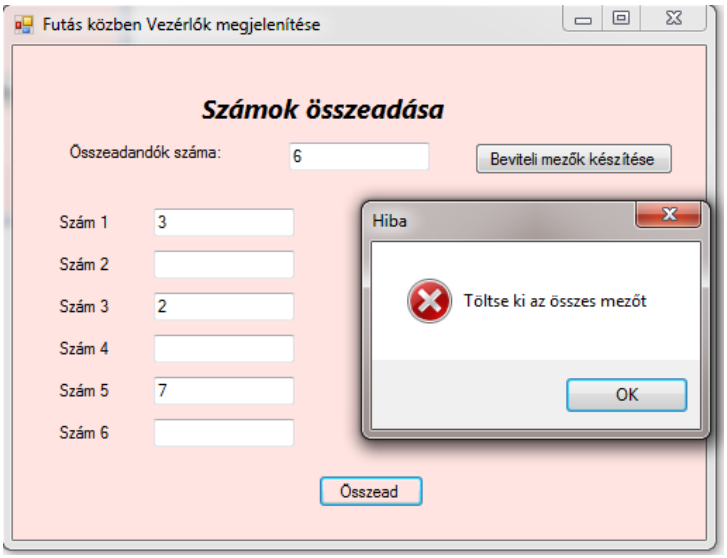
btnFelrak_Click
folytatás

```
//a szövegdobozok listáját létrehozzuk és inicializáljuk
inputTextBoxes = new List<TextBox>();

//Létrehozzuk a feliratokat (label) és a szövegdobozokat
for (int i = 1; i <= beSzamDarab; i++)
{
    //Új címke és szövegdoboz létrehozása
    Label labelInput = new Label();
    TextBox textBoxNewInput = new TextBox();

    //Címkék tulajdonságai
    labelInput.Text = "Szám " + i;
    labelInput.Location = new Point(30, txtOsszadandokSzama.Bottom + (i * 30));
    labelInput.AutoSize = true;

    //Szövegdobozok kezdeti tulajdonságai
    textBoxNewInput.Location = new Point(labelInput.Width, labelInput.Top - 3);

    //TexBox hozzáadása a listához
    inputTextBoxes.Add(textBoxNewInput);

    //A címke és a szövegdoboz hozzáadása a Form-hoz
    this.Controls.Add(labelInput);
    this.Controls.Add(textBoxNewInput);
}
```

Ezt követően már csak a nyomógomb objektumot kell létrehoznunk a megfelelő tulajdonságértékekkel, majd őt is hozzá kell adni a form Controls gyűjteményéhez.

```
        }  // for ciklus vége
        //Összead Nyomógomb létrehozása
        Button btnOsszead = new Button();
        //Tulajdonságok
        btnOsszead.Text = "Összead";
        //Tegyük a gombot a form közepére a legutolsó szövegdoboz alá
        btnOsszead.Location = new Point(this.Width / 2 - btnOsszead.Width / 2,
            inputTextBoxes[inputTextBoxes.Count - 1].Bottom + 20);
        //Adjunk egy eseménykezelőt a gomb Click eseményéhez
        btnOsszead.Click += new EventHandler(btnOsszead_Click);

        //Adjuk a gombot a Form-hoz
        this.Controls.Add(btnOsszead);

        //Tegyük a formot a képernyő közepére
        this.CenterToScreen();

    // btnFelrak_Click vége
    }
```

Természetesen célszerű ellenőrizni, hogy a felrakott szövegdobozok mindegyikébe megadta-e a felhasználó az összeadandó értékét, vagy sem. Ha hiba van, erről értesítést kap a felhasználó.



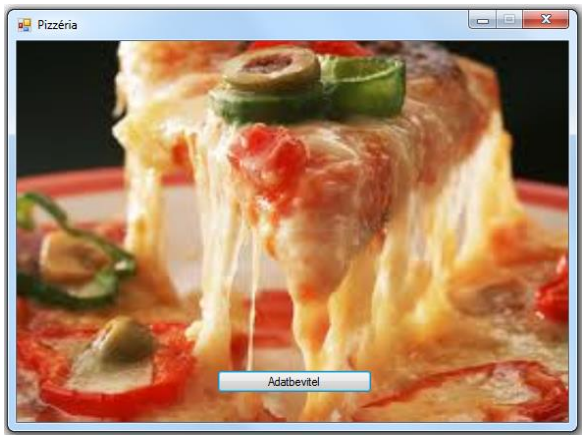
Az Összead nyomógomb eseménykezelője végzi az adatok ellenőrzését és az eredmény kiszámítását is, melyet egy MessageBoxban jelenít meg.

```
private void btnOsszead_Click(object sender, EventArgs e) {
    int osszeg = 0;

    foreach (TextBox inputBox in inputTextBoxes)
    {
        if (inputBox.Text == String.Empty)
        {
            MessageBox.Show("Töltse ki az összes mezőt", "Hiba",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        else
        {
            osszeg += Int32.Parse(inputBox.Text);
        }
    }
    MessageBox.Show("A számok összege: " + osszeg,"Eredmény");
}
```

2.2. Gyakorlati példa *(Achs Á.)*

2.2.1. Pizzéria

Célunk egy olyan alkalmazás elkészítése, amelynek segítségével pizzákat lehet rendelni, és kiszámolja a megrendelés árát.

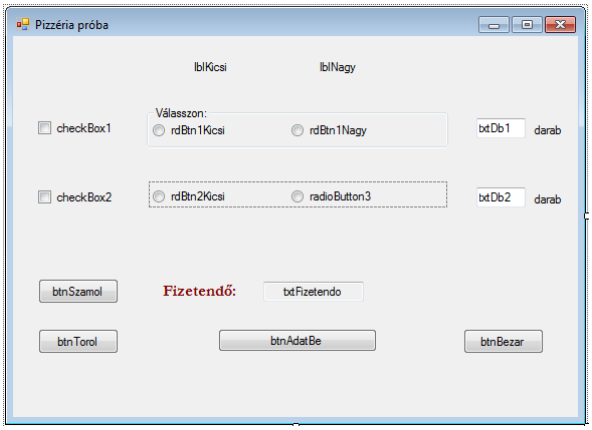


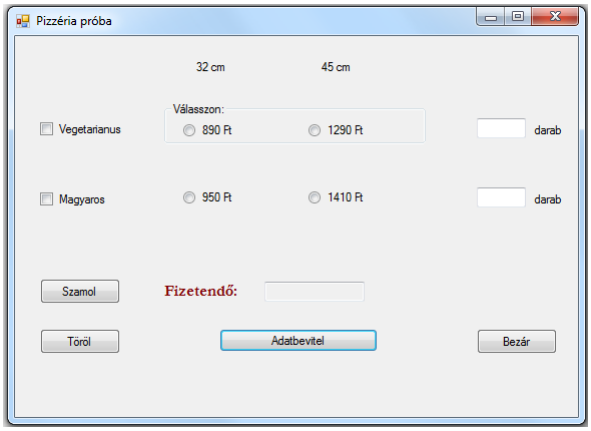
Ha már programot írunk, akkor nyilván azt szeretnénk, hogy esetleg több pizzéria is használhassa, és természetesen nem csak egyetlen napon. Mivel sem azt nem tudjuk, hogy hányféle pizzát árul az adott pizzéria, és azt sem, hogy ezeket milyen áron és hányféle méretben, ezért az adatokat fájlból olvassuk be, emiatt magukat a vezérlőelemeket is a programkódnak kell felraknia a felületre.

Mielőtt azonban megoldanánk a tényleges feladatot, szerezzünk tapasztalatokat egy egyszerűsített változat elkészítésével.

Egyszerűsített (előkészítő) feladat:

Az adatfájl most összesen kétféle pizza adatait tárolja, és mindegyikhez kétféle ár tartozik (kicsi és nagyméretű pizza ára). Ezeket a fixen felrakott vezérlőelemek kialakításához használjuk fel. A fájlszerkezet: soronként a pizza neve; a kisebbik méret ára; a nagyobbik méret ára. Az induló és az adatbevitel hatására kapott felület:





Hozza létre az induló felületet! A gombok felirata legyen a futási ábrán látható szöveg, a szövegmezők legyenek üresek, és lehetőleg az induló felületen látható változóneveket használja, a közölt megoldás ugyanis ezeket fogja. Két rádiógombot fogjon össze egy panelen, másik kettőt pedig egy GroupBox elem segítségével.

Az **Adatbevitel** nyomógombra kattintva olvashatjuk be a fájl tartalmát. A beolvasáshoz használjunk **OpenFileDialog** ablakot.

A **Számol** gomb hatására a fizetendő összeg megjelenik a csak olvasható szövegmezőben.

A számolás során végezzünk ellenőrzéseket (van-e kiválasztott pizza, megadtuk-e a szükséges méretet, darabszámot, tényleg pozitív egész számot gépeltünk-e).

A **Töröl** gomb az összes szövegmező tartalmát törli, és megszünteti a kijelöléseket.

Egy lehetséges megoldás

Az induló felületre fel kell rakni két `checkBox`-ot, egy `panel`-t, rajta két `radioButton`-t, illetve egy `groupBox`-ot, bele szintén két `radioButton`-t. (Természetesen most csak a kipróbálás kedvéért van panel is és csoportdoboz is, „élesben” csak egyfélét kellene kirakni.)

Ezeken kívül még fel kell rakni egy `openFileDialog` vezérlőelemet, ami nem látszik az induló felületen, mert egy dialógusablak, amely csak egy esemény hatására nyílik meg, de létre kell hozni, különben nem tudunk hivatkozni rá. (Ez is „rádobható” a formra, de a form alatt jelenik meg az ikonja.)

A projekt pizzákkal foglalkozik, vagyis meg kell írnunk a `Pizza` nevű osztályt is. Bár elvileg az is változhat, hogy hányféle méretben készítenek egy-egy pizzafajtát, az egyszerűség kedvéért a továbbiakban feltételezzük, hogy csak kétféle méret létezik.

```
class Pizza {
    public string Nev { get; private set; }
    public int ArKicsi { get; set; }
    public int ArNagy { get; set; }

    public Pizza(string nev, int arKicsi, int arNagy) {
        this.Nev = nev;
        this.ArKicsi = arKicsi;
        this.ArNagy = arNagy;
    }

    public override string ToString() {
        return this.Nev;
    }
}
```

A `Form1` osztály kommentekkel ellátott kódja:

```
public partial class Form1 : Form {

    // konstruktor
    public Form1() {
        // A generált inicializálás.
        // Tartalma megnézhető a Form1.Designer.cs fájlban.

        InitializeComponent();

        // Beállítjuk a fájldialógus aktuális mappáját (ez jelenik meg megnyitáskor)
        // és a keresés során felkínált fájlnevet.

        openFileDialog1.InitialDirectory = Environment.CurrentDirectory;
        openFileDialog1.FileName = "";
    }

    private int meretKicsi = 32, meretNagy = 45;
    private List<Pizza> pizzak = new List<Pizza>();

    /// <summary>
    ///
    /// Az Adatbevitel gomb hatására meghívja az AdatBevitel() metódust.
    /// Ha az sikeresen lefut, akkor meghívja az ElemekMegjelenitese() metódust.
    /// Ha valahol hiba keletkezett, azt a catch ág elkapja, és kiírja a
    /// keletkezett hibafajtához tartozó üzenetet is.
    /// (Megjegyzés: az ilyen üzenetek főleg a fejlesztőnek nyújtanak információt,
    /// a felhasználó számára néha nem árt átfogalmazni őket.)
    ///
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void btnAdatBe_Click(object sender, EventArgs e) {
        try {
            AdatBevitel();
            ElemekMegjelenitese();
        }
        catch (Exception ex) {
            MessageBox.Show(ex.Message, "Hiba");
        }
    }
```

```
/// <summary>
///
/// Megnyitja a fájl-dialógus ablakot.
/// Ha megnyomtuk az OK gombot, akkor lekéri
/// a választott fájl nevét, és evvel a névvel
/// meghívja az AdatBeolvasas() metódust.
///
/// Itt több hiba is keletkezhet:
/// 1. Nincs is ilyen nevű fájl (vagy nem választottunk semmit).
/// Ezt is ki kellene védenünk, de szerencsére az
/// openFileDialog1 alapbeállítása az, hogy ezt ellenőrzi,
/// ezért nem kell megírnunk.
///
/// 2. A fájl olvasásakor is lehet hiba, ezt kezeli a
/// try-catch blokk.
///
/// </summary>
private void AdatBevitel() {
    DialogResult result = openFileDialog1.ShowDialog();
    if (result == DialogResult.OK) {
        try {
            string fajlNev = openFileDialog1.FileName;
            AdatBeolvasas(fajlNev);
        }
        catch (Exception) {
            MessageBox.Show("Hiba a fájl beolvasásakor", "Hiba");
        }
    }
}

/// <summary>
/// A megadott nevű fájlból soronként beolvassa az adatokat,
/// és a Feldolgoz() metódussal feldolgoztat egy-egy sort.
///
/// </summary>
/// <param name="fajlNev"></param>

private void AdatBeolvasas(string fajlNev) {
    string adat;
    StreamReader olvasoCsatorna = new StreamReader(fajlNev);

    while (!olvasoCsatorna.EndOfStream) {
        adat = olvasoCsatorna.ReadLine();
        Feldolgoz(adat);
    }
    olvasoCsatorna.Close();
}

/// <summary>
///
/// A beolvasott sort felvágja a ';' mentén, és az így kapott
/// adatok alapján létrehoz egy Pizza típusú példányt, és
/// hozzáadja a pizzák listájához.
///
/// </summary>
/// <param name="adat"></param>
private void Feldolgoz(string adat) {
   string[] adatok = adat.Split(';');
   pizzak.Add(new Pizza(adatok[0],int.Parse(adatok[1]), int.Parse(adatok[2])));
}
```

```
/// <summary>
///
/// A két labelre kiírja a pizzák méretét,
/// majd az egyes vezérlőelemek szövegeként
/// megadja az egyes pizzák adatait.
///
/// A feladatot most "gyalog" oldottuk meg, pont azért,
/// hogy tapasztalatokat szerezzen, de nyilván nem
/// az a megoldás, hogy copy-paste segítségével programozunk,
/// hanem az, hogy ciklust szervezünk az elemek megjelenítésére.
///
/// </summary>
private void ElemekMegjelenitese() {
    lblKicsi.Text = meretKicsi + " cm";
    lblNagy.Text = meretNagy + " cm";

    checkBox1.Text = pizzak[0].Nev;
    rdBtn1Kicsi.Text = pizzak[0].ArKicsi + " Ft";
    rdBtn1Nagy.Text = pizzak[0].ArNagy + " Ft";

    checkBox2.Text = pizzak[1].Nev;
    rdBtn2Kicsi.Text = pizzak[1].ArKicsi + " Ft";
    rdBtn2Nagy.Text = pizzak[1].ArNagy + " Ft";
}
```

```
/// <summary>
///
/// A metódus nagyon "gyalog" és nagyon "fapados", egy rendes programban
/// nyilván nem így kell csinálni, hanem ezt is ciklusban.
/// De arra jó, hogy tapasztalatokat gyűjthessen.
/// Gondolja végig, hogy mikor milyen kivétel keletkezhet, milyen
/// hibalehetőségeket kell kivédeni.
///
/// A metódus egyébként helyes adatmegadás esetén kiszámolja és kiírja
/// a választott pizzákért fizetendő összeget.
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnSzamol_Click(object sender, EventArgs e) {
    try {
        int osszeg = 0, ar = 0, db = 0;

        // Ha az első pizzát választottuk
        if (checkBox1.Checked) {
                // ha az első fajtát választottuk
            if (rdBtn1Kicsi.Checked) ar = pizzak[0].ArKicsi;
                // ha a második fajtát választottuk
            else if (rdBtn1Nagy.Checked) ar = pizzak[0].ArNagy;
                // ha egyáltalán nem választottunk
            else throw new MissingFieldException("Válasszon méretet");

            // ide már csak akkor jut, ha tényleg választottunk, ezért
            // kiolvashatja a db értékét. Ha hibás formátumot adunk meg
            // (üres vagy nem egész szám), akkor FormatException kivételt dob.
            db = int.Parse(txtDb1.Text);

            // azt is vizsgálni kell, hogy nem írtunk-e negatív értéket.
            if (db < 0) throw new ArgumentOutOfRangeException("Nem lehet negatív");
            osszeg += ar * db;
        }

        // Ha a második pizzát választottuk
        if (checkBox2.Checked) {
            if (rdBtn2Kicsi.Checked) ar = pizzak[1].ArKicsi;
            else if (rdBtn2Nagy.Checked) ar = pizzak[1].ArNagy;
            else throw new MissingFieldException("Válasszon méretet");
            db = int.Parse(txtDb2.Text);
            if (db < 0) throw new ArgumentOutOfRangeException("Nem lehet negatív");
            osszeg += ar * db;
        }

       // Ha egyiket sem választottuk
       // Figyelje meg, hogy mi a különbség a kikommentezett és a másik
       // kivételdobás között.
       /*if (!checkBox1.Checked && !checkBox2.Checked)
               throw new MissingFieldException("Válasszon pizzát"); */
       if (!checkBox1.Checked && !checkBox2.Checked) throw new MissingFieldException();

       // Kiíratjuk a fizetendő összeget.
       txtFizetendo.Text = osszeg + " Ft";
   }
```

```
            // A kivételek elkapása.
            // Figyelje meg, hogy mi a különbség, ha saját üzenetet írunk,
            // vagy a keletkezett kivétel üzenetét.
        catch (FormatException) {
            MessageBox.Show("Hibásan adta meg a darabszámot", "Hiba");
        }
        catch (ArgumentOutOfRangeException aox) {
            MessageBox.Show(aox.Message, "Hiba");
        }
        catch (MissingFieldException mex) {
            MessageBox.Show(mex.Message, "Hiba");
        }
    }

    /// <summary>
    ///
    /// Kitörli a kijelöléseket, és hogy legyen némi programozás is
    /// a megoldásban, ezt most nem "gyalog" teszi. :))
    ///
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void btnTorol_Click(object sender, EventArgs e) {
        // Végimegyünk a felületre tett vezérlőelemeken
        foreach (Control item in this.Controls) {
            // Töröljük a checkBox-ok kijelölését
            if (item is CheckBox) (item as CheckBox).Checked = false;
            // Töröljük a textBox-ok kijelölését
            // csak azért szerepel kétféle típuskényszerítés, hogy mindkettőre lásson példát
            if (item is TextBox) ((TextBox)item).Clear();

            // A rádiógombok nem közvetlenül a form-on vannak, ezért a panel,
            // ill. groupBox gyerekeit kell vizsgálnunk.
            if (item.HasChildren) {
                foreach (Control gyerek in item.Controls) {
                    if (gyerek is RadioButton) (gyerek as RadioButton).Checked = false;
                }
            }
        }
    }
}
```

FONTOS megjegyzés:

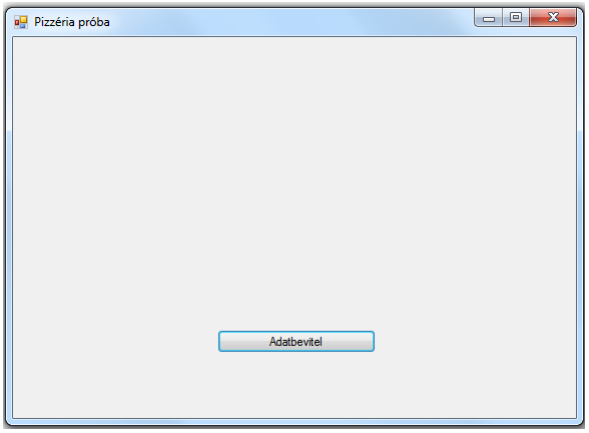
Ha a fájl kiválasztását az `openFileDialog1` vezérlőelemre bízzuk, és elfogadjuk annak default beállítását, akkor az `AdatBeolvasas(fajlNev)` metódushívásra már csak helyesen (azaz megnyitható módon) megadott fájlnév esetén kerülhet sor. Ugyanakkor maga az adatfájl lehet hibás, emiatt keletkezhet hiba a beolvasáskor. Evvel nincs is baj, hiszen éppen ezért írtuk `try-catch` blokkba. Viszont van még egy probléma: Ha valami miatt hibásan zárul az adatbevitel, akkor nem záródik le az olvasócsatorna. Ez nem égbekiáltó hiba, de mégis az a tisztességes megoldás, ha gondoskodunk róla, hogy hiba esetén is bezáródjon. A korrekt megoldás tehát az, ha az olvasócsatornát is a `try` blokkon belül nyitjuk meg, és a `finally` ágon zárjuk be. (A `finally` ág akkor is végrehajtódik, ha hiba nélkül fut le a `try` blokk, de akkor is, ha a `catch` ágra kerül a vezérlés.)

Az ilyen elvű megoldás:

```
/// <summary>
/// Ha megnyitotta az olvasócsatornát (vagyis nem null),
/// akkor akár van hiba az olvasásban, akár nincs, le fogja zárni
/// az olvasócsatornát.
/// </summary>
private void AdatBevitel() {
    if (openFileDialog1.ShowDialog() == DialogResult.OK) {
        StreamReader olvasoCsatorna = null;
        try {
            String fajlNev = openFileDialog1.FileName;
            olvasoCsatorna = new StreamReader(fajlNev);
            AdatBeolvasas(olvasoCsatorna);
        }
        catch (Exception e) {
            MessageBox.Show("Adatbeviteli hiba", "Hiba");
        }
        finally {
            if (olvasoCsatorna != null) {
                olvasoCsatorna.Close();
            }
        }
    }
}

/// <summary>
/// A paraméterben adott olvasócsatornán keresztül olvas
/// </summary>
/// <param name="olvasoCsatorna"></param>
private void AdatBeolvasas(StreamReader olvasoCsatorna) {
    string adat;

    while (!olvasoCsatorna.EndOfStream) {
        adat = olvasoCsatorna.ReadLine();
        Feldolgoz(adat);
    }
}
```

Még egy előkészítés:

Oldjuk meg, hogy ilyen felületről induljon ez előző „pizzarendelés”.

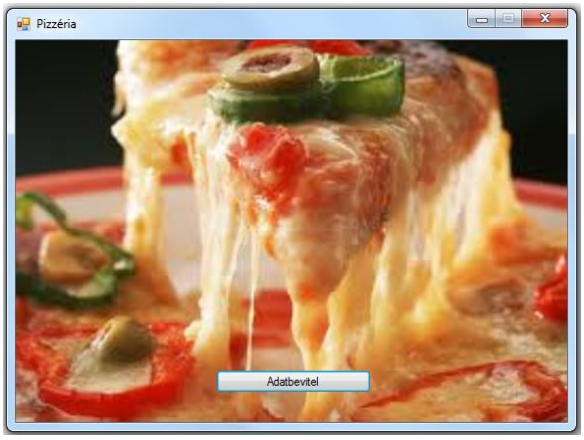


A feladat egyszerűen megoldható. Ugyanazokat a vezérlőelemeket kell felraknunk a formra, mint az előzőekben, az eddig megírt metódusok is csaknem változatlanok lesznek, egyetlen dologra kell odafigyelni, a vezérlőelemek láthatóságára. A form betöltésekor az Adatbevitel gomb kivételével mindegyikőjüknek láthatatlanná kell válnia, sikeres adatbevitel után pedig láthatóvá. Az már megállapodás kérdése, hogy az Adatbevitel gomb továbbra is látható legyen-e vagy sem.

A megoldáshoz szükséges metódusok:

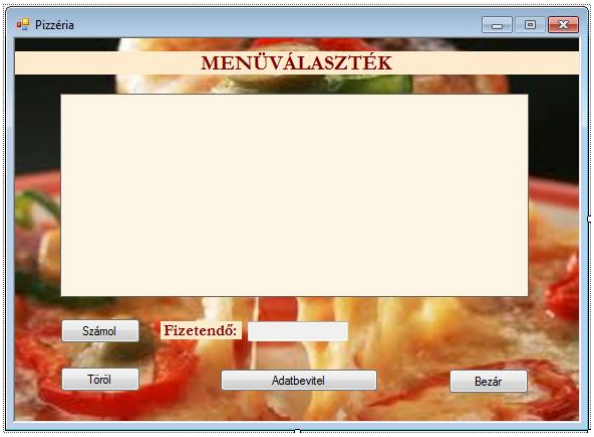
```
/// <summary>
///
/// A form betöltésekor állítsa false-ra a láthatóságot.
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_Load(object sender, EventArgs e) {
    LathatosagBeallitasa(false);
}

/// <summary>
///
/// Beállítja az összes vezérlőelem láthatóságát.
/// Az adatbevitel gomb láthatósága épp ellenkezője a többinek.
///
/// </summary>
/// <param name="lathatoE"></param>
private void LathatosagBeallitasa(bool lathatoE) {
    foreach (Control item in this.Controls) {
        item.Visible = lathatoE;
    }
    btnAdatBe.Visible = !lathatoE;
}

private void btnAdatBe_Click(object sender, EventArgs e) {
    try {
        AdatBevitel();
        ElemekMegjelenitese();
        LathatosagBeallitasa(true);
    }
    catch (Exception ex) {
        MessageBox.Show(ex.Message, "Hiba");
    }
}
```

Ezek után a tényleges feladat (és emlékeztetőül a két felületi kép):



Az induló felület alsó része ugyanaz, mint az előző feladat esetében.

A háttérben egy kép látható (a `PizzaForm` osztály `BackgroundImage` tulajdonságának beállításával lehet megadni, melyet érdemes `Stretch` layoutra állítani). A képet célszerű ugyanoda rakni, ahova az adatfájlt, vagyis a projekt bin\Debug mappájába, és a `BackgroundImage` tulajdonság beállításakor innen importálni.



A szükséges vezérlőelemeket célszerű egy központi panelre helyezni (legyen a neve `pnlKozponti`), hiszen szükség esetén a panel görgethető is (`AutoScroll` = `true`).

Betöltéskor az AdatBevitel gomb kivételével egyik vezérlőelem sem látható, majd sikeres beolvasás után láthatóvá válnak, az adatbevitelre használt gomb pedig láthatatlanná.

A láthatóságot beállíthatjuk a form betöltésekor is, illetve a konstruktorban is.

A megrendeléshez szükséges jelölőnégyzeteket, rádiógombokat, szövegdobozokat az adatfájl alapján hozzuk létre, és mivel később figyeljük a kiválasztásukat, ezért célszerű listákban tárolni őket.

```
public partial class PizzaForm : Form {

    private bool lathato;

    private List<Pizza> pizzak = new List<Pizza>();
    private List<CheckBox> jeloloNegyzetek = new List<CheckBox>();
    private List<RadioButton> rdBtnKicsiArak = new List<RadioButton>();
    private List<RadioButton> rdBtnNagyArak = new List<RadioButton>();
    private List<TextBox> txtDarabok = new List<TextBox>();
```

A láthatatlanság beállítása:

```
public PizzaForm() {
    InitializeComponent();
    // Beállítjuk az induló mappa helyét - ezt kínálja fel,
    // amikor megnyílik a dialógus ablak
    openFileDialog1.InitialDirectory = Environment.CurrentDirectory;
    // Beállítjuk, hogy milyen fájlnevet ajánljon fel indulásként
    // Ezt a tulajdonságot a Properties ablakban is beállíthatjuk
    openFileDialog1.FileName = "";
    // Beállítjuk a vezérlőelemek láthatóságát
    lathato = false;
    LathatosagBeallitas(lathato);
}

/// <summary>
///
/// Beállítja az összes vezérlőelem láthatóságát.
/// Az adatbevitel gomb láthatósága épp ellenkezője a többinek.
///
/// </summary>
/// <param name="lathatoE"></param>
private void LathatosagBeallitas(bool lathatoE) {
    foreach (Control item in this.Controls) {
        item.Visible = lathatoE;
    }
    btnAdatBe.Visible = !lathatoE;
}

private void btnAdatBe_Click(object sender, EventArgs e) {

    DialogResult result = openFileDialog1.ShowDialog();
    if (result == DialogResult.OK) {
        StreamReader olvasoCsatorna = null;
        try {
            // Ha megnyomtuk a dialógus-ablak OK gombját, akkor az ott megadott
            // fájlnév kerüljön át a mi alkalmazásunk  fajlNev változójába,
            // és evvel hívjuk meg az AdatBeolvasas() metódust.
            string fajlNev = openFileDialog1.FileName;
            olvasoCsatorna = new StreamReader(fajlNev);

            AdatBeolvasas(olvasoCsatorna);

            // Ez a metódus tölti fel a vezérlőelemeket
            ValasztekFeltoltes();

            // A láthatóság átállítása.
            lathato = true;
            LathatosagBeallitas(lathato);
            btnAdatBe.Visible = false;
            // Leszedjük a háttérképet.
            this.BackgroundImage = null;
        }
```

```
            catch (Exception ex) {
                MessageBox.Show(ex.Message, "hibaüzenet a fejlesztő számára");
            }
            finally {
                if (olvasoCsatorna != null) {
                    olvasoCsatorna.Close();
                }
            }
        }
    }
```

Megjegyzés: Ha nagyon precízen dolgoznánk, akkor itt több `catch` blokk is kellene, és külön kellene elkapni a fájl beolvasásából származó hibát, külön a választékfeltöltésből adódót, sőt talán még a láthatóság beállításából keletkezőt is. Ezekkel a hibákkal azonban a felhasználó nem sokat tud kezdeni (kivétel talán a fájlból való olvasás, hiszen ott ő is választhat rossz fájlt), ezért a fájlra vonatkozó üzenet kivételével a többit nem is neki kellene címezni, vagyis nem is üzenetdobozba kellene írni, hanem valamilyen log-fájlba, amit csak a fejlesztő nézeget.

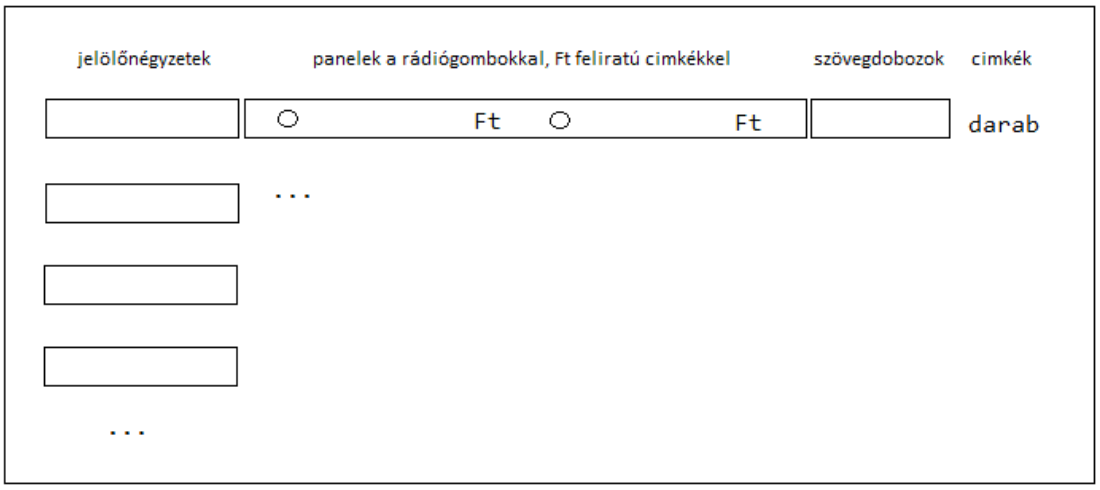
A beolvasás a szokásos, ezért itt nem közöljük.

A menüválaszték vezérlőelemeinek megjelenítése

Az adatbeolvasás során a fájl adataiból létrehozunk egy `pizzak` nevű, `Pizza` típusú objektumokból álló listát. A központi panelre (`pnlKozponti`) annyi sort kell felraknunk, ahány eleme van ennek a listának.

Vagyis: ennyi jelölőnégyzet kell, ennyiszer két rádiógomb, és ennyi szövegdoboz az adatok bevitelére. Mivel ezek mindegyikét használjuk majd a fizetendő összeg megállapításakor, ezért mindegyikből egy-egy listát kell képeznünk. Erre nem a vezérlőelemek felrakása miatt van szükség, hanem – mint már említettük – azért, mert később ezekből kell választanunk. Egyébként ez megoldható csak a vezérlőelemek segítségével is (azaz külön lista nélkül), csak talán így könnyebb átlátni. Gondolja végig, hogy hogyan oldható meg kizárólag vezérlőelemekkel.

A központi panelre ilyen elrendezésben kell felraknunk a vezérlőelemeket:



Az elemek felrakásához ismernünk kell:

- Az első jelölőnégyzet bal felső sarkának helyét (a központi panel bal szélétől és tetejétől való távolság).
- Két jelölőnégyzet egymás alatti távolságát (ez többféle módon is megadható, lehet úgy, hogy a felső négyzet bal alsó sarkának és a következő négyzet bal felső sarkának a távolsága, de lehet úgy is, hogy a két bal felső sarok távolsága).
- Az egy sorban lévő összes vezérlőelem magasságát.
- A jelölőnégyzet dobozának szélességét (erre azért van szükség, hogy ne a ráírt szöveg hosszától függjön, hogy utána hova kerül a rádiógomb).
- A rádiógombokat tartalmazó panel szélességét.
- A rádiógombok dobozának szélességét (erre szintén az egymás alá való pozicionálás miatt van szükség).
- A Ft feliratot tartalmazó címke szélességét (ennek elejére kerül a felirat, vagyis azért kell ez a szélesség, hogy kicsit arrébb kezdődjön a második rádiógomb).
- A darabszám bevitelére vonatkozó szövegdoboz szélességét.
- Mivel az árak és a darabszám is szám, ezért az értéküket tartalmazó szöveget jobbra kell igazítani, emiatt az őket tartalmazó doboz (rádiógombok, szövegmező) és a hozzájuk tartozó felirat között is kell egy kis közt hagynunk, így ennek értékét is meg kell adnunk.

A doboz feliratú címke szélességét nem kell megadnunk, csak arra kell majd vigyáznunk, hogy beleférjen a központi panelbe.

Ezek ismeretében egy ciklusban felrakhatók a vezérlőelemek. A feladatkiírásban látható felület létrehozásakor a következő beállításokat használtuk, de természetesen ettől el is lehet térni:

```
// Az első jelölőnégyzet távolsága a központi panel baloldalától és tetejétől
private int bal = 20, fent = 10;
// Két-két jelölőnégyzet bal felső sarkának egymástól való (függőleges) távolsága
private int kozY = 40;
// Az egyes vezérlőelemek magassága:
private int meretY = 20;
// A rádiógombokat tartalmazó panel vízszintes mérete:
private int panelX = 200;
// A jelölőnégyzet dobozának vízszintes mérete:
private int meretChk = 120;
// A rádiógombok dobozának vízszintes mérete:
private int meretAr = 50;
// A Ft felirat vízszintes mérete:
private int meretFt = 40;
// A darabszám bevitelére szolgáló szövegdoboz vízszintes mérete:
private int meretDb = 50;
// A Ft, ill. darab szöveg távolsága az előtte lévő doboztól:
private int koz = 3;
```

Egy-egy vezérlőelem létrehozásának lépései:

Létrehozunk egy példányt.

Beállítjuk a szükséges tulajdonságait (pozícióját (`Location`), méretét (`Size`), ha van hozzá felirat, akkor a feliratát (`Text`), illetve esetleges egyéb tulajdonságokat).

Rárakjuk a központi panelre (hozzáadjuk a vezérlő komponenseit tartalmazó gyűjteményhez (`Collections`)). A rádiógombokat természetesen a központi panelre kerülő, és őket tartalmazó panelre rakjuk.

Ha a programból hivatkozni szeretnénk az illető vezérlőelemre, akkor még a vezérlőelemeket tartalmazó listához is hozzáadjuk, ha nem hivatkozunk rá, akkor nincs szükség ilyen listára. (Esetünkben pl. létre kell hozni a jelölőnégyzeteket, rádiógombokat, illetve a darabszámok megadására szolgáló szövegmezőket tartalmazó listákat, de nincs szükség a rádiógombokat tartalmazó paneleket, illetve a feliratokat tartalmazó labelekből álló listákra.)

Fontos jó tanács: annak végiggondolásához, hogy mire van szükség, illetve azt hogyan lehet beállítani, sok segítséget kaphatunk a PizzaForm.Designer.cs fájl tanulmányozásával – ebben szerepel a Visual Studio által generált `InitializeComponent()` metódus, abban meg lehet nézni a generált beállításokat. Persze, jelenleg alapértelmezetten egyetlen checkBox vagy radioButton sincs fent, de ezen lehet segíteni: ideiglenesen felrakunk egy-egy elemet a központi panelre, és megnézzük, mit generál a Visual Studio. Ezt már gyerekjáték átírni a mi szájunk íze szerint.

Most a központi panelhez hozzá kell adnunk pizzánként egy-egy

- jelölőnégyzetet (`CheckBox`)

- panelt (`Panel`), és erre a panelre rá kell tennünk
    - két rádiógombot (`RadioButton`)
    - két címkét (`Label`) – ezek a „Ft” szöveg kiíratása miatt kellenek
    - szövegdobozt (`TextBox`)
    - cimkét (`Label`) – ez a „darab” szöveg kiíratása miatt kell.

Ezek közül a jelölőnégyzeteket, rádiógombokat és szövegdobozokat egyenként hozzá kell adni a megfelelő listához is.

A szöveget is tartalmazó vezérlőelemek esetében állítsuk be az igazítást is (`TextAlign` tulajdonság). Próbálja ki a beállítások nélkül is. Láthatja majd, hogy ekkor kicsit táncikálnak a szövegek.

A felrakást végző metódus:

```
private void ValasztekFeltoltes() {

    CheckBox checkBox;
    Label label;
    RadioButton radioButton;
    Panel panel;
    TextBox textBox;

    for (int i = 0; i < pizzak.Count; i++) {

        // checkBox felrakása
        checkBox = new CheckBox();
        checkBox.TextAlign = ContentAlignment.MiddleLeft;
        checkBox.Text = pizzak[i].Nev;
        checkBox.Location = new Point(bal, fent + i*kozY);
        checkBox.Size = new Size(meretChk, meretY);

        // a most beállított jelölőnégyzetet hozzáadjuk
        // a jelölőnégyzetek listájához
        jeloloNegyzetek.Add(checkBox);

        // a most beállított jelölőnégyzetet rárakjuk
        // a központi panelre
        pnlKozponti.Controls.Add(checkBox);

        // panel felrakása
        panel = new Panel();
        panel.Size = new Size(panelX, meretY);
        panel.Location = new Point(bal + meretChk, fent + i * kozY);

        // A panelt rárakjuk a központi panelre
        pnlKozponti.Controls.Add(panel);

        // radiobtn kicsi ár felrakása
        radioButton = new RadioButton();
        radioButton.Size = new Size(meretAr, meretY);
        radioButton.TextAlign = ContentAlignment.MiddleRight;
        radioButton.Text = pizzak[i].ArKicsi.ToString();
        radioButton.Location = new Point(0, 0);
        rdBtnKicsiArak.Add(radioButton);

        // A rádiógombot rárakjuk az őket tartalmazó panelre
        panel.Controls.Add(radioButton);

        // Fix Ft felirat
        label = new Label();
        label.TextAlign = ContentAlignment.MiddleLeft;
        label.Text = " Ft";
        label.Location = new Point(meretAr + koz, 0);
        label.Size = new Size(meretFt, meretY);
        panel.Controls.Add(label);
```

```
            // radiobtn nagy ár
            // TODO: oldja meg önállóan


            // Fix Ft felirat
            // TODO: oldja meg önállóan


            // textBox darabszám
            // TODO: oldja meg önállóan


            // Fix darab felirat
            // TODO: oldja meg önállóan
        }
```

Megjegyzés: A „Ft” kiíratásához nem feltétlenül kellene külön label, de ha a rádiógombok szövegébe írjuk bele, akkor „táncol” a kiíratás, így viszont pontosan egymás alá kerülnek a „Ft” feliratok.

A Számol gombhoz tartozó metódus közlése előtt beszélgessünk róla egy kicsit.

Az egyértelmű, hogy egy ciklussal végig kell mennünk a jelölőnégyzetek listáján, és a kiválasztottak esetén összegezni a hozzájuk tartozó kiszámított árakat. A baj csak ott van, hogy rendelés közben a felhasználó hibázhat. Végig kell tehát gondolnunk, hogy milyen hibák fordulhatnak elő, és ezekre hogyan lehet felkészíteni a programunkat. Figyelni kell tehát, hogy egyáltalán választottunk-e pizzát, megadtuk-e a méretét, illetve helyes darabszámot gépeltünk-e.

Első nekifutásra úgy tűnik, hogy ezt a problémát már megoldottuk az előkészítő feladatban, azonban itt kicsit bonyolultabb a helyzet, ugyanis előfordulhat, hogy több pizzafajtát is választottunk, és többhöz nem rendeltünk méretet vagy több darabszámot is rosszul adtunk meg. Hogyan és hol kezeljük ezeket az eseteket?

Ennek megoldására több alternatíva is kínálkozik.

1. A legcsúnyább megoldás az (és remélhetőleg ez a változat eszébe sem jut ☺), hogy az egész ciklust egy `try-catch` blokk `try` ágába írjuk, és egy általános `Exception` típusú kivételt dobva hiba esetén kiíratjuk a „Hibás adatmegadás” szöveget. Ez azonban egyáltalán nem felhasználóbarát megoldás, egyikünk sem szereti, ha felhasználóként ilyen semmitmondó üzenettel találkozik.

2. Egy fokkal jobb, ha legalább a jelölőnégyzetet kivesszük a kivételkezelés hatása alól, és egy logikai változó segítségével figyeljük, hogy egyáltalán van-e kiválasztás, és csak a kiválasztott eseteket bízzuk a kivételkezelésre. De még mindig ott van problémaként a többi hibalehetőség. A semmitmondó hibaüzenet elkerüléséhez célszerű figyelni a kivétel keletkezésének okát (azaz a dobott kivétel típusát). A méretválasztás elmulasztása esetén `MissingFieldException` típusú kivétel keletkezik. Az előkészítő feladatban azt is megkülönböztettük, hogy a darabszám megadása miért hibás (rossz formátumú a szám, vagy negatív egész), erre azonban nem feltétlenül van szükség, a „Hibás adatmegadás” szöveg alapján a felhasználó maga is rájöhet a hiba okára. Ezért itt a kétféle kivétel helyett használhatjuk az általános `Exception` típust.

3. További kérdés lehet, hogy valóban a kivételkezelés tartalmazza-e a teljes ciklust, vagy épp fordítva, a cikluson belül legyen kivételkezelés. Az előbbi változat mellett az szól, hogy ekkor csak egyszer kapjuk meg az adott fajta hibaüzenetet, függetlenül attól, hogy pl. hány darabszámot rontottunk el. Ha a ciklus tartalmazza a kivételkezelést, akkor annyiszor kapjuk meg a hibaüzenetet, ahányszor elkövettük az adott hibatípust. Ez elég idegesítő lehet (ámbár pont emiatt nagyobb fegyelemre szoktatja a felhasználót ☺). Ugyanekkor emellett is szólhat – egy halvány – érv. Az, hogy ez esetben például át tudjuk színezni a hiba forrását (pl. a darabszám megadására szolgáló mezőt). Hogy az átszínezésre is lásson példát, ezért a közölt kód ezt a változatot tartalmazza majd, de ha egy kicsit ügyes, akkor kombinálja a kettőt, vagyis próbálja megoldani azt, hogy az adott hibára csak egyetlen hibaüzenetet kapjunk, de mégis mindegyik hibás mező át legyen színezve.

4. Ha már az alternatíváknál járunk, említsünk meg még továbbiakat. Nemegyszer lehet látni olyan programot, amelyben eleve kikerülik azt az esetet, hogy nincs bejelölve egyik rádiógomb sem, mégpedig úgy, hogy alapértelmezett beállítást definiálnak. Esetünkben ez pl. azt jelentené, hogy az alkalmazás indulásakor már eleve ki van választva a két rádiógomb közül az egyik. Ez úgy oldható meg, hogy a `ValasztekFeltoltes()` metódusban az egyik rádiógomb `Checked` tulajdonságát `true`-ra állítjuk. Így talán elkerülhető egy kivételkezelés, de meglehetősen csúnya felületet kapnánk, ezért nem szerencsés ez a megoldás.

5. Az is megoldás lehet, hogy alapértelmezettnek tekintjük valamelyik változatot, de nem a választék felrakásakor, hanem csak akkor, amikor kiválasztottuk a hozzá tartozó pizzát. Ez úgy oldható meg, hogy figyeljük a jelölőnégyzet állapotának változását, és ha bekapcsoljuk, akkor a program jelölje ki az egyik rádiógombot, ha kikapcsoljuk, akkor állítsa vissza őket jelöletlenre. Ehhez a jelölőnégyzethez tartozó eseményt kell figyelni. A fejezet végén ezt a változatot is megbeszéljük, mert hasonló módon lehet eseményt rendelni bármelyik más, kódból feltett vezérlőelemhez.

Nézzük tehát a 3. számú alternatívában említett megoldás kódját. (Ismételten hangsúlyozzuk, hogy nem ez a legjobb változat, az igazi az ugyancsak ebben a pontban felvetett és önálló megoldásra ajánlott „kevert” elképzelés lenne.)

```
private void btnSzamol_Click(object sender, EventArgs e) {
    bool vanKijeloles = false;
    int db, osszeg = 0, ar = 0;

    for (int i = 0; i < jeloloNegyzetek.Count; i++) {

        if (jeloloNegyzetek[i].Checked) {
            vanKijeloles = true;
            try {
                // ha az első fajtát választottuk
                if (rdBtnKicsiArak[i].Checked) ar = pizzak[i].ArKicsi;
                // ha a második fajtát választottuk
                else if (rdBtnNagyArak[i].Checked) ar = pizzak[i].ArNagy;
                // ha egyáltalán nem választottunk
                else throw new MissingFieldException();

                // ide már csak akkor jut, ha tényleg választottunk, ezért
                // kiolvashatja a db értékét. Ha hibás formátumot adunk meg
                // (üres vagy nem egész szám), akkor FormatException kivételt dob.
                db = int.Parse(txtDarabok[i].Text);

                // azt is vizsgálni kell, hogy nem írtunk-e negatív értéket.
                if (db < 0) throw new Exception();
                txtDarabok[i].BackColor = Color.White;
                osszeg += ar * db;
            }
            catch (MissingFieldException) {
                MessageBox.Show("Nem választott méretet", "Hiba");
            }
            catch (Exception) {
                MessageBox.Show("Hibásan adta meg a darabszámot", "Hiba");
                txtDarabok[i].BackColor = Color.Coral;
                txtDarabok[i].Clear();
            }
        }
    }
    if (!vanKijeloles) MessageBox.Show("Nincs kijelölve semmi", "Figyelmeztetés");
    else txtFizetendo.Text = osszeg + " Ft";
}
```

A Töröl gomb kezelését oldja meg önállóan! Arra figyeljen, hogy ciklusban kell kitörölni minden beállítást.

Végül térjünk rá az 5. alternatívaként említett változat megbeszélésére.

Ha ezt az alternatívát választja, akkor – a többi beállítás változatlanul hagyása mellett – a `ValasztekFeltoltes()` metódusban a `checkBox` objektumhoz újabb beállításként azt is hozzá kell vennünk, hogy figyelje és kezelje a jelölés megváltoztatása eseményt:

```
// hozzárendeljük az eseménykezelést is
checkBox.CheckedChanged += new EventHandler(checkBox_CheckedChanged);
```

Megjegyzés: most igazán sok segítséget kaphat, ha próbaképpen felrak egy ideiglenes checkBox elemet a tervezési felületre, hozzárendeli az eseményt, és megfigyeli, hogy ennek hatására milyen kód íródott a PizzaForm.Designer.cs fájl `InitializeComponent()` metódusába.

Az esemény bekövetkezésekor azt kell figyelnünk, hogy be van-e kapcsolva az aktuális jelölőnégyzet, mert ha igen, akkor bekapcsoljuk az egyik rádiógombot is. (Ha a pizzéria tulajdonosa a megrendelő, akkor valószínűleg a nagyobb méretűt illik bekapcsolni. ☺) Ha ki van kapcsolva, akkor viszont mindkét rádiógombnak kikapcsolt állapotban kell lennie.

Egyetlen dolgot kell végiggondolnunk: honnan tudja majd a metódus, hogy melyik jelölőnégyzet küldte az eseményt?

Pontosan ezt mondja meg a metódus `sender` paramétere. Meg kell néznünk, hogy ez a jelölőnégyzeteket tartalmazó lista hányadik eleme, és máris tudjuk, hogy melyik rádiógombot kell beállítani. Egyetlen probléma van még: a `sender` alapértelmezetten `object`, és nem `CheckBox` típusú. De ezen egy típuskényszerítéssel könnyedén tudunk változtatni.

Az eseményt kezelő metódus:

```
private void checkBox_CheckedChanged(object sender, EventArgs e) {

    CheckBox valasztoGomb = (CheckBox)sender;
    // megállapítjuk, hogy melyik sorszámú jelölőnégyzetet
    // választottuk
    int index = jeloloNegyzetek.IndexOf(valasztoGomb);

    if (valasztoGomb.Checked) {
        rdBtbNagyArak[index].Checked = true;
    }
    else {
        rdBtbKicsiArak[index].Checked = false;
        rdBtbNagyArak[index].Checked = false;
    }
}
```

2.2.2. Diákkezelő alkalmazások

Ebben a fejezetben vezérlőelemek kódból való felrakását gyakoroljuk két kisebb feladaton

keresztül.

Az Adatbevitel gomb hatására olvassuk be a *diakok.txt* fájlból a diákok adatait. (Adatszerkezet: nev;tanulmányi-kód;születési_év). Beolvasás után az Adatbevitel gomb váljon inaktívvá, a Kiválasztás aktívvá. A Kiválasztás gomb hatására a kiválasztott diákok kerüljenek be a kiválasztottak listájába, és egyúttal kerüljenek be a kiválasztottak közül a legidősebbek a legidősebbek listájába. Ha senki sincs kiválasztva, adjon hibaüzenetet.

A kiválasztottak listájára kattintva a listák alatt jelenjen meg a kiválasztott diák neve, kódja és születési éve.

Megoldásrészletek:

Mivel futás közben kiválasztott fájlból akarunk olvasni, ezért fel kell rakni egy `OpenFileDialog` vezérlőelemet. Ennek sok jó alapbeállítása van, de kettőt mindenképpen nekünk kell megírni: annak beállítását, hogy mi legyen az induló mappa, illetve azt, hogy milyen fájlnevet ajánljon fel.

Legegyszerűbb, ha az induló mappa a projekt bin\Debug mappája, a felkínált fájlnév pedig üres. Ezeket pl. a form konstruktorában állíthatjuk be (a fájlnevet a Properties ablakban is beállíthatjuk):

```
openFileDialog1.InitialDirectory = Environment.CurrentDirectory;
openFileDialog1.FileName = "";
```

Induláskor beállítjuk a gombok aktivitását is – ezt lehetne a konstruktorban is, de megoldhatjuk a form betöltésekor is:

```
private void Form1_Load(object sender, EventArgs e) {
    GombBeallitas(false);
}

private void GombBeallitas(bool aktiv) {
    btnAdatbevitel.Enabled = !aktiv;
    btnKivalaszt.Enabled = aktiv;
}
```

Ahhoz, hogy a beolvasott adatokat kezelni tudjuk, szükségünk van a `Diak` osztályra, ezt írja meg önállóan.

Az `AdatBevitel()` metódusban megnyitjuk a dialógusablakot, és ha sikerült kiválasztanunk a fájlt, akkor megpróbáljuk beolvasni az adatokat. Ha ez is sikerült, akkor az adatok alapján felrakjuk a felületre a vezérlőelemeket, majd átállítjuk a gombok aktivitását.

```
private void Adatbevitel() {
    DialogResult eredmeny = openFileDialog1.ShowDialog();
    if (eredmeny == DialogResult.OK) {
        string fajlNev = openFileDialog1.FileName;
        try {
            AdatBeolvasas(fajlNev);
            FelrakDiakok();
            GombBeallitas(true);
        }
        catch (Exception) {
            MessageBox.Show("Hiba a fájl beolvasásakor", "Hiba");
        }
    }
}
```

Megjegyzés: A fájlnév megadását akkor nem kell beraknunk a try blokkba, ha elfogadjuk az `openFileDialog1` alapértelmezett beállítását, azt, hogy `CheckFileExists = True`, `CheckPathExists = True`.

Az adatbeolvasás során csak az a kérdés, hogyan dolgozzuk fel a beolvasott sorokat.

```
private void Feldolgoz(string adat) {
    // Az adatfájl szerkezete:
    // Hegedűs Emma;HEEMAB.PTE;1990

    string[] adatok = adat.Split(';');

    Diak diak = new Diak(adatok[0], adatok[1], int.Parse(adatok[2]));
    diakok.Add(diak);
}
```

A `diakok` nevű változó a diákok listáját jelöli.

Egy kicsit időzzünk el a vezérlőelemek felrakásánál. Célszerű őket egy panelre tenni, mert beállítható, hogy a panel automatikusan gördíthetővé váljon, ha így kívánja a felrakott vezérlőelemek mennyisége. (Be kell állítani a panel `AutoScroll` tulajdonságát.)

A panelre `pnlDiakok` néven hivatkozunk.

Mivel később vizsgálni kell őket, ezért a felrakott `CheckBox` elemekből is listát kell készítenünk, illetve mindegyiket hozzá kell adnunk a panelhez.

Ezt csinálja a `FelrakDiakok()` metódus.

```
private void FelrakDiakok() {
    CheckBox chkBox;

    for (int i = 0; i < diakok.Count; i++) {
        chkBox = new CheckBox();
        chkBox.AutoSize = true;
        chkBox.Location = new Point(kezdoX, kezdoY + i * chkYKoz);
        chkBox.Text = diakok[i].ToString();

        pnlDiakok.Controls.Add(chkBox);
        chkBoxok.Add(chkBox);
    }
}
```

Egy-egy vezérlőelem felrakásához sokat segít a Form1.Designer.cs fájl tanulmányozása. Ezt nézze át, de nem kell minden beállítást átvennünk onnan.

Most ennyit csináltunk:

Létrehoztunk egy `CheckBox` példányt, beállítottuk automatikus méretezésre (így a mérete alakul majd a hozzá tartozó szöveg hosszához), megadtuk, hogy hol legyen a vezérlőelem bal felső sarka, és megadtuk, hogy milyen szöveg tartozzon hozzá.

Végül hozzáadtuk az elkészült példányt a panelhez is és a `chkBoxok` nevű listához is. Ez utóbbi a `CheckBox` típusú elemeket tartalmazó lista.

A `kezdoX` a panel bal oldalától való távolságot jelöli, most függőlegesen is ugyanekkora távolsággal dolgoztunk (ez a `kezdoY` érték). A `chkYkoz` pedig a checkbox-elemek közötti távolságot határozza meg.

A kiválasztás során ellenőriznünk kell, hogy ki van-e választva valamelyik jelölőnégyzet. Ha egyik sincs, akkor hibaüzenetet adunk, ha van kijelölt, akkor a kiválasztott diákpéldányokat berakjuk az `lstKivalasztottak` nevű listadobozba, a legidősebbekhez tartozó példányokat pedig az `lstLegidosebbek` nevűbe. Mivel többször is meghívhatjuk ezt az eseményt, ezért előtte törölnünk kell a listadobozokat.

```
private void btnKivalaszt_Click(object sender, EventArgs e) {
    Kivalaszt();
}
```

```
private void Kivalaszt() {
    bool vanValasztott = false;
    lstKivalasztottak.Items.Clear();
    for (int i = 0; i < chkBoxok.Count; i++) {
        if (chkBoxok[i].Checked) {
            lstKivalasztottak.Items.Add(diakok[i]);
            vanValasztott = true;
        }
    }
    if (!vanValasztott) {
        MessageBox.Show("Senkit sem választott", "Hiba");
    }
    else {
        MinKeres();
    }
}

private void MinKeres() {
    lstLegidosebbek.Items.Clear();
    int min = (lstKivalasztottak.Items[0] as Diak).szulEv;
    foreach (Diak diak in lstKivalasztottak.Items) {
        if (diak.szulEv < min) min = diak.szulEv;
    }
    foreach (Diak diak in lstKivalasztottak.Items) {
        if (diak.szulEv == min) lstLegidosebbek.Items.Add(diak);
    }
}
```

A `MinKeres()` metódus során ki kell keresnünk az `lstKivalasztottak` listából a legrégebben születetteket, és berakni őket az `lstLegidosebbek` nevű listába.

Megjegyzések:

1. A konkrét feladat esetében nincs is szükség a `vanValasztott` változóra, hiszen azt is vizsgálhatnánk, hogy az `lstKivalasztottak.Items` gyűjtemény nem üres-e (elemeinek száma nulla). Ez a megoldás azonban kihasználja, hogy mi történik a kiválasztás hatására. Ha csak az a kérdés, hogy van-e kiválasztott jelölőnégyzet, akkor a most közölt módon tudjuk megoldani.

2. A feladat megoldható a `chkBoxok` lista nélkül is – ekkor a `pnlDiakok.Controls` gyűjteményében lévő elemek közül kell kiválasztanunk a `CheckBox` típusúakat, és ezeket kell vizsgálnunk. Ha elég ügyesnek érzi magát, próbálja ki!

Két részlet (a kettő közül egyszerre csak az egyik alkalmazható):

a/

```
for (int i = 0; i < pnlDiakok.Controls.Count; i++) {
    if ((pnlDiakok.Controls[i] as CheckBox).Checked) {
        lstEredmenyek.Items.Add(diakok[i]);
        vanValasztott = true;
    }
}
```

b/

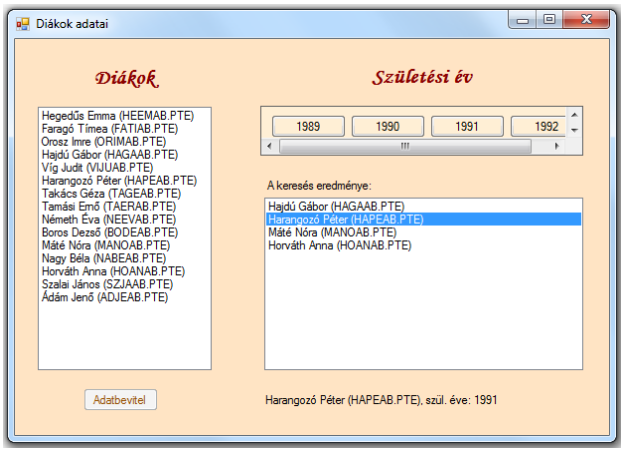
```
int i = 0;
foreach (CheckBox item in pnlDiakok.Controls) {
    if (item.Checked) {
        lstEredmenyek.Items.Add(diakok[i]);
        vanValasztott = true;
    }
    i++;
}
```

FONTOS: Mindkét esetben kihasználtuk, hogy a panelen kizárólag `CheckBox` típusú példányok vannak. Ha lennének más vezérlőelemek is, akkor itt vizsgálni kellene, hogy az elem valóban `CheckBox` típusú-e, illetve az index meghatározása is bonyolultabb lenne.

Utolsó lépésként meg kell oldanunk, hogy a listára kattintva olvashassuk a diákok adatait az `lblDiak` nevű címkén.

```
private void lstKivalasztottak_SelectedIndexChanged(object sender, EventArgs e) {
    Diak diak = (Diak)lstKivalasztottak.SelectedItem;
    if (diak != null) lblDiak.Text = diak + ", születési éve: " + diak.szulEv;
}
```

A feladat b/ része:



Most a diákok adatait egy listadobozba olvassuk be, az adatbeolvasás során a Születési év felirat alatt jelenjenek meg olyan gombok, amelyek a diákok születési évét tartalmazzák feliratként (minden év csak egyszer szerepelhet). Az adott évszámgombra kattintva a keresés eredményeként jelenjen meg a listában az abban az évben született diákok névsora.

A listára kattintva a lista alatt jelenjenek meg a kiválasztott diák adatai.

Megoldásrészletek:

Összesen két dolgot kell megbeszélnünk, a `Feldolgoz()` és a `FelrakEvek()` metódust, illetve azt, ami ezekből még következik.

A feldolgozás során ugyanúgy hozhatjuk létre a `diak` példányt, ahogy az előző feladatban, a különbség annyi, hogy most nem a `diakok` nevű `List` gyűjteménybe rakjuk őket, hanem meg akarjuk jeleníteni egy listadobozban. A listadoboz felületére azonban csak azok az adatok kerülnek ki, amelyek benne vannak a listadoboz `Items` nevű gyűjteményében. Vagyis ehhez kell most hozzáadnunk a diákpéldányt.

Ezenkívül van még egy feladatunk: ki kellene gyűjteni az `evek` nevű, `int` típusú elemeket tartalmazó listába (`List`) a születési éveket. De mindegyiket csak egyszer. Ezért csak akkor kerül be egy évszám a listába, ha még nincs benne.

```
private void Feldolgoz(string adat) {
    string[] adatok = adat.Split(';');
    Diak diak;
    diak = new Diak(adatok[0], adatok[1], int.Parse(adatok[2]));
    lstDiakok.Items.Add(diak);
    if (!evek.Contains(diak.SzuletesiEv)) evek.Add(diak.SzuletesiEv);
}
```

A `FelrakEvek()` metódus helyezi el a panelre a gombokat. (A panelre `pnlEvek` néven hivatkozunk.)

Annyi gombot kell felraknunk, ahány eleme van az `evek` listának. A vezérlőelemeket ugyanúgy rakjuk fel, mint az előző feladatban, most annyi az újdonság, hogy minden egyes gombhoz eseményt is rendelünk, vagyis közöljük az eseménykezelővel, hogy figyelje a gombnyomás eseményt, és ha bekövetkezik, akkor hajtsa végre a `Kivalaszt()` metódust.

FIGYELEM: a paraméterben csak a metódus neve szerepel!

Nyilván meg kell írnunk ezt a metódust, könnyebb azonban, ha a Visual Studio-val generáltatjuk.

A metódus egyik paramétere az az objektum, amelyen az esemény keletkezett. Ez jelenleg az épp aktuális gomb. Mivel `Object` típusú, ezért típuskényszerítést kell alkalmaznunk.

```
private void FelrakEvek() {
    Button btn;
    evek.Sort();
    for (int i = 0; i < evek.Count; i++) {
        btn = new Button();
        btn.Location = new Point(kezdoX + i * btnXKoz, kezdoY);
        btn.Text = evek[i].ToString();

        // Hozzárendeljük minden egyes gombhoz azt, hogy
        // a gombnyomás esemény hatására hajtsa végre a Kivalaszt
        // nevű metódust. (Magát a metódust célszerű generáltatni.)
        btn.Click += new System.EventHandler(Kivalaszt);

        // Rárakjuk a panelre a gombot
        pnlEvek.Controls.Add(btn);

        // Hozzáadjuk a gombok listájához.
        btnEvek.Add(btn);
    }
}

private void Kivalaszt(object sender, EventArgs e) {

    // Az ev értéke az aktuálisan megnyomott gomb felirata.
    int ev = int.Parse((sender as Button).Text);

    lstEredmeny.Items.Clear();
    foreach (Diak diak in lstDiakok.Items) {
        if (diak.SzuletesiEv == ev) lstEredmeny.Items.Add(diak);
    }
}
```

Megjegyzés: az `evek.Sort()` növekvő sorrendbe rendezi az éveket, de csak primitív típusokból álló lista esetén ilyen egyszerű a rendezés, objektumokat tartalmazó listák esetén kicsit bonyolultabb.

3. Menükezelés, formok közötti kapcsolat

3.1. Elméleti háttér *(Szendrői E.)*

Egy grafikus alkalmazás általában nem egyetlen ablakból áll. Az ablak különböző menüket, funkciógombokat tartalmaz, melyek több másik ablakba navigálják a felhasználót. A grafikus felületű alkalmazásoknál az ablak megjelenése szabványosított, abban az értelemben, hogy a menüsornak az ablak tetején kell lennie, és a menüpontok sorrendje és tartalma is kötött, gondoljunk a minden ablakban látható Fájl és Szerkesztés menükre. Természetesen a fejlesztő maga is készíthet saját menüket.

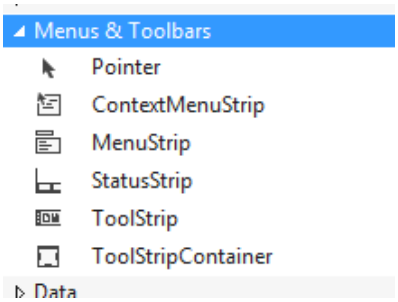
Menük

Minden ablakban az ablak tetején az úgynevezett főmenü helyezkedik el, ami futás közben mindig látható, s első sora általában nem közvetlen parancsot, hanem menücsoportok nevét tartalmazza.

A C# nyelven Menük a Menustrip és ContextMenuStrip osztályokkal készíthetők. Mindkét menütípus a ToolStrip osztály leszármazottja. A Windows form alkalmazásokban kétféle menütípust használhatunk:

- Legördülő menü-> MenuStrip,
- Környezeti menü -> ContextMenuStrip.

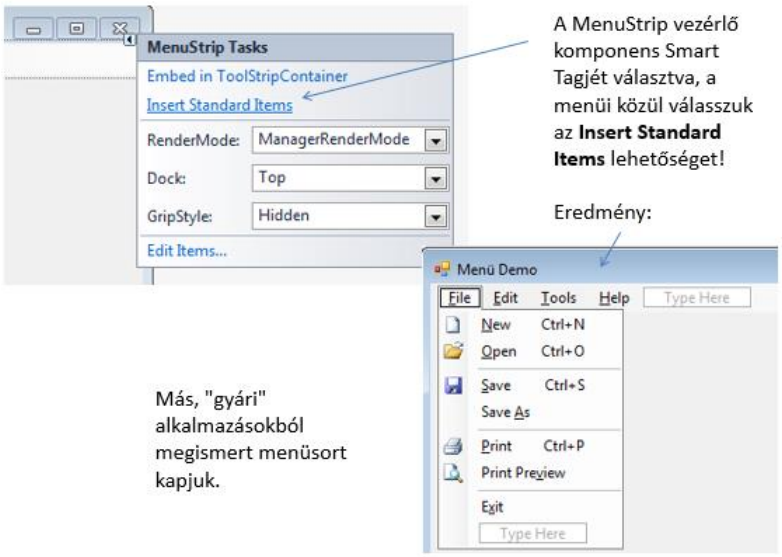
A menüpontokhoz tartozó utasításokat a menüpont Click eseménykezelőjébe kell megírnunk. A menüvezérlő komponenseket a ToolBox Menus & Toolbars csoportjában találjuk.



Amikor ráhúzzuk a formra a MenuStrip vezérlőelemet, a menuStrip objektum a komponens tálcán jelenik meg. Ha rákattintunk a menüsor TypeHear feliratára, megadhatjuk a létrehozandó menüpont nevét.

Ha a **MenuStrip** jobb felső sarkán látható *Smart Tagre* kattintunk, akkor a látható menük közül kiválaszthatjuk az I*nsertStandardMenu*-t. Ennek az lesz az eredménye, hogy a

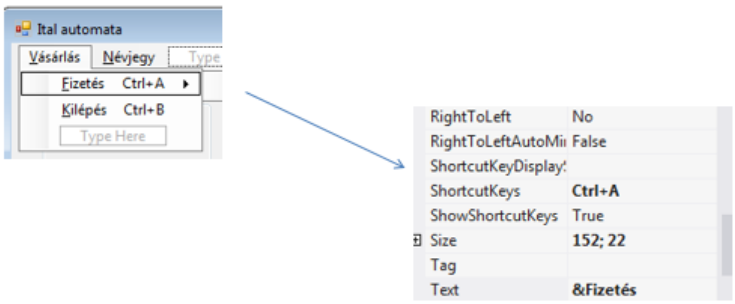
Windows Formokon szokásos, szabványos menüsor kerül a formra. Természetesen ez módosítható.



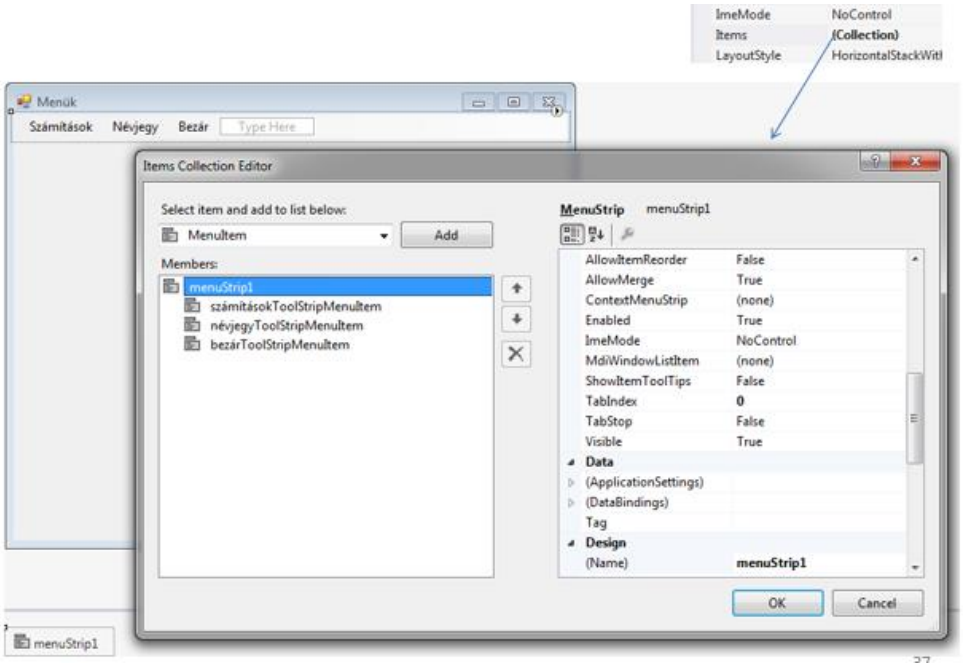
A MenuStrip alapértelmezett objektumtípusa a MenuItem objektum.

A menü objektumok legfontosabb tulajdonságai az alábbi ábrán láthatók. A ShortCutKey tulajdonság úgynevezett forró billentyűk menüponthoz való rendelését teszik lehetővé. Ezek a forró billentyűk vagy a Ctrl vagy Shift és valamilyen másik billentyű kombinációi. A Text tulajdonság a menü szövege. A menüelemeket az Enabled tulajdonság segítségével tehetjük aktívvá és kapcsolhatjuk ki, és a Visible tulajdonsággal rejtjük el vagy fedjük fel őket. A ShowShortCutKey értékének true-ra állításával a menü felirata mellett a forró billentyű is látható lesz. Az Items tulajdonság a menüelemek gyűjteményét adja vissza.

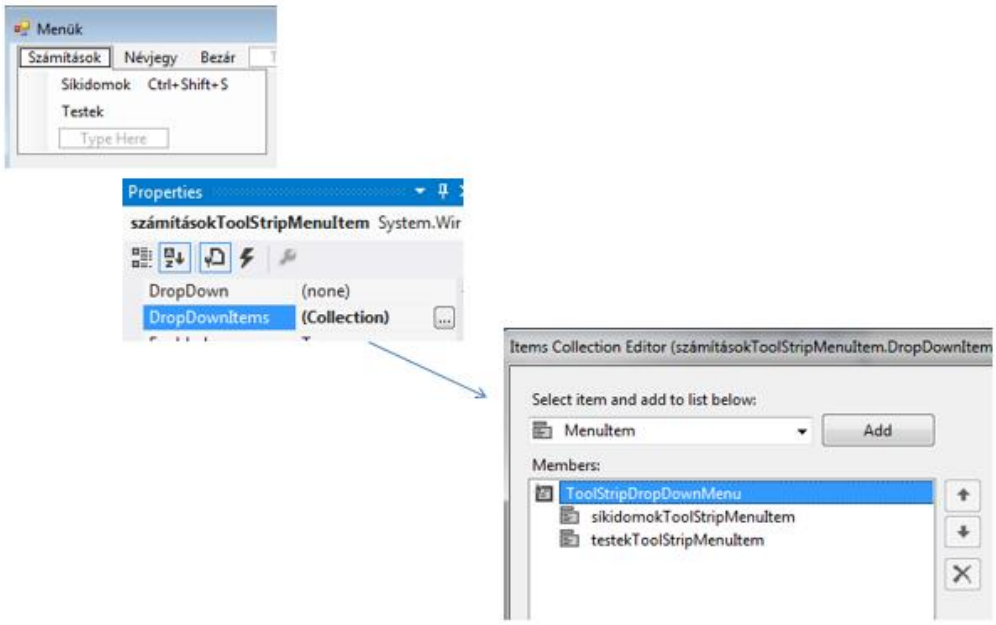
DropDownItems	Items
Image	ShortcutKeys
ShowShortcutKey	Text



Mivel a menu Items tulajdonsága egy menügyűjtemény, elemeit szerkeszthetjük egy külön ablakban. Az Items Collection ablakban menüpontokat törölhetünk, újakat adhatunk hozzá, megváltoztathatjuk a sorrendjüket.



Egy menüpontnak, ha vannak legördülő elemei, akkor azok a MenuItem DropdownItems gyűjtemény Items Collection Editor ablakában szerkeszthetők.



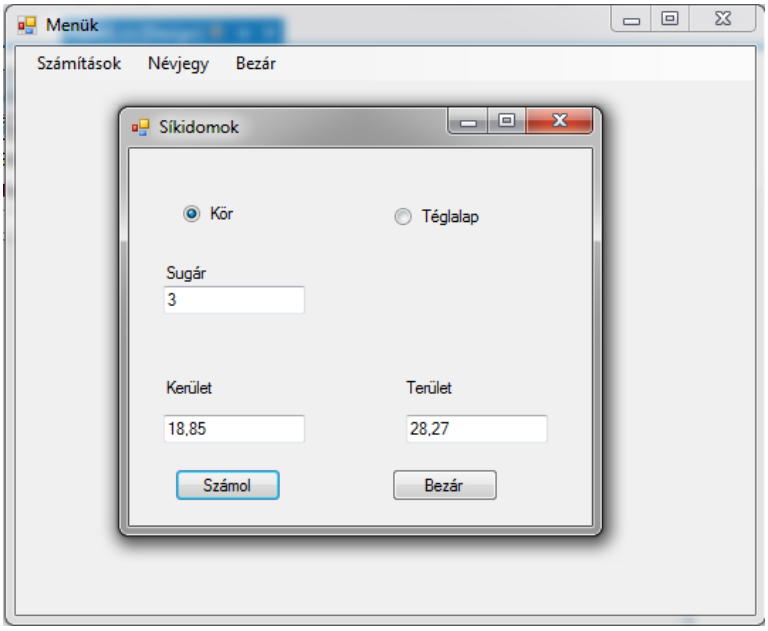
Amikor egy felhasználó egy menüpontra kattint, különböző események következhetnek be. A menüpontok esetében a leggyakrabban figyelt esemény a **Click** esemény. Amikor ez bekövetkezik egy menüponton, akkor a menüponthoz tartozó funkciót kell végrehajtani. Ezt a

funkciót, úgy, mint eddig minden esemény bekövetkeztekor, egy eseménykezelő metódusban valósítjuk meg. Az alábbi kódrészletben a Bezár menüre kattintás eseménykezelője, valamint a Síkidomok és a Névjegy menükre kattintás eseménykezelője látható.

A Bezár menü Click eseménykezelőjében meghívjuk a Close metódust, hogy bezárja az alkalmazás ablakát. A másik két menüpontban a funkciónak megfelelő form objektumokat hozunk létre, és az objektum Show() metódusával megnyitjuk a megfelelő formot.

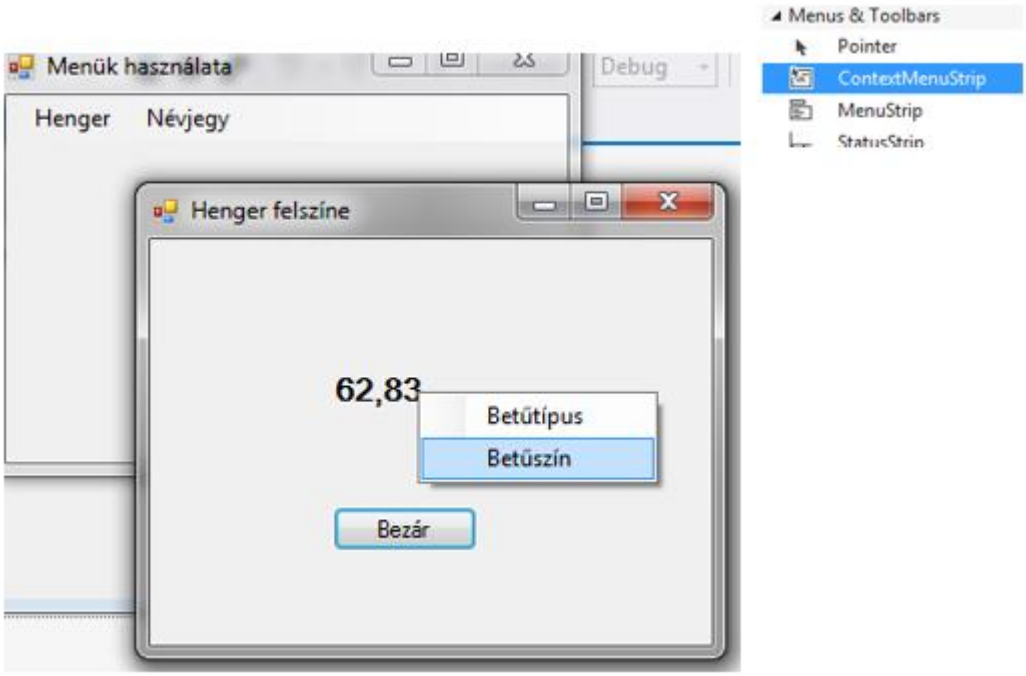
```
private void bezárToolStripMenuItem_Click(object sender, EventArgs e)
{
    Close();
}

private void síkidomokToolStripMenuItem_Click(object sender, EventArgs e)
{
    Sikidom sik = new Sikidom();
    sik.Show();
}

private void névjegyToolStripMenuItem_Click(object sender, EventArgs e)
{
    Nevjegy nevj = new Nevjegy();
    nevj.Show();
}
```

Az alábbi képen a Számítások menü Síkidomok menüjének Click eseménykezelőjében megírt kód fut le, betöltődik a Síkidom Form és elvégezhetjük a szükséges számításokat.

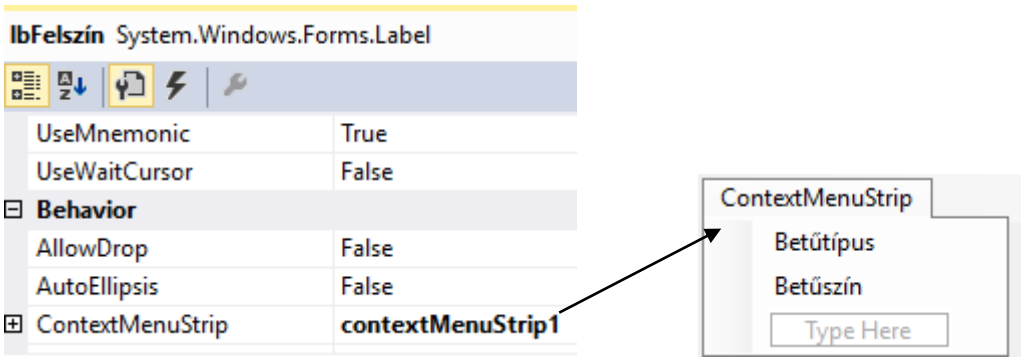


Környezeti vagy felbukkanó menü (ContextMenuStrip)

A menük egy másik elterjedt típusa a felbukkanó menük. A ContextMenu menüpontjai akkor bukkannak fel, amikor egy vezérlőre a jobb egérgombbal kattintunk.



A felbukkanó menüket a vezérlőelem ContextMenuStrip tulajdonságának a ContextMenu komponensre való beállításával kapcsolhatjuk.



Az alábbi ábrán a menüpontok Click eseménykezelőinek kódja látható.

```
private void betűtípusToolStripMenuItem_Click(object sender, EventArgs e)
{
    FontDialog ftg = new FontDialog();
    ftg.Font = lbFelszín.Font;
    ftg.ShowDialog();
    lbFelszín.Font = ftg.Font;
}

private void betűszínToolStripMenuItem_Click(object sender, EventArgs e)
{
    ColorDialog clg = new ColorDialog();
    clg.Color = lbFelszín.ForeColor;
    // Szín dialógus ablak betöltése
    clg.ShowDialog();
    lbFelszín.ForeColor = clg.Color;
}
```

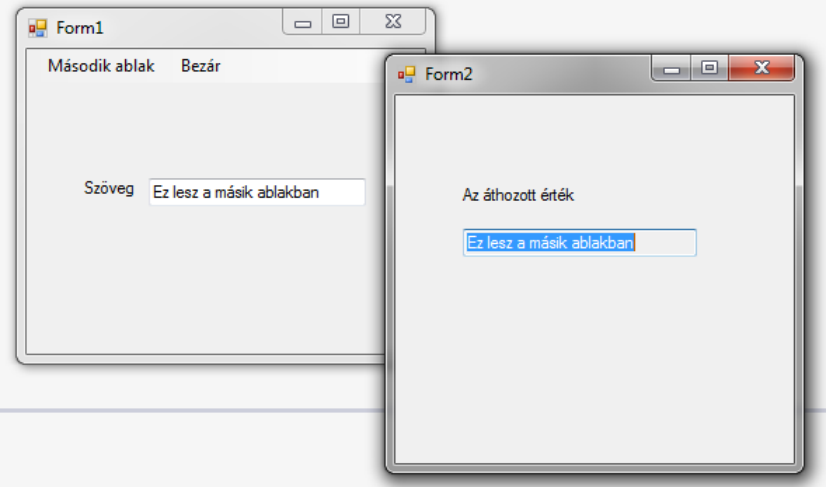
Többformos alkalmazások

Amikor megtervezzük programjainkat, különböző funkciókat más-más formon valósítunk meg. Mi történik akkor, ha az egyik formon szereplő értékre valamilyen másik formon is szükség van? Hogyan cserélhetünk adatokat a formok között? Milyen lehetőség van többformos alkalmazások létrehozására? Mi lesz az induló form? Ezek a kérdések merülnek fel, ha összetettebb feladatot kell elkészítenünk, ahol már nem elegendő egyetlen form segítségével megoldani a feladatot.

Az egyik fontos kérdésünk az volt, hogy hogyan tudunk értékeket átadni formok között? A válasz tulajdonképpen egyszerű, ugyanúgy, mint bármilyen más objektumok között, vagyis:

- tulajdonságok segítségével,
- konstruktor paraméterezésével,
- metódus paraméterezésével.

Nézzünk a probléma szemléltetésére egy igen egyszerű példát. Készítünk egy Windows Form alkalmazást, amelyben a Form1 ablak tartalmaz egy menüsort, amelynek első menüpontja a *Második ablak* nevet viseli, valamint egy TextBoxot, ahova a felhasználó beleírhat bármilyen szöveget. A feladat az, hogy ha rákattint a felhasználó a *Második ablak* nevű menüre, akkor jelenjen meg a második ablak, és mutassa az átadott értéket egy textBoxban.



Először tulajdonság segítségével adjuk át a felhasználó által beírt szöveget a második ablakba. Nézzük a megoldás kódját.

Formok közötti adatcsere – tulajdonsággal

A Form2 osztályban deklarálunk egy **BejovoAdat** nevű string típusú tulajdonságot. Ezt a tulajdonságértéket kapja meg a Form2 **Load**() eseménykezelőjében a szövegdoboz Text tulajdonsága. Mire a felhasználó elé kerül a Form2 ablak, a szövegdoboz Text tulajdonságában benne lesz ez az érték.

```
public partial class Form2 : Form
{
    public Form2() {
        InitializeComponent();
    }
    public string BejovoAdat { get; set; }

    private void Form2_Load(object sender, EventArgs e) {
        textBox1.Text = BejovoAdat;
    }

}
```

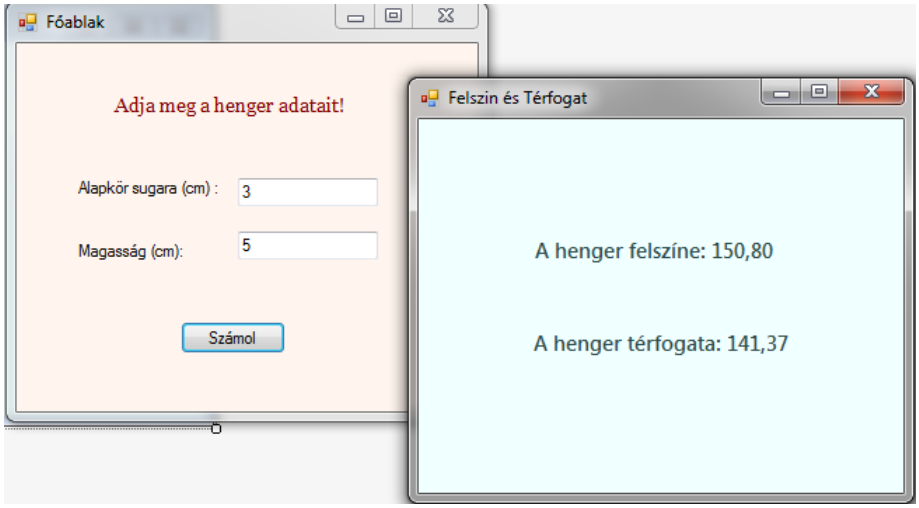
A tulajdonság értékét átadjuk a szövegdoboz Text tulajdonságának.

A Form1 osztályban a **masodikAblakMenu** nevű menuItem objektum Click eseménykezelőjében létrehozzuk a Form2 osztály egy objektumpéldányát **frmMasodik** névvel, ekkor már, mivel létezik az objektum, hozzáférünk a **BejovoAdat** tulajdonságához és átadhatjuk neki a szövegdoboz tartalmát (Text tulajdonság). Ezután meghívjuk a **frmMasodik** objektum **Show**() metódusát, ennek eredményeként megjelenik a második form ablaka a helyesen átadott értékkel.

```
public partial class Form1 : Form
{
    public Form1() {
        InitializeComponent();
    }

    private void masodikAblakMenu_Click(object sender, EventArgs e) {
        Form2 frmMasodik = new Form2(); // létrehozzuk a Form2 egy példányát
        // a form2 példány BejovoAdat tulajdonságának átadjuk
        // a Form1 szövegdobozába beírt szöveget
        frmMasodik.BejovoAdat = txtSzoveg.Text;
        //megjelenítjuk a Form2 frmMasodik nevű példányát
        frmMasodik.Show();
    }

    private void bezarMenu_Click(object sender, EventArgs e) {
        this.Close();
    }
}
```

Adatcsere formok között konstruktor paraméterezésével

Az alábbi példában egy henger felszínét és térfogatát kell kiszámítanunk, a **Számol** gomb megnyomásakor. Az induló ablak csak az adatok megadásáért felel, magát a számolást és az eredmény megjelenítését a második ablak végzi.



Ebben a feladatban is szükség van arra, hogy átadjunk adatokat egy másik formnak. Most azt a megoldást választjuk, hogy a második Form konstruktora egy paraméterlistán keresztül fogja megkapni a számoláshoz szükséges adatokat. Mivel a második ablak objektumpéldányának a létrejöttekor ismertek lesznek a számításhoz szükséges adatok, rögtön a form **Load**() eseménykezelőjében kiszámolhatjuk a felszín és térfogat értékét. Vizsgáljuk meg a megoldást:

```
namespace AblakAdatCserKonstruk
{
    4 references
    public partial class FelszinTerfogat : Form
    {
        private double sugar, magassag;
        1 reference
        public FelszinTerfogat(double sugar, double magassag)
        {
            InitializeComponent();
            this.sugar = sugar;
            this.magassag = magassag;
        }

        1 reference
        private void FelszinTerfogat_Load(object sender, EventAr
```

A FelszinTerfogat form osztályban két double típusú privát adattagot deklaráltunk. Ezek az adattagok az osztály paraméteres konstruktora segítségével kapnak értéket, az objektum létrejöttekor.

A form Load() eseménykezelője a privát adattagok értékét felhasználva kiszámolja és megjeleníti a felszín- és térfogatértéket.

```
1 reference
private void FelszinTerfogat_Load(object sender, EventArgs e)
{
    double felszin =
        sugar * sugar * Math.PI * 2 + 2 * sugar * Math.PI * magassag;
    lblFelszín.Text = "A henger felszíne: "+felszin.ToString("F2");

    lblTerfogat.Text = "A henger térfogata: " +
        (sugar * sugar * Math.PI * magassag).ToString("F2");
}
```

Az indító form **Számol** nyomógombjának **Click**() eseménykezelőjében hozzuk létre a FelszinTerfogat form egy frm nevű példányát a szövegdobozok Text tulajdonságából vett értékekkel, majd a **Show**() metódussal megjelenítjük a Felszín- és Térfogatszámoló ablakot.

```
    double sugar, magassag;
    1 reference
    private void btnSzamol_Click(object sender, EventArgs e)
    {
        sugar = double.Parse(txtSugar.Text);
        magassag = double.Parse(txtMagassag.Text);

        // létrehozzuk a FelszínTerfogat ablak egy példányát
        FelszinTerfogat frm = new FelszinTerfogat(sugar, magassag);
        frm.Show();
    }
}
```

Formok közötti adatcsere metódussal

Legyen ugyanaz a feladatunk, mint az előbb, ismét a henger felszínét és térfogatát kell kiszámítanunk. A számoláshoz szükséges input adatokat ismét az indító form szövegdobozainak Text tulajdonságától kapjuk. Tekintsük meg ennek a megoldásnak is a kódját!

A **FelszinTerfogat** form osztályban most egy **public void Szamitas()** nevű metódust készítettünk, amelynek két bemenő paramétere van. Mivel a metódus publikus, kívülről, más osztályból meghívható. A metódus a bejövő paraméterek felhasználásával kiszámolja a felszínt és a térfogatot és rögtön ki is írja a megfelelő címkékbe az eredményt.

```
public partial class FelszinTerfogat : Form
{
    1 reference
    public FelszinTerfogat()
    {
        InitializeComponent();

    }
    1 reference
    public void Szamitas(double sugar, double magassag) {
        double felszin =
            sugar * sugar * Math.PI * 2 + 2 * sugar * Math.PI * magassag;
        lblFelszín.Text = "A henger felszíne: " + felszin.ToString("F2");

        lblTerfogat.Text = "A henger térfogata: " +
            (sugar * sugar * Math.PI * magassag).ToString("F2");
    }
}
```

Az InduloAblak osztályban deklarálunk két double típusú adattagot, majd a Szamol gomb Click() eseménykezelőjében értéket adunk a két adattagnak, létrehozzuk a FelszinTerfogat form egy frm nevű objektumpéldányát a default konstruktorral. A létrejött frm objektum publikus Szamitas() metódusát ekkor már meg tudjuk hívni és elvégeztetjük vele a feladatát. Ezután meghívjuk a frm form objektum Show() metódusát, amelynek hatására megjelenik a második form, mutatva a kiszámolt eredményeket.

```
3 references
public partial class InduloAblak : Form
{
    1 reference
    public InduloAblak()
    {
        InitializeComponent();
    }
    double sugar, magassag;
    1 reference
    private void btnSzamol_Click(object sender, EventArgs e)
    {
        sugar = double.Parse(txtSugar.Text);
        magassag = double.Parse(txtMagassag.Text);

        // létrehozzuk a FelszínTerfogat ablak egy példányát
        FelszinTerfogat frm = new FelszinTerfogat();
        frm.Szamitas(sugar, magassag);
        frm.ShowDialog();

    }
}
```

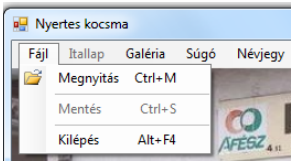
A feladatmegoldások többségében a bemutatott megoldások közül az első és a harmadik változatot alkalmazzák. Ugyanakkor lehetnek olyan szituációk, amikor már az objektum létrehozásához amúgy is szükség lenne az átadni kívánt adatokra, és később sem változnak.

3.2. Gyakorlati példa *(Achs Á.)*

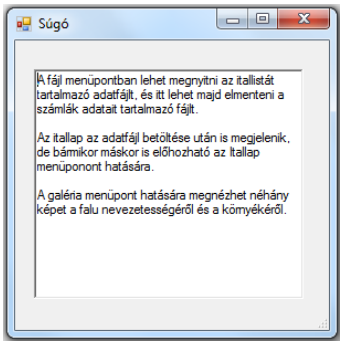
Az ELTE PPK (Pedagógiai és Pszichológiai kar) Gazdaság- és Környezetpszichológia Tanszéke 2014-ben a „vendéglátó-ipari egység fantázianév” kategória győzteseként a képen látható kocsmát jelölte meg.

(https://www.facebook.com/photo.php?fbid=513577955420949&set=a.360077537437659.1073741828.359692027476210&type=1&theater)

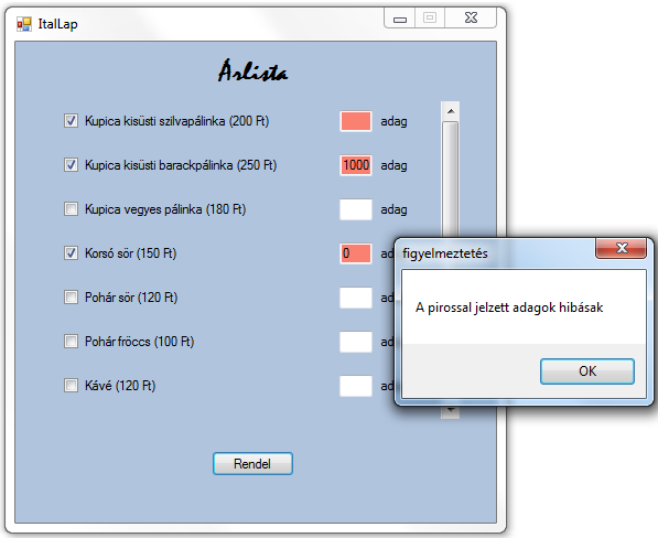
A rövid híradásból az nem derült ki, hogy kaptak-e érte valamilyen jutalmat, mi azonban ajándékozzuk meg őket egy C# programmal.



A Fájl menü a „szokásos”: meg lehet nyitni a kocsma választékát és árait tartalmazó adatfájlt. Az Itallap menüpont is és a Mentés is csak ezután válik elérhetővé. A Súgó menüpont hatására megnyíló ablakocska segít a továbbiakban:



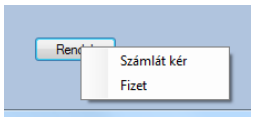
A fájl betöltése után azonnal (illetve bármikor később is az Itallap menüpontra kattintva) megjelenik az árlista. Ebből lehet rendelni. Mivel kis kocsmáról van szó, ezért egy italfajtából egyszerre legföljebb csak 999 adagot lehet kérni.



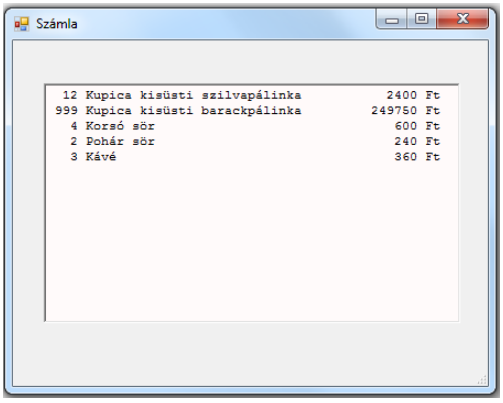
A Rendel gombra bal egérgombbal kattintva tudjuk leadni a rendelést. Ha semmi sincs kijelölve, akkor adjunk hibaüzenetet, ha valamelyik bejelölt italhoz tartozó adagot rosszul töltötték ki, akkor változzon pirosra a hozzá tartozó szövegdoboz háttérszíne, de akárhány hiba is van, csak egyetlen üzenetdoboz jelezze, hogy hibásak az adagok.

Helyesen kitöltött adatok esetén vegye fel a rendelést, és szüntessük meg a kijelöléseket, ürítsük ki a szövegdobozokat.

Ha jobb egérgombbal kattintunk a gombra, akkor jelenjen meg egy úgynevezett context menü:

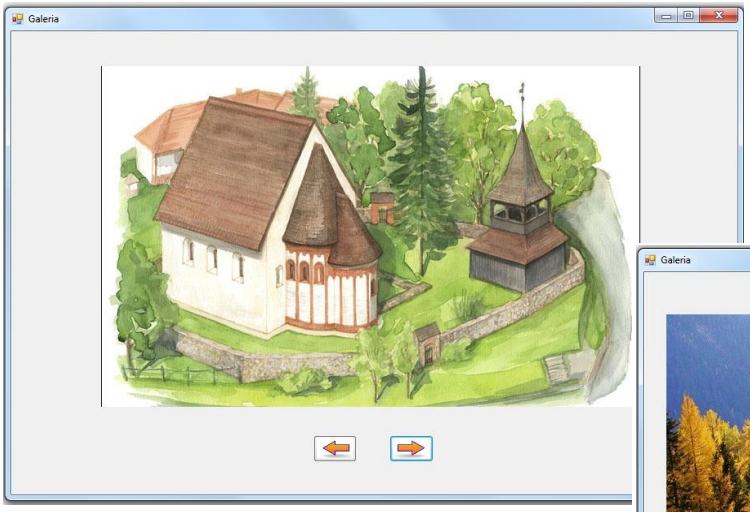


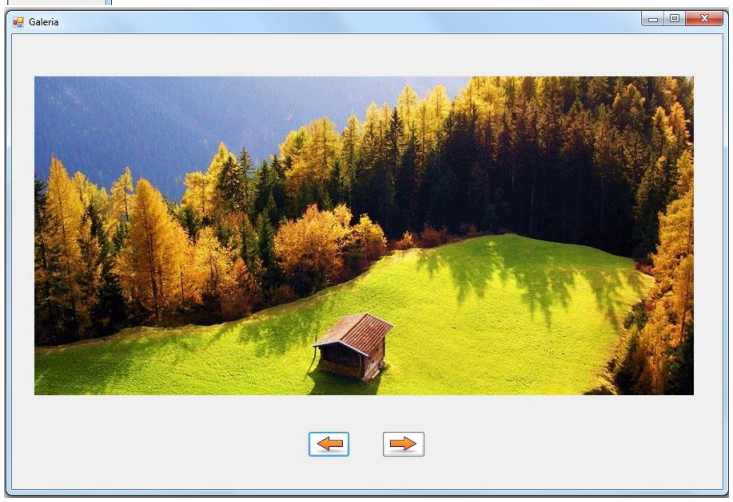
A Számlát kér menüpont hatása:
(A számla egy richTextBox-ban jelenik meg.)



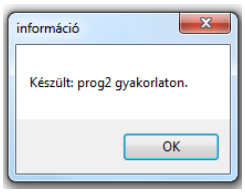
A Fizet menüpont hatására kiürül a számla, a kifizetett értékeket pedig „könyvelik” – egyelőre még csak a memóriában, a Mentés menüpont hatására kerül majd fájlba. Az eredményfájl szerkezete italfajtánként: az ital neve; összesen eladott adag; bevétel.

A Galéria menüpont hatására egy képgaléria nyílik ki, ahol lapozgatni lehet a környékről készült képek között:





A névjegy most csak egy üzenet:



Egy lehetséges megoldás néhány részlete:

1. A felület kialakítása

Form1 (de át is nevezheti):

Hogy a kép ne váljon túlzottan aránytalanná, a formot célszerű átméretezhetetlenre állítani (`FormBorderStyle`, illetve `MaximizeBox` tulajdonság beállítása).

Bár van más megoldás is, de talán legegyszerűbb, ha a képeket szintén a projekt bin/debug mappájába rakjuk, és innen választjuk ki a megfelelő `BackgroundImage` példányt. A `BackgroundImageLayout` tulajdonságot `Strech`-re célszerű állítani.

A menüpontok felrakásakor figyeljen rá, hogy a menü `Text` tulajdonságában használhat ékezetes karaktereket, az elnevezésben (`Name` tulajdonság) viszont sohasem.

Ehhez a formhoz rendeljük még hozzá az `openFileDialog1` és `saveFileDialog1` objektumokat. Mindkettő esetén állítsa be az induló mappát (`InitialDirectory`) és a fájlnevet.

A `Form1` osztály mellett szükségünk lesz még további felületekre is. Ezek a következők: `ItalLapForm` – ez tartalmazza majd az árlistát, `SzamlaForm` – erre kerül a számla, `GaleriaForm` – ezen nézegethetjük majd a képeket és a `SugoForm` – a súgó szövege számára.

ItalLapForm:

Ezt is célszerű átméretezhetetlenre állítanunk. A címet tartalmazó labelen kívül csak egy panelt tartalmaz (a kódban `pnlValasztek` néven hivatkozunk majd rá) és egy gombot – a gomb `btnRendel` néven szerepel majd a kódban.

A panelre kerülnek majd az árlistához szükséges vezérlőelemek. Mivel előre nem lehet tudni a darabszámukat, ezért a panel `AutoScroll` tulajdonságát mindenképpen `True`-ra kell állítani.

A felület látszólag ennyi, azonban van még egy fontos eleme, mégpedig a `contextMenuStrip1` objektum. Ez abban különbözik lényegesen a főmenütől, hogy alapértelmezetten nem látható, csak akkor jelenik meg, ha a hozzá tartozó vezérlőelemre kattintunk a jobb egérgombbal. A vezérlőelemet ugyanúgy rakhatjuk fel a formra, mint a főmenüt vagy a fájldialógus-elemeket, és hasonlóan lehet beállítani a tulajdonságait, de van még egy fontos teendőnk: hozzá kell rendelni a „kontextushoz”, vagyis a megfelelő vezérlőelemhez – jelenleg a `btnRendel` gombhoz. Ezt így tehetjük meg: a gomb tulajdonságai között szerepel a `ContextMenuStrip` tulajdonság is, ennek kell értékül adni a most felrakott `contextMenuStrip1` példányt.

SzamlaForm:

Összesen egyetlen richtextBox-ot tartalmaz. Neve: `rchTxtSzamla`.

Figyeljen rá, hogy csak olvashatóra állítsa, és még a tabulátort se engedje rajta parkolni.

GaleriaForm:

A rajta lévő vezérlőelemek: `pictureBox1`, `btnBal`, `btnJobb`. Mindkét gomb esetén üres a `Text`, a `BackgroudImage` pedig `Strech` módban tartalmazza a bal, illetve jobb nyilat.

A formokkal kapcsolatban nem is az az igazi kérdés, hogy milyen a felületük, hanem inkább az, hogy mitől válnak láthatóvá, illetve mi történik rajtuk.

Már most szeretnénk hangsúlyozni, hogy ezek ugyanolyan osztályok, mint amilyeneket számtalanszor használtunk. Vagyis nyilván példányosítani kell őket ahhoz, hogy használni lehessen az így létrejövő objektumokat. A konkrét példányokat tudjuk majd láthatóvá tenni, illetve ezekkel tudunk majd bármit is csinálni – ugyanúgy, ahogy a `Form1`-et is példányosítani kell ahhoz, hogy használni tudjuk.

2. Kódolás

A Fájl menü Megnyitás pontjának hatása a szokásos adatbevitel. Az adatokból `Ital` típusú példányokat hozunk létre, és hozzáadjuk őket az `italok` nevű listához.

Egyelőre maradjunk annyiban, hogy az italokat a nevük és egységáruk megadásával lehet definiálni, vagyis az `Ital` osztály ide vonatkozó részlete:

```
public string ItalNev { get; private set; }
public int EgysegAr { get; set; }

public Ital(string italNev, int egysegAr) {
    this.ItalNev = italNev;
    this.EgysegAr = egysegAr;
}
```

Megjegyzés: Ezt most csak az egyszerűség kedvéért tesszük így, egy „éles” feladatban **mindig kell** egy egyedi azonosító (mondjuk vonalkód). Ennek hiányában most a név játssza az egyedi azonosító szerepét, ezért nem lehet megváltoztatni.

Azt gondolnánk, hogy a feladatmegoldás legelső lépése az árlista beolvasása. Természetesen kezdhetjük ennek megírásával, de a végrehajtás során mégsem ez az első, hanem a képek betöltése, ugyanis a Galéria menüpontot akár hamarabb is megnyithatjuk, mint hogy „megszomjaznánk”. Persze, jogosan merül fel az az ötlet, hogy csak akkor töltsük be a képe-

ket, ha megnyitjuk a galériát. Ugyanakkor azt is végig kell gondolnunk, hogy többször is rákattinthatunk erre a menüpontra, és nem lenne túl szerencsés, ha minden egyes alkalommal ismét és ismét be kellene töltenünk a képfájlokat. Ezért jobb megoldásnak tűnik az, ha csak egyszer, a főform (`Form1`) betöltésekor töltjük be a képeket is, és majd igény esetén átadjuk a Galéria form számára.

A képeket most *kep1.jpg*, *kep2.jpg* stb. módon neveztük el, ezért nagyon egyszerű a betöltésük, azt kell csak tudni, hogy hány ilyen fájl van, a fájlneveket már könnyedén lehet generálni.

Ezek alapján a képek és az adatfájl betöltéséig így néz ki a kód:

```
public partial class Form1 : Form {

    private List<Ital> italok = new List<Ital>();
    private List<Image> kepek = new List<Image>();
    private int kepekSzama = 10;

    // Két helyről is hivatkozunk rá, ezért adattagként kell deklarálni.
    private ItalLapForm italForm = new ItalLapForm();

    public Form1() {
        InitializeComponent();
        openFileDialog1.InitialDirectory = Environment.CurrentDirectory;
        openFileDialog1.FileName = "arlap.txt";
        saveFileDialog1.InitialDirectory = Environment.CurrentDirectory;
        saveFileDialog1.FileName = "konyveles.txt";
    }

    // A képeket csak egyszer akarjuk betölteni, ezért van itt a helye.
    private void Form1_Load(object sender, EventArgs e) {
        try {
            KepBetoltes();
        }
        catch (Exception) {
            MessageBox.Show("Hiba a képek létrehozásakor", "hiba");
        }
    }

    private void KepBetoltes() {
        Image kep;
        for (int i = 1; i <= kepekSzama; i++) {
            kep = Image.FromFile("kep" + i + ".jpg");
            kepek.Add(kep);
        }
    }
```

```
// Megnyitás menüpont
private void openMenuItem_Click(object sender, EventArgs e) {
    if (openFileDialog1.ShowDialog() == DialogResult.OK) {

        StreamReader olvasoCsatorna = null;
        try {
            string fajlNev = openFileDialog1.FileName;
            olvasoCsatorna = new StreamReader(fajlNev);

            AdatBevitel (olvasoCsatorna);

            ItallapMenuItem.Enabled = true;
            SaveMenuItem.Enabled = true;
        }
        catch (Exception ex) {
            MessageBox.Show(ex.Message, "hibaüzenet a fejlesztő számára");
        }
        finally {
            if (olvasoCsatorna != null) {
                olvasoCsatorna.Close();
            }
        }
    }
}

private void AdatBevitel(StreamReader olvasoCsatorna) {

    string sor = olvasoCsatorna.ReadLine();
    string[] adatok;
    while(sor != null){
        adatok = sor.Split(';');
        // Kupica kisüsti szilva pálinka;200

        italok.Add(new Ital(adatok[0],int.Parse(adatok[1])));
        sor = olvasoCsatorna.ReadLine();
    }
}
```

Mint látható, sikeres adatbeolvasás után aktívvá válik az Itallap és a Mentés menüpont is. Az Itallap menüpont hatására meg lehet majd nyitni az árlistát.

És most eljutottunk oda, hogy hogyan lehet láthatóvá tenni az itallapot tartalmazó formot.

Mint már megbeszéltük, nyilván példányosítanunk kell az `ItalLapForm` osztályt, csak az a kérdés, hogy hol. Mivel a példányt az Itallap menüpont hatásaként is szeretnénk látni, de a fájlból való olvasás után is, vagyis a példányra két különböző metódusnak is szüksége van, ezért globálisan, vagyis adattagként kell kezelnünk, így is kell példányosítanunk (ahogy már láthatta is a korábban közölt kódrészletben):

```
// Két helyről is hivatkozunk rá, ezért adattagként kell deklarálni.
private ItalLapForm italForm = new ItalLapForm();
```

A létrehozott példányt a `Show()` vagy a `ShowDialog()` metódussal tudjuk megmutatni. A kettő között ez a különbség: a `Show()`és a `ShowDialog()` metódus is megmutatja a felületet, de a `Show()` metódus megengedi, hogy mellette használjuk a másik formot is, míg a `ShowDialog()` metódus hatására addig nem is lehet áttérni más formra, amíg ez nyitva van.

Esetünkben mindkét megoldás elképzelhető. Az első változat úgy értelmezhető, hogy egyszerre többen is adhatnak le rendelést, a második úgy, hogy egyszerre csak egy ember. Mivel ez az első találkozásunk a többformos alkalmazással, most vegyük az egyszerűbb esetet, és csak egyetlen példányt engedjünk meg, vagyis ezt: `italForm.ShowDialog();`

Még egy fontos különbség a kétféle változat között: a `Show()` metódus hatására egyszerre több azonos külalakú form is nyitva lehet, de ez nyilván azt is jelenti, hogy ugyanennyi példány jön létre az adott formból, ami a bezárás után meg is szűnik. Vagyis az így megnyitandó formokat mindig annyiszor kell példányosítani, ahányszor meg akarjuk nyitni (azaz esetünkben nem a deklaráláskor, hanem a hívó metódusban). A `ShowDialog()` metódussal megnyitott form lezárás után is életben marad.

Mindez szép és jó, csakhogy egyelőre még csak az üres formot látjuk, pontosabban csak annyit, amennyit „kézzel” ráraktunk, az italválasztékot nem. Ezt nyilván kódból kell majd felraknunk, ehhez azonban ismernünk kell az italok listáját. Igen ám, de ezt a `Form1`-ben hoztuk létre.

Semmi baj: átadjuk az `italForm` példánynak, és az `italok` lista ismeretében az majd felpakolja a vezérlőelemeket.

Ennek megfelelően az Itallap menüpont hatása:

```
// Itallap menüpont
private void ItallapMenuItem_Click(object sender, EventArgs e) {
    italForm.ArlapotIr(italok);
    italForm.ShowDialog();
}
```

Ne felejtsük el, hogy a feladat azt kérte, hogy az adatbeolvasás után közvetlenül is jelenjen meg ez a felület. Mi sem könnyebb ennél: írjuk be ezt a két sort a Megnyitás menüponthoz tartozó metódusba is. (Tény, hogy így kódismétlés jön létre, vagyis az a szebb megoldás, hogy az ismétlődés helyett külön metódust írunk, de amikor csak két sor ismétlődik, és csak kétszer, akkor meggondolandó, hogy valóban jobb-e a külön metódus.)

Ezek átgondolása után az `openMenuItem_Click()` metódus `try` blokkjának módosított változata:

```
try {
    string fajlNev = openFileDialog1.FileName;
    olvasoCsatorna = new StreamReader(fajlNev);

    AdatBevitel (olvasoCsatorna);

    ItallapMenuItem.Enabled = true;
    SaveMenuItem.Enabled = true;

    italForm.ArlapotIr(italok);
    italForm.Show();
}
```

Majd visszatérünk még a `Form1` osztályhoz, de most nézzük meg, mi történik az `ItalLapForm` osztályban.

Első körben az a kérdés, hogyan kerülnek fel az italokhoz tartozó vezérlőelemek. Ezt oldja meg az `ArlapotIr()` metódus. Hasonló feladatot oldottunk már meg, így most nem részletezzük túlzottan, csak annyit, hogy az italok listájára a rendeléskor is szükségünk van, és ugyancsak a rendelés miatt kényelmesebb, ha külön listát hozunk létre a jelölőnégyzetek és a szövegdobozok számára is.

```
public partial class ItalLapForm : Form {
    public ItalLapForm() {
        InitializeComponent();
    }

    private List<Ital> italok = new List<Ital>();
    private List<CheckBox> chkBoxok = new List<CheckBox>();
    private List<TextBox> txtBoxok = new List<TextBox>();

    private int bal = 10, fent = 10, chkXMeret = 250, chkYTav = 40,
                txtXmeret = 30, txtYmeret = 17, xKoz = 5;

    private int maxAdag = 999;

    internal void ArlapotIr(List<Ital> italok) {

        pnlValasztek.Controls.Clear();
        chkBoxok.Clear();
        txtBoxok.Clear();

        this.italok = italok;

        CheckBox chkBox;
        TextBox txtBox;
        Label lbl;

        for (int i = 0; i < italok.Count; i++) {
            chkBox = new CheckBox();
            txtBox = new TextBox();
            lbl = new Label();

            // chkBox
            chkBox.Location = new Point(bal, fent + i * chkYTav);
            chkBox.Size = new Size(chkXMeret, txtYmeret);
            chkBox.Text = italok[i].Arlistaba();
            chkBox.TextAlign = ContentAlignment.MiddleLeft;

            // txtBox
            txtBox.Location = new Point(bal + chkXMeret, fent + i * chkYTav -2);
            txtBox.Size = new Size(txtXmeret, txtYmeret);


            // lbl
            lbl.AutoSize = true;
            lbl.Location = new Point(bal + chkXMeret + txtXmeret + xKoz,
                                     fent + i * chkYTav + 1);
            lbl.Text = "adag";
```

```
            // panelre pakolás
            pnlValasztek.Controls.Add(chkBox);
            pnlValasztek.Controls.Add(txtBox);
            pnlValasztek.Controls.Add(lbl);

            // listákba pakolás
            chkBoxok.Add(chkBox);
            txtBoxok.Add(txtBox);
        }
    }
```

Megjegyzések:

1. A metódus lehetne publikus is, de ekkor (orvosolható) hibaüzenetet kapnánk, mégpedig valami ilyesmit: **Inconsistent accessibility**, vagyis baj van a láthatósággal.

Ezt vagy úgy javítjuk, hogy visszaállítjuk – az egyébként generált – `internal` láthatóságot, vagy úgy, hogy az `Ital` osztályt is publikussá tesszük (`public class`).

2. A `txtBox`, illetve `lbl` elemek helyének beállításakor a `Location()` metódus paraméterezésében látható egy-egy „misztikus” -2, ill. +1 érték. Ezeket nyugodtan ki lehet hagyni, de belenézve a Designer fájlba, hogy a visual studio hogyan generálja az egy sorba felrakott `checkBox`, `textBox` és `label` helyét, megdöbbenve láthatjuk ezeket az eltéréseket. Így kerülnek egy vonalba a hozzájuk tartozó szövegek. ☹

3. Ha érthetőbbnek – vagy legalábbis olvashatóbbnak – tartja, akkor a textBox és a label helye így is megadható:

```
txtBox.Location = new Point(bal + chkXMeret, chkBox.Location.Y -2);

lbl.Location = new Point(txtBox.Location.X + txtXmeret + xKoz, chkBox.Location.Y);
```

4. Bár még nem részleteztük az `Ital` osztály kódját, de úgy írjuk majd meg, hogy az osztály itt, az árlap kialakításakor meghívott `Arlistaba()` metódusa az árlap felületén kért formátumú stringet adja vissza. Törlésre pedig azért van szükség, hogy többszöri megnyitás esetén is csak az eredeti adatok kerüljenek a felületre.

5. Az `ArlapotIr()` metódus hívásakor a hívó fél (esetünkben a `Form1` osztály) átadja az `ItalLapForm` osztály számára a saját listapéldányát. Hogy ezt a példányt csak lokális változóként vagy mezőértékként kezeli az átvevő osztály, azon múlik, hogy csak az aktuális metódusban akarja-e használni, vagy máshol is. Ez nem igényel különösebb magyarázatot. A lista átadásával azonban baj van, mert ez sérti az egységbezárás elvét. A `Form1` osztály ugyanis avval, hogy átadja a listapéldányt másnak, azt teljes mértékben hozzáférhetővé és módosíthatóvá teszi. Ugyanakkor jelenleg ez könnyebbség, mert így ugyanazt a listát módosítja az összes form, így a módosítások könnyen nyomon követhetőek, de egyúttal ez a program sérülékenységét is eredményezheti.

Beszéljük meg a rendelést. Ennek kapcsán egyrészt végig kell gondolnunk, hogy miképp ellenőrizzük azt, hogy melyik jelölőnégyzet van bekapcsolva, illetve egyáltalán be van-e

kapcsolva valamelyik, hogyan ellenőrizzük, hogy a bekapcsolt italokhoz helyesen adtuk-e meg a rendelt mennyiséget, másrészt viszont azt, hogy mit is jelent a rendelés.

Amíg nem fizettünk, addig a korábbi rendeléshez hozzáadódik az új, fizetéskor azonban minden rendeződik, a következő vendégnek tiszta lappal kell indulnia. Ugyanakkor majd a Mentés menüpont hatására könyvelnünk is kell a teljes forgalmat és bevételt.

Ahhoz, hogy ezt meg tudjuk oldani, vissza kell térnünk az `Ital` osztályhoz, és meg kell írnunk annak kódját.

Szükségünk lesz néhány újabb adattagra: a `Mennyiseg` jelöli majd azt, hogy egy vendég mennyit rendelt. Ennek értéke a rendelés során növekszik. Az `OsszMennyiseg` az összes vendég által fogyasztott mennyiséget jelöli. Ennek értéke a fizetés során növekszik. Ekkor növekszik a `Bevetel` is, a `Mennyiseg` értéke viszont ismét visszaáll nullára.

Gondolnunk kell az italok kiíratására is. Az itallistára ki kell íratnunk az ital nevét és egység-árát. A számlára a név mellé a fogyasztott mennyiséget és a fizetendő értéket. A könyvelés fájlba pedig a nevet, összesen fogyasztott mennyiséget és a bevételt. Ez bizony háromféle string. Semmi baj, írunk három metódust. Mivel a számlának kell a legszebben formázottnak lennie, ezért az ehhez szükséges `string`-et adja majd vissza a `ToString()` metódus, a másik kettőt pedig két másik metódus.

```
class Ital {

    public string ItalNev { get; set; }
    public int EgysegAr { get; set; }
    public int Mennyiseg { get; private set; }
    public int OsszMennyiseg { get; private set; }
    public string Bevetel { get; private set; }

    public Ital(string italNev, int egysegAr) {
        this.ItalNev = italNev;
        this.EgysegAr = egysegAr;
    }

    public void Rendel(int db) {
        Mennyiseg += db;
    }

    public int Fizetendo() {
        return Mennyiseg * EgysegAr;
    }

    public void Fizet() {
        OsszMennyiseg += Mennyiseg;
        Bevetel += Mennyiseg * EgysegAr;
        Mennyiseg = 0;
    }
```

```
    public string Arlistaba() {
        return ItalNev + " (" + EgysegAr + " Ft)";
    }

    public string Konyvelesbe() {
        return ItalNev + ";" + OsszMennyiseg + ";" + Bevetel;
    }

    public override string ToString() {
        return Mennyiseg.ToString().PadLeft(4) + " " + ItalNev.PadRight(35) +
               Fizetendo().ToString().PadLeft(10) + " Ft";
    }
}
```

Térjünk vissza az `ItalLapForm` osztályhoz. A Rendelés gomb hatása:

```
private void btnRendel_Click(object sender, EventArgs e) {

    bool valasztott = false, vanHibasAdag = false;
    int mennyiseg = 0;

    for (int i = 0; i < chkBoxok.Count; i++) {
        if (chkBoxok[i].Checked) {
            valasztott = true;
            try {
                mennyiseg = int.Parse(txtBoxok[i].Text);
                if (mennyiseg <= 0 || mennyiseg > maxAdag) throw new Exception();
                italok[i].Rendel(mennyiseg);
                txtBoxok[i].BackColor = Color.White;
                chkBoxok[i].Checked = false;
                txtBoxok[i].Clear();
            }
            catch (Exception) {
                txtBoxok[i].BackColor = Color.Salmon;
                vanHibasAdag = true;
            }
        }
    }
    if (!valasztott) {
        MessageBox.Show("Semmit sem választott!", "figyelmeztetés");
    }
    else if (vanHibasAdag) {
            MessageBox.Show("A pirossal jelzett adagok hibásak", "figyelmeztetés");
    }
}
```

Bár remélhetőleg magától is érti a kódot, de röviden: ha minden stimmel, azaz kiválasztottunk egy italt, jól megadtuk a darabszámot, akkor arra az italra felveszi a rendelést, és vissza is állítja alapállapotba a jelölőnégyzetet is és a szövegdobozt is.

Ha semmit sem választottunk (azaz a `valasztott` nevű logikai változó értéke nem válik igazzá), akkor hibaüzenetet kapunk, ha pedig bármilyen baj van a kiválasztott italhoz tartozó szövegdoboz tartalmával, akkor rozsdavörössé válik a szövegdoboz háttérszíne. Ha van hibás érték, akkor itt is hibaüzenetet kapunk.

Nézzük meg a két contextMenupont hatását!

A Számlát kér menüpont hatására csak meg szeretnénk nézni a számlát. Ehhez példányosítanunk kell a `SzamlaForm`-ot (lehet helyileg is) és át kell adni neki az italok listát. Ez a lista természetesen már tartalmazza a változásokat, hiszen a rendeléskor pontosan ennek a listának az elemeihez tartozó metódusokat hívtuk meg. (Ezért kellett adattagként deklarálni a listát, és ezért volt szükség arra, hogy ez az adattag értékként fel is vegye az `ArlistatIr()` metódus paraméterében lévő listát.)

A Fizetes menüpont hatására meghívjuk az italok `Fizet()` metódusát.

```
// Számlát kér menüpont
private void szamlatKerMenuItem_Click(object sender, EventArgs e) {
    SzamlaForm szlaForm = new SzamlaForm();
    szlaForm.Kitolt(italok);
    szlaForm.ShowDialog();
}

// Fizet menüpont
private void fizetMenuItem_Click(object sender, EventArgs e) {
    foreach (Ital item in italok) {
        item.Fizet();
    }
}
```

A `SzamlaForm` osztály:

```
public partial class SzamlaForm : Form {

    public SzamlaForm() {
        InitializeComponent();
    }

    internal void Kitolt(List<Ital> italok) {
        rchTxtSzamla.Clear();
        foreach (Ital item in italok) {
            if (item.Mennyiseg != 0) {
                rchTxtSzamla.Text += item.ToString() + "\r\n";
            }
        }
    }
}
```

Megjegyzés:

Nyilván észrevette, hogy a `ToString()` metódusban használtuk a `PadLeft()`, `PadRight()` metódusokat. Ezek arra szolgálnak, hogy balról, illetve jobbról kitöltsék szóközökkel a `string` kiírása után maradt helyet, és így szépen, egymás alá formázottan lehet kiíratni őket. De bizony könnyen csalódás érhet minket. Ez a beállítás ugyanis nem működik minden betűtípusra. Sőt a legtöbb esetben nem működik. Kizárólag csak az egyenlő közű betűtípusokra alkalmazható, vagyis csak a Consolas és a Courier típusra. ☹

Fejezzük be ezt a vonalat, és beszéljük meg a `Form1` osztály Mentés menüponthoz tartozó részletét.

Ennek hatására fájlba kell írni az adatokat, mégpedig illik olyanba, amelyet később adatfájlként fel tudunk dolgozni. Ezért írtuk meg a korábban már látott formában az `Ital` osztály `Konyvelesbe()` metódusát.

```
// Mentés menüpont
private void SaveMenuItem_Click(object sender, EventArgs e) {
    if (saveFileDialog1.ShowDialog() == DialogResult.OK) {
        StreamWriter iroCsatorna = null;
        try {
            string fajlNev = saveFileDialog1.FileName;
            iroCsatorna = new StreamWriter(fajlNev);
            FajlbaIr(iroCsatorna);
        }
        catch (Exception) {
            MessageBox.Show("Hiba a fájl írásakor", "hiba");
        }
        finally {
            if (iroCsatorna != null) {
                iroCsatorna.Close();
            }
        }
    }
}

private void FajlbaIr(StreamWriter iroCsatorna) {

    foreach (Ital item in italok) {
        iroCsatorna.WriteLine(item.Konyvelesbe());
    }
}
```

Térjünk rá a Galéria menüpontra (`Form1` osztály). Ez kellően látványos eredményt adó, de egyszerű kódrészlet lesz. Nyilván létre kell hozni egy `GaleriaForm` típusú példányt és megmutatni. Most használjuk a `Show()` metódust, és ha lokálisan példányosítjuk a formot, akkor egyszerre akár több példányban is megnyithatjuk (annyiban, ahányszor meghívjuk a metódust), így egyszerre eltérő képeket is láthatunk majd, persze csak akkor, ha átadjuk a képeket a példánynak:

```
// Galéria menüpont
private void GaleriaMenuItem_Click(object sender, EventArgs e) {
    GaleriaForm galeriaForm = new GaleriaForm();
    galeriaForm.Atad(kepek);
    galeriaForm.Show();
}
```

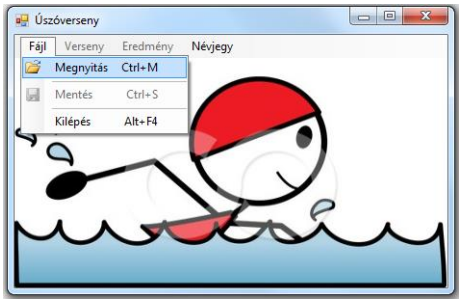
A `GaleriaForm` osztály:

```
public partial class GaleriaForm : Form {

    public GaleriaForm() {
        InitializeComponent();
    }

    private List<Image> kepek = new List<Image>();
    private int aktualisIndex;

    internal void Atad(List<Image> kepek) {
        this.kepek = kepek;
        pictureBox1.Image = kepek[aktualisIndex];
    }

    private void btnJobb_Click(object sender, EventArgs e) {
        if (aktualisIndex < kepek.Count -1) aktualisIndex++;
        else aktualisIndex = 0;
        pictureBox1.Image = kepek[aktualisIndex];
    }

    private void btnBal_Click(object sender, EventArgs e) {
        if (aktualisIndex > 0) aktualisIndex--;
        else aktualisIndex = kepek.Count - 1;
        pictureBox1.Image = kepek[aktualisIndex];
    }
}
```

A súgót, kilépést és névjegyet oldja meg önállóan.

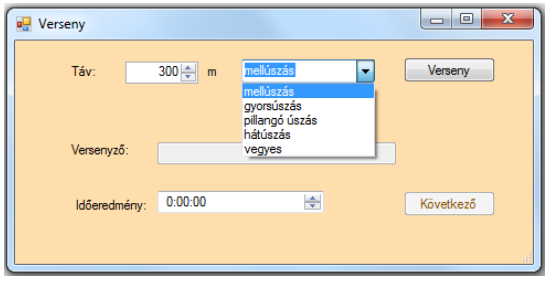
3.3. Összefoglaló feladat *(Achs Á.)*

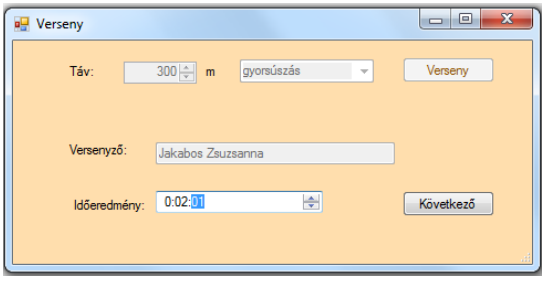
Alakítsuk át menüvezéreltté a korábban megoldott úszóversenyes feladatot (1.2.3. fejezet) – egyúttal módosítsuk is egy kicsit.



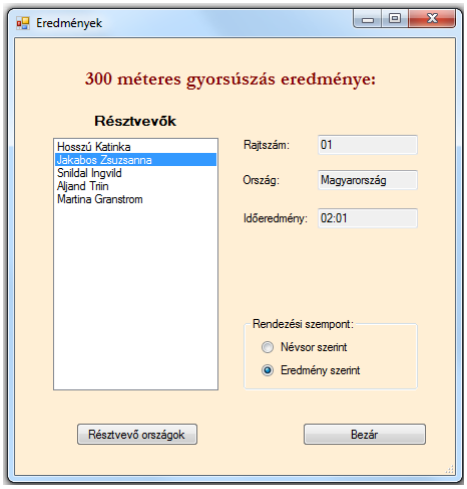
Az induló felületről először megnyithatjuk a versenyzők adatait tartalmazó fájlt. A versenyzők mindegyike automatikusan kapjon egy egyedi rajtszámot (01-től indulva egyesével).

Sikeres megnyitás után aktívvá válik a Verseny menüpont, amelyre kattintva a Verseny felületen beállíthatjuk azt, hogy milyen úszásnemről és milyen távról van szó, majd megadhatjuk az egyes versenyzők időeredményét:



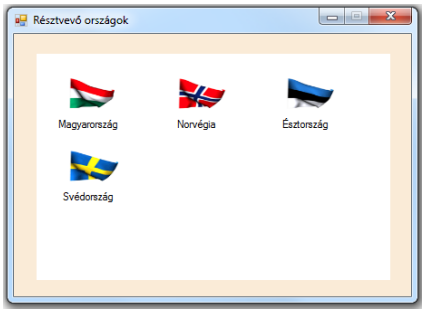


Ha az utolsó versenyző eredményét is megadtuk, akkor záródjon be az ablak, és ismét jelenjen meg az induló felület (ami legyen láthatatlan addig, amíg kitöltjük a verseny adatait), de ekkor már legyen aktív az Eredmény menüpont (és inaktív a Verseny). A menüpontra kattintva megnézhetjük a verseny eredményét. (Közben szintén ne lehessen látni az indító ablakot.)



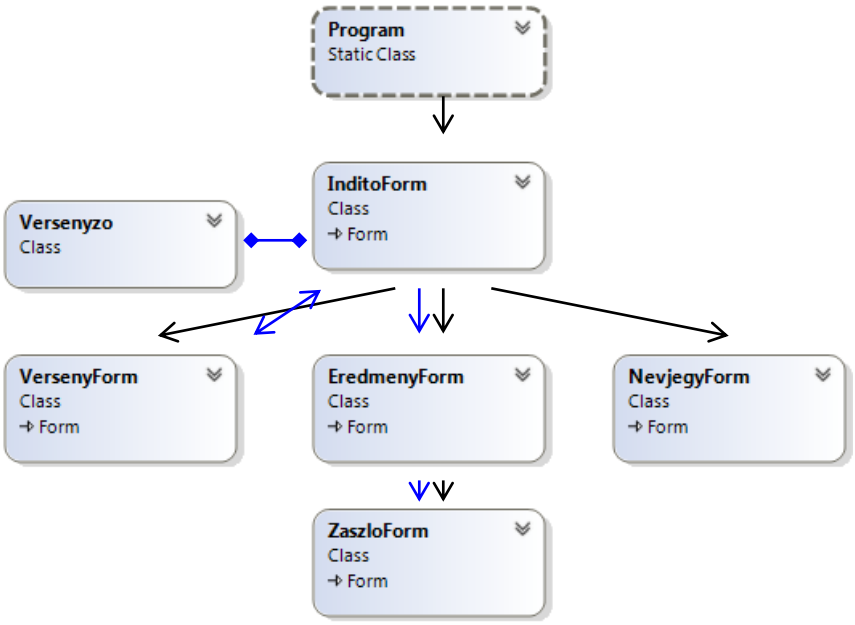
A résztvevők listájára kattintva megnézhetjük a kijelölt versenyző adatait. A lista adatai a megadott rendezési szempont alapján jelennek meg – kezdetben a regisztráció (azaz az eredeti fájlban lévő sorrend szerint), az egyéb sorrend csak valamelyik rádiógomb megnyomása után jelenik meg.

A Résztvevő országok feliratú gombra kattintva megnézhetjük a részt vevő országok zászlóit. (Mindegyik ország csak egyszer szerepel, függetlenül attól, hogy hány versenyzője van).



Egy lehetséges megoldás néhány részlete:

Először is tekintsük át a program szerkezetét!



Látható, hogy a programot hét osztály alkotja. A (nem szabványos) nyilak azt mutatják, hogy mely osztályok között van kapcsolat.

Fekete nyilak: A vezérlő osztály (`Program`) példányosítja az `InditoForm`-ot, ennek hatására nyílik meg az induló felület, és ez tartja életben egészen addig, amíg le nem zárjuk az alkalmazást.

Az `InditoForm` példányosítja majd a `VersenyForm`, `EredmenyForm` és `NevjegyForm` osztályt, és ez nyitja meg az osztályok által definiált felületeket.

Az `EredmenyForm` a `ZaszloForm` osztály példányosításáért és felületének megnyitásáért felel.

Kék nyilak: Az `InditoForm` osztályban hozzuk létre a `Versenyzo` osztály példányait, azaz itt hozzuk létre (az adatfájlból való olvasás során) a `versenyzo` példányokat és a belőlük álló `versenyzok` listát. Később az `InditoForm` továbbadja ezt a listát a `VersenyForm` számára, hiszen itt állítjuk majd be az egyes versenyzők időeredményét. Amikor végzett, a `VersenyForm` visszaadja a módosított listát az `InditoForm` példánynak, majd ez átadja az `EredmenyForm` számára, hogy az ki tudja írni az eredményeket.

Az `EredmenyForm`–nak már nem feltétlenül kellene továbbadnia a versenyzők listáját, hiszen az is megoldható lenne, hogy kigyűjti a `versenyzok` lista adatai alapján a zászlónevekből álló listát, és ezt adja tovább a `ZaszloForm`-nak, hogy az a lista alapján ki tudja rakni a zászlókat. De talán logikusabb, ha az összes, zászlókkal kapcsolatos funkciót a `ZaszloForm` osztály végzi, ezért adjuk át neki is a `versenyzok` listát.

Nagyon fontos, hogy átlássa ezt a szerkezetet (és általában, minden megoldandó feladat esetén az adott program szerkezetét). Ha ezt átlátja és érti, akkor innen kezdve már önállóan is meg tudja oldani a feladatot. Azt kell csak végiggondolni, hogy

a/ Hogyan lehet példányosítani és megmutatni egy-egy formot;

b/ Hogyan lehet átadni egyik formból a másiknak a versenyzők listáját, illetve az esetleg még szükséges adatokat.

Foglaljuk össze (és ismételjük át) ezeket:

a/ **Form példányosítása, megmutatása**

1. Mivel egy form ugyanolyan osztály, mint bármelyik, nyilván példányosítani is ugyanúgy kell. Egyedül az lehet a kérdés, hogy hol és mikor példányosítsuk. A formot valamilyen esemény hatására szeretnénk megnyitni (vagy a program indulásakor vagy egy menüpont, egy gombnyomás, egy egérkattintás stb. esemény hatására).

Egyik lehetséges megoldás, hogy akkor példányosítunk, amikor bekövetkezik ez az esemény. Ekkor minden egyes esetben, amikor bekövetkezik az esemény, létrejön egy újabb példány.

Másik lehetséges megoldás, hogy összesen egyszer példányosítunk. Ehhez az kell, hogy a formot adattagként deklaráljuk, és vagy a deklaráció során vagy a példányosító osztály konstruktorában példányosítsuk.

Hogy mikor melyik megoldást választjuk, sok mindentől függhet. Lehet olyan eset, amikor mindkét megoldás elfogadható, lehet olyan, amikor szerencsésebb az egyik vagy a másik.

Ha több metódus is akar hivatkozni ugyanarra a form példányra, akkor az adattagként való deklarációt kell választanunk. Ha nem, akkor azt kell végiggondolni, hogy okoz-e problémát az, ha esetleg párhuzamosan több példány is él a programban. Ennek végiggondolásához nem árt tudni azt, hogy hogyan mutatunk meg egy formot.

2. Formot a `Show()` vagy a `ShowDialog()` metódussal mutatunk meg. (Akinek szimpatikusabb, meg lehet mutatni a `form.Visible = true` módon is.)

A `Show()` egy `void` metódus, ha így mutatunk meg egy formot, akkor ezen is és a hívó formon is tudunk dolgozni, ekkor több hasonló formot is kinyithatunk stb. **Ha lezárunk egy `Show()` metódussal megmutatott formot, akkor megszűnik a példány is**. (Ez azt is jelenti, hogy ha egynél többször szeretnénk megnézni egy formot, akkor azt csak lokálisan lehet példányosítani, vagyis nem lehet adattagként kezelni.)

A `ShowDialog()` egy `DialogResult` típusú eseményt visszaadó metódus (olyan, mint pl. az `OpenFileDialog` osztály `ShowDialog()` metódusa). Esetünkben ez csak annyiban érdekes, hogy emiatt vizsgálni tudja, hogy ki az aktív, és nem is engedi, hogy más form legyen az, amíg ő nyitva van. Vagyis, ha így mutatunk meg egy formot, akkor ebből a formból egyszerre csak egy példány lehet nyitva, és csak ez lehet aktív. **Ha „lezárjuk”, azaz**

megnyomjuk az ablak lezáró gombját, akkor nem szűnik meg az objektum, csak láthatatlanná válik.

Bár az igazi az lenne, ha ezeket mindig végiggondolná, kellő programozói tapasztalat után így is dolgozik majd, de egyelőre nem tragédia, ha nem tudja teljes mélységben végiggondolni.

b/ **Adat átadása egyik formból a másiknak**

Ebben az ég adta világon semmi újnak nem lenne szabad lennie. Számtalanszor csináltunk ilyet: egy osztályban példányosítottunk egy másikat, és hivatkoztunk annak adataira, vagy fordítva, mi adtunk át adatokat az objektum számára. Ugyanezt csináljuk most is. Ezt háromféle tanult módon oldhatjuk meg:

1. ***Konstruktoron keresztül***.

Ezt a megoldást ugyanakkor érdemes választani, amikor máskor is konstruktoron keresztül adjuk át az adatot, vagyis akkor, ha ez az adat feltétlenül kell a form létrehozásához, illetve, ha később nem akarjuk módosítani. Aki ezt a megoldást választja, annak nagyon oda kell figyelnie arra, hogy ne feledkezzen meg a form eredeti konstruktorában generálódott `InitializeComponent()` metódusról, és ezt ne felejtse ki a módosított konstruktorból sem. (Többek között ezért sem túl szerencsés ez a megoldás.)

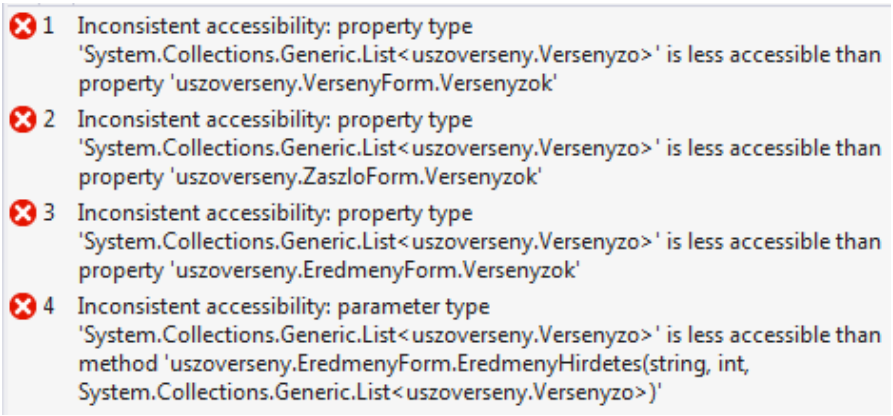
2***. Metóduson keresztül***.

Ugyanúgy működik, mint bármelyik más metódus. Összesen arra kell figyelni, hogy ezt a metódust (vagy ezeket a metódusokat) még a `Show()` vagy `ShowDialog()` metódushívás előtt aktiváljuk, különben nem látjuk a hatásukat. Még egy megjegyzés: célszerű generáltatni a metódust, mert ekkor csaknem biztos, hogy helyes lesz a paraméterátadás és a láthatóság is. (Ez a legjobb változat.)

3. ***Tulajdonságokon keresztül***.

Ehhez az kell, hogy a hivatkozott objektumnak legyenek tulajdonságai (properties), és ezekre tudjunk hivatkozni. Ha át akarunk adni értéket, akkor természetesen az kell, hogy a tulajdonságnak legyen publikus set része, ha kiolvasni, akkor pedig az, hogy legyen megfelelő get. Ez a változat azonban meglehetősen veszélyes, nagyon oda kell figyelni, hogy az esetleges programozói kényelmesség kedvéért ne tegyük kockára a program biztonságát, és ne sértsünk meg lényeges láthatósági elveket.

FONTOS: Bár nagyon fontos az osztályok láthatóságának kérdése is, ezek alapos átgondolása jó programozóktól el is várható, de ha még nem teljesen biztos ebben a témában, akkor jelenleg az is elfogadható, ha kicsit engedékenyebb. Könnyen belefuthat ugyanis ilyen jellegű hibába:



A képen látható hibaüzenetek mindegyike azért reklamál, mert a `Versenyzo` osztály kevésbé látható, mint a rá vonatkozó hivatkozás. Ennek az az oka, hogy a `Versenyzo` osztály `class Versenyzo {}` módon generálódik, a hibaüzenetben kifogásolt metódus és tulajdonság viszont publikus. A megoldás a láthatóság alapos átgondolása lenne, de most még az is elfogadható, ha a hiba kiküszöbölése kedvéért publikusnak deklaráljuk a `Versenyzo` osztályt:

```
public class Versenyzo {…
```

(Mint már említettük, ha generáltatjuk a metódusokat, akkor ilyen hiba csaknem biztos, hogy nem jön elő.)

Nagyon fontos, hogy az eddigieket jól és alaposan megértse. Ha érti, akkor viszonylag könnyedén meg tud oldani hasonló feladatokat is. Ha nem, akkor viszont nem lesz könnyű dolga. De reméljük, érti! ☺

További néhány megjegyzés

Még egy témát szeretnénk általánosságban „körbejárni”, illetve inkább néhány észrevételt tenni a kérdéssel kapcsolatban. Ez a téma: vezérlőelemek kódból való felrakása.

Ennek sem szabad az újdonság erejével hatnia, hiszen ugyanúgy tudunk hivatkozni rájuk, mint bármely másik kész osztályra. A zászlók felrakásakor például `PictureBox` típusú példányokat kell felraknunk. Ennek menete:

- Példányosítjuk a megfelelő objektumot.
- Beállítjuk a szükséges tulajdonságait (pl. hely, méret, esetleges szöveg, a hozzá tartozó kép stb.).
- Hozzáadjuk az őket tartalmazó komponens (pl. panel) `Controls` gyűjteményéhez.
- Szükség esetén hozzáadjuk egy, az adott vezérlőelemeket tartalmazó listához. Erre akkor van szükség, ha később hivatkozni szeretnénk ezekre a vezérlőelemekre.
- Ennél a zászlós feladatnál nincs ilyen, de előfordulhat, hogy eseménykezelőt is kell rendelnünk az így felrakott vezérlőelemekhez.

Egy korábbi példára utalva, pl. eseménykezelést rendelhetünk a kódból felrakott gombokhoz. Hogy átismételjük a szükséges tudnivalókat, kitérőként szerepeljen itt egy részlet. Tegyük fel, hogy nyomógombokat kell felraknunk, és a gombokhoz gombnyomás eseményt is rendelünk.

Legyen a gombok felrakásakor használt általános változó: `Button gomb;`

Ekkor a szükséges tulajdonságok beállításához az eseménykezelés beállítása is hozzátartozik: `gomb.Click += new System.EventHandler(gombhozTartozoMetodus);`

Fontos segítség:

Kérdezhetné, hogy vajon ezeket mind fejből kell tudni? Akinek jó feje van, annak persze igen, de nem valószínű, hogy sok értelme lenne ezeket bemagolni. Nem is kell, hiszen kéznél van egy kiváló segítség: ha pl. egy ilyen nyomógomb kódból való felrakását kell megoldani, akkor legegyszerűbb, ha tervezési nézetben felrak a felületre egy nyomógombot, beállítja úgy, ahogy a kódban szereplőt szeretné, és megnézi, hogy milyen kódot is generál a Visual Studio. Egy egészen minimális (de végiggondolt) átírás után meg is van a saját kódunk. (A generált kód a Form.Designer.cs nevű fájlban van.) Utána persze le lehet törölni ezt az ideiglenes gombot. ☺

Még egy kérdés: hogyan tudjuk kezelni azt, hogy épp melyik gombot nyomtuk meg, ha esetleg több gomb is szerepel a kódban?

Ezt az eseménykezeléskor meghívott `gombhozTartozoMetodus()` metódusban tehetjük meg. A metódus feje: `private void gombhozTartozoMetodus (object sender, EventArgs e) {`

Itt a `sender` az eseményt kiváltó objektumot jelenti, vagyis esetünkben azt a gombot, amelyet épp megnyomtunk.

Csakhogy a `sender object` típusú, vagyis ha `Button`-ként szeretnénk hivatkozni rá, akkor típuskényszerítést kell alkalmaznunk. Ezt kétféle módon tehetjük meg, mindegy, hogy melyiket használja, ízlés kérdése:

vagy
```
Button gomb = sender as Button;
```

vagy
```
Button gomb = (Button)sender;
```

Miután ennyire összefoglaltuk a lényeget, nem is kellene részletezni a konkrét feladat megoldását, hiszen avval önállóan is boldogulnia kellene. Nem is beszéljük meg az összes részletet, de néhányat mégis.

Kezdjük a `Versenyzo` osztállyal.

Ennek `Rajtszam`, `Nev`, `OrszagNev`, `ZaszloNev` és `IdoEredmeny` tulajdonsága van, ezek mindegyike `private set` elérhetőségű. Az `IdoEredmeny` `TimeSpan` típusú. A rajtszámon lehet vitatkozni, de mivel a feladat azt kérte, hogy ne 1, hanem 01 legyen pl. az első, no meg rajtszámhoz nemegyszer betűket is használnak, ezért ez legyen `string` típusú.

Megjegyzés: Felmerülhet a kérdés, hogy miért kell külön adatként kezelni az ország- és zászlóneveket. Ha garantáltan csak angol neveket használnánk, akkor valóban nem is kellene, viszont az országok elnevezésében szerepelhetnek ékezetes karakterek, a zászlónevek viszont egyúttal a hozzájuk tartozó képfájlok neve is, és fájlnévben soha ne használjunk ékezetes karaktereket.

A konstruktor paraméterei: név, országnév, zászlónév – az adatfájlban is ezek az adatok szerepelnek soronként, pontosvesszővel elválasztva.

Az időeredményt egy metóduson keresztül lehet beállítani:

```
public void Versenyez(TimeSpan idoEredmeny){
   this.IdoEredmeny = idoEredmeny;
}
```

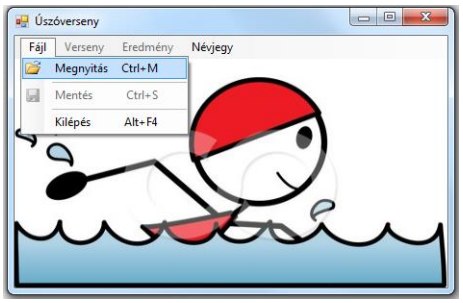
A versenyző adatainak megjelenítéséhez szükség van még a `ToString()` metódusra, ez a feladatkiírásnak megfelelően a `Nev` tulajdonságot adja vissza. (Ugyanis ez szerepel az eredmény form listadobozában.)

Egyetlen részt kell még megbeszélnünk, mégpedig az egyedi rajtszám kérdését. Az egyediséget semmiféle külső osztály nem tudja garantálni, ezt itt, a `Versenyzo` osztályon belül kell megoldani. (Amolyan auto-increment megoldás kellene.)

Különösebb indoklás nélkül közöljük a megoldást, fejtse meg:

```
private static int sorSzam;

    public Versenyzo( string nev, string orszag, string zaszlo) {
    this.Nev = nev;
    this.Orszag = orszag;
    this.Zaszlo = zaszlo;
    sorSzam++;
    this.Rajtszam = (sorSzam < 10)?("0" + sorSzam): sorSzam.ToString();
}
```

Az `InditoForm` kialakítása:



Itt arra kell figyelnie, hogy melyik menüpont aktív, melyik nem (`Enabled` tulajdonság), illetve arra, hogy megfelelően átnevezze a menüpontok `Name` tulajdonságát (annyira mindenképpen, hogy ne legyen benne ékezet). A jó változónév-adás egyébként már fél siker, nagyon sokat segít a program írásában és későbbi olvasásában is.

A Megnyitás menüpont hatására meg kell nyitnunk a már korábban felrakott `openFileDialog1` vezérlőelem segítségével a fájldialógus ablakot, és kiválasztani belőle a megfelelő fájlt. Az `openFileDialog1` beállításánál figyeljen rá, hogy a `FileName` tulajdonság vagy üres maradjon, vagy adja meg alapértelmezettként a *versenyzok.txt* fájlt, illetve arra, hogy állítsa be az `InitialDirectory` tulajdonságát.

Ha kiválasztotta a fájlt, akkor meghívhatja az adatbeviteli metódust. Figyeljen rá, és ne felejtse ki a kivételkezelést, hogy bármikor jelezhesse, ha baj van a fájlból való olvasással.

Az `AdatBevitel()` metódus a szokásos, nyilván önállóan is meg tudja oldani, most csak a versenyző példányok létrehozására vonatkozó kódrészletet közöljük:

```
string sor;
string[] tordelt;
string nev, orszag, zaszlo;

while (!olvasoCsatorna.EndOfStream) {
    sor = olvasoCsatorna.ReadLine();
    if (sor != "") {
        tordelt = sor.Split(';');
        nev = tordelt[0];
        orszag = tordelt[1];
        zaszlo = tordelt[2];
        versenyzok.Add(new Versenyzo(nev, orszag, zaszlo));
    }
```

(A ciklusban azt is kivédtük, ha netalántán üres sorokat tartalmazna az adatfájl – ez főleg a fájl végén fordulhat elő.)

A hiányzó részeket írja meg önállóan.

Az `AdatBevitel()` metódus lefutása után aktívvá kell tennünk a Verseny menüpontot, és inaktívvá a Megnyitást. (A megnyitást azért zárjuk le, mert most ez a feladatkiírás, de természetesen ez akár aktív is maradhatna, ekkor viszont meg kellene oldani, hogy a következő adatbevitel előtt törlődjön a `versenyzok` lista tartalma, vagy még jobb: dobjon fel egy választási lehetőséget, hogy az új adatokkal ki akarja-e cserélni a régit, vagy hozzájuk akarja fűzni. De ez most nem feladat.)

A Versenyzés menüpont hatására láthatóvá kell tennünk a versenyzést lebonyolító felületet, azaz példányosítanunk kell a `VersenyForm` osztályt, majd megmutatni a felületét. Mivel a formon megadott úszásnemre és távra az eredmény kiírásakor (azaz az Eredmény menüpont megnyomásakor) is hivatkozunk, ezért célszerű adattagként deklarálni a `versenyForm` objektumot. Ugyancsak így deklaráltuk az `eredmenyForm` példányt is:

```
private List<Versenyzo> versenyzok = new List<Versenyzo>();
private VersenyForm versenyForm = new VersenyForm();
private EredmenyForm eredmenyForm = new EredmenyForm();
```

A menüpont hatására tehát megmutatjuk a `versenyForm` objektum felületét, de nemcsak ezt tesszük, hanem át is adjuk neki a `versenyzok` listát, hiszen ott kell majd beállítanunk az egyes versenyzők időeredményét.

Ahogy az elején megbeszéltük, az átadási módok közül talán a metóduson keresztül való átadás a legbiztonságosabb, most is így adjuk át (ezt végzi el a `VersenyForm` osztály `Fogad()` metódusa). Ugyanakkor máris láthatunk példát a tulajdonságokon keresztül való átvételre (az átvétel mindig biztonságosabb, mint az átadás, hiszen ehhez csak a tulajdonság getterét kell publikussá tenni) és majd később a konstruktoron keresztül való átadásra is.

A `VersenyForm` osztály egyik feladata ugyanis a verseny típusának meghatározása (úszásnem és táv), ezeket a versenyform lezárásakor – azaz az induló form ismét láthatóvá tételekor máris átvehetjük, sőt tovább is adhatjuk az eredményform számára. Ez utóbbi esetben szintén metódusok segítségével adjuk át a szükséges adatokat.

Ezek után lássuk az `InduloForm` osztály `versenyMenu_Click()` metódusát:

```
private void versenyMenu_Click(object sender, EventArgs e) {
    versenyForm.Fogad(versenyzok);

    this.Hide();
    versenyForm.ShowDialog();
    this.Show();

    eredmenyMenu.Enabled = true;
    versenyMenu.Enabled = false;
    saveMenu.Enabled = true;
    int tav = versenyForm.Tav;
    string uszasNem = versenyForm.UszasNem;
    eredmenyForm.EredmenyHirdetes(uszasNem, tav, versenyzok);
}
```

A feladatkiírásnak megfelelően úgy van megoldva, hogy a `versenyForm` kibontásakor eltűnik az indító form, majd lezárása után ismét láthatóvá válik. A `versenyForm` megmutatásához célszerű a `ShowDialog()` metódust használni, hiszen egyszerre eleve csak egy verseny lehet, és addig nincs is értelme továbblépni, amíg meg nem adtuk az összes versenyző eredményét.

Az „átadó” metódusokat majd a megfelelő osztályok tárgyalásakor részletezzük.

A program megírásakor most célszerű lenne a `VersenyForm` osztály kialakításával folytatnunk, hogy minél hamarabb ki is tudjuk próbálni a munkánk eredményét. Javasoljuk, hogy így is dolgozzon, most azonban, a megoldás ismertetésekor, más utat választunk, és folytatjuk az `InduloForm` metódusaival.

Ha meg tudtuk adni a verseny típusát (úszásnem, táv), és beállítottuk a versenyzők eredményét, akkor jöhet az eredmények megmutatása. Ezt az Eredmény menüpont aktívvá válása (ld. előző kód) teszi lehetővé.

A menüpontra kattintva most az `eredmenyForm` objektumnak kell megmutatkoznia. Mivel az előbb tárgyalt metódusban már a szükséges adatokat is átadtuk a számára, most a form megmutatásán kívül nincs semmi más feladatunk:

```
private void eredmenyMenuItem_Click(object sender, EventArgs e) {
    this.Hide();
    eredmenyForm.ShowDialog();
    this.Show();
}
```

A formon címként megjelenik majd a kiválasztott úszásnem és táv, illetve egy listadobozban láthatjuk a versenyzők névsorát.

Még két további menüpont maradt, a Mentés és a Névjegy.

A mentésről volt már szó a korábbi fejezetekben, nyilván meg tudja oldani önállóan is, és nyilván nem feledkezik meg a kivételkezelésről, illetve az írócsatorna helyes lezárásáról. (Ez utóbbira különösen figyeljen, mert lezáratlan írócsatorna akár programhibát is okozhat.) De hogy tudja ellenőrizni a munkáját, közöljük a konkrét kiírást végző ciklust:

```
foreach (var versenyzo in versenyzok) {
    iroCsatorna.WriteLine(versenyzo.Rajtszam + ";" + versenyzo.Nev + ";"
                          + versenyzo.Orszag + ";" + versenyzo.IdoEredmeny);
}
```

Essen néhány szó a névjegyről is, de csak azért, hogy a konstruktorral való értékátadást is átismételhessük. A `NevjegyForm` most csak annyi, hogy egyetlen textBoxba kiírjuk a készítő nevét. Ezt a nevet most az `InditoForm`-ban adjuk meg (konstansként), hogy legyen mit átadni a másik form számára.

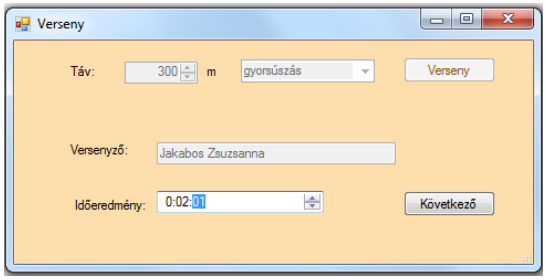
Az ide vonatkozó kódrészlet:

```
private void nevjegyMenu_Click(object sender, EventArgs e) {
    string keszito = "BitBajnok";
    NevjegyForm about = new NevjegyForm(keszito);
    about.ShowDialog();
}
```

A `NevjegyForm`:

```
public partial class NevjegyForm : Form {

    private string keszito;

    public NevjegyForm(string keszito) {
        InitializeComponent();
        this.keszito = keszito;
    }

    private void NevjegyForm_Load(object sender, EventArgs e) {
        txtNevjegy.Text = "Készítette: " + keszito;
    }
}
```

A `VersenyForm` kialakítása:



A felület kialakítása nem okozhat problémát, figyeljen rá, hogy minden vezérlőelemnek adjon rendes, beszédes nevet.

Az úszásnemeket tartalmazó comboBoxot fájlból is fel lehet tölteni, de most maradhatunk a „gyalog” feltöltésnél, azaz megadhatunk néhány konkrét úszásnemet. Célszerű úgy beállítani, hogy már induláskor is legyen egy kiválasztott úszásnem (beállíthatjuk pl. úgy, hogy a 0 legyen a kiválasztott index), illetve a dateTimePickert is beállíthatjuk úgy, hogy minden egyes versenyző esetén 0:00:00-ról induljon.

Induláskor a Verseny gomb aktív, a Következő pedig inaktív. Amikor a Verseny gombot inaktiváljuk, akkor célszerű inaktívvá tenni a comboBoxot is és a távolság megadására szolgáló numericUpDown vezérlőt is.

Megbeszéltük, hogy a versenyzők listáját, a távot és az úszásnemet másik osztálynak is ismernie kell, ezért mindháromhoz publikus getter tartozik, de kívülről közvetlenül egyik sem módosítható:

```
public List<Versenyzo> Versenyzok { get; private set; }
public string UszasNem { get; private set; }
public int Tav { get; private set; }
```

A versenyzők adatait a Fogad() metódus veszi át:

```
internal void Fogad(List<Versenyzo> versenyzok) {
    Versenyzok = versenyzok;
}
```

A `Tav` és az `UszasNem` a Verseny gomb megnyomásakor kap értéket, ekkor aktiválódik a Következő gomb, és inaktiválódik a többi.

A Következő gomb megnyomásakor a szövegdobozban megjelenik a következő versenyző neve, és beállíthatjuk az időeredményét. Itt azonban könnyen elcsúszhatnak az adatok, ha nem figyelünk. Végig kell gondolni ugyanis, hogy a megfelelő versenyzőhöz a megfelelő időt rendeljük. Emiatt célszerű lenne az a megoldás, hogy már a Verseny gomb megnyomásakor jelenjen meg az első versenyző neve, a dateTimePicker pedig álljon alapállapotba (csupa nulla), hogy ez után állíthassuk be a versenyző idejét, és a Következő gombot megnyomva a megfelelő emberhez kerüljön az időeredmény, ill. a gomb hatására jelenjen meg a következő ember neve és álljon alaphelyzetbe a dateTimePicker.

Az is kérdés, hogy hogyan tudjuk a dateTimePickerből kiolvasott `DateTime` típusú értéket `TimeSpan` típusúvá „konvertálni”. A szó azért van idézőjelben, mert nyilván nem lehet a két egészen más típust egymásba konvertálni, viszont egy dátum alapján is meg tudunk állapítani egy időintervallumot. Ennek módja például lehet az, hogy megadunk egy alapdátumot, és ezt vonjuk ki a dateTimePickerből kiolvasott értékből. Mivel csak az idő érdekel bennünket, ezért az alapdátum dátum része teljesen hasraütés-szerű is lehet:

```
private DateTime alap = new DateTime(2000, 01, 01, 0, 0, 0);
private int sorSzam;
```

(A sorszám majd a következő versenyző megadásához kell.)

Ha azt szeretnénk, hogy induláskor csupa nullát mutasson a dateTimePicker, akkor ezt például így tehetjük meg (természetesen a `dateTimePicker1` példány tulajdonságait be kell állítani úgy, hogy ne dátumot, hanem időt mutasson – Format, ShowUpDown tulajdonságok):

```
public VersenyForm() {
    InitializeComponent();
    dateTimePicker1.Value = alap;
    comboUszasNem.SelectedIndex = 0;
}
```

A Verseny gomb megnyomásakor tehát be kell állítanunk az első versenyzőt (név és dateTimePicker alapállapot), és ugyanezt kell tennünk a Következő gomb megnyomásakor is, csak ekkor már a következő versenyzőkkel.

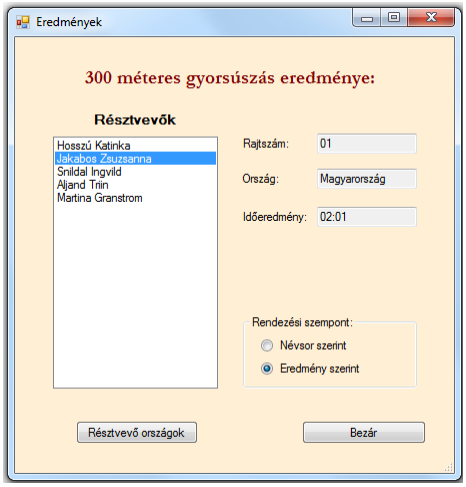
Ezért célszerű megírni egy `VersenyzoBeallitas()` metódust, amelyet mindkét gomb megnyomásakor meghívhatunk. Ha elfogytak a versenyzők, akkor záródjon be a form:

```
private void VersenyzoBeallitas() {
    dateTimePicker1.Value = alap;
    if (sorSzam < Versenyzok.Count) {
        txtVersenyzo.Text = Versenyzok[sorSzam].ToString();
    }
    else {
        this.Close();
    }
}
```

A Kovetkezo gomb hatása pedig:

```
private void btnKovetkezo_Click(object sender, EventArgs e) {
    TimeSpan idoEredmeny = dateTimePicker1.Value - alap;
    Versenyzok[sorSzam].Versenyez(idoEredmeny);
    sorSzam++;
    VersenyzoBeallitas();
}
```

Az `EredmenyForm` kialakítása



A felület kialakítása nem okozhat problémát.

Az osztálynak szüksége lesz a `versenyzok` listára, de mivel metóduson keresztül vesszük át, és senki sem akarja lekérdezni az értékét, ezért nem fontos, tulajdonságként definiáljunk (bár lehet), elég csak adattagként. Majd az `EredmenyHirdetés()` metódusban kap értéket.

Az `IditoForm`-ból hívott metódus:

```
private List<Versenyzo> versenyzok;

public void EredmenyHirdetes(string uszasNem, int tav, List<Versenyzo> versenyzok) {
    lblCim.Text = tav + " méteres " + uszasNem + " eredménye: ";
    this.versenyzok = versenyzok;
    foreach (Versenyzo versenyzo in versenyzok) {
        lstVersenyzok.Items.Add(versenyzo);
    }
}
```

A versenyzők listából való kiválasztása már szerepelt korábbi fejezetben is (mint ahogy a feladatmegoldás zöme ☺ – de nem baj, nyilván az ismétlés a cél). A hatására lefutó metódus:

```
private void lstVersenyzok_SelectedIndexChanged(object sender, EventArgs e) {
    try {
        Versenyzo versenyzo = (Versenyzo)lstVersenyzok.SelectedItem;
        txtRajtszam.Text = versenyzo.Rajtszam;
        txtOrszag.Text = versenyzo.Orszag;
        txtIdoEredmeny.Text = new DateTime(versenyzo.IdoEredmeny.Ticks).ToString("mm:ss");
    }
    catch (Exception) {
        MessageBox.Show("Hibás választás", "Hiba");
    }
}
```

A névsorba rendezés nagyon egyszerű: csak be kell kapcsolni a listadoboz `Sorted` tulajdonságát:

```
private void rdBtnNevsor_CheckedChanged(object sender, EventArgs e) {
    lstVersenyzok.Sorted = true;
}
```

De figyelem! Ez nem névsorba rendez, hanem a listadobozban lévő `string`-eket rendezi ábécé sorrendbe, azaz a bele kerülő elemek `ToString()` metódusának megfelelő `string`-ek lesznek ábécé sorrendben. Jelenleg azonban ezek pont a versenyzőnevek.

Az eredmény szerinti rendezésre több jó megoldás is lehetséges. A `versenyzo` listának van `Sort()` metódusa, de ez csak bizonyos egyéb ismeretek mellett alkalmazható:

1. Ha már találkozott vele, akkor használhatja a lambda-kifejezéssel való rendezést;

2. Ha nem, akkor felhasználhatja a `CompareTo()` metódust, amelyhez vagy írni kell egy hasonlító osztályt vagy a `Verseny` osztályt kell összehasonlíthatóvá tenni (egy interfész implementálásával).

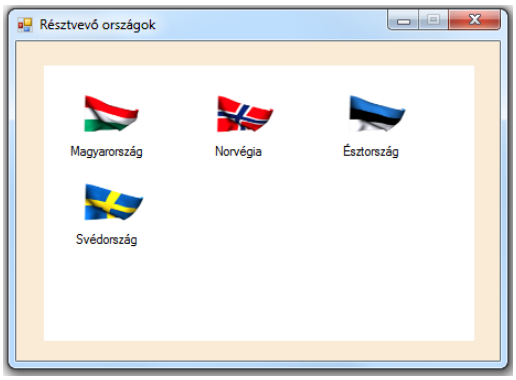
Ha ezek egyikét sem ismeri, és érdekli, akkor nézzen utána, de ebben az esetben is meg tudja írni a rendezést, hiszen tanultak rendezési algoritmusokat. Ezek közül talán a legegyszerűbb (de messze nem a leghatékonyabb) a buborékrendezés:

```
private void rdBtnEredmeny_CheckedChanged(object sender, EventArgs e) {
    lstVersenyzok.Sorted = false;
    lstVersenyzok.Items.Clear();
    Versenyzo temp;
    for (int i = 0; i < versenyzok.Count-1; i++) {
        for (int j = i + 1; j < versenyzok.Count; j++) {
            if (versenyzok[i].IdoEredmeny > versenyzok[j].IdoEredmeny) {
                temp = versenyzok[i];
                versenyzok[i] = versenyzok[j];
                versenyzok[j] = temp;
            }
        }
    }
    foreach (Versenyzo versenyzo in versenyzok) {
        lstVersenyzok.Items.Add(versenyzo);
    }
}
```

Már csak annyi van hátra, hogy meglobogtassuk a zászlókat. Ahogy a bevezetőben is megbeszéltük, ezt úgy is meg lehetne oldani, hogy itt, az `EredmenyForm`-on gyűjtjük össze őket, és csak a zászlók listáját adjuk át a `ZaszloForm`-nak, de talán logikusabb, ha minden, a zászlókkal kapcsolatos műveletet a `ZaszloForm` osztály végez. Ezért átadjuk a versenyzők listát a versenyformon már tárgyalt `Fogad()` metódushoz nagyon hasonló módon:

```
private void btnOrszagok_Click(object sender, EventArgs e) {
    ZaszloForm zaszloForm = new ZaszloForm();
    zaszloForm.Fogad(versenyzok);
    zaszloForm.ShowDialog();
}
```

A `ZaszloForm` kialakítása:



A felületre tegyünk fel egy panelt, és ehhez adogatjuk majd hozzá a képeket.

Megjegyzés: Most nem tesszük, de lehetne alkalmazni a layout menedzsereket is, amelyek segítségével könnyen hozhatnánk szép formátumúra a felületet. Ha van kedve, ideje, nézzen utána, próbálja megoldani annak segítségével is.

Különösebb magyarázat nélkül kerülnek ide a kódrészletek, fejtse meg:

```
public partial class ZaszloForm : Form {

    public ZaszloForm() {
        InitializeComponent();
    }

    private int bal = 20, fent = 20, tavX = 120, tavY = 80,
                zaszloX = 75, zaszloY = 50, labelY = 15, oszlopSzam = 3;

    private List<Versenyzo> versenyzok;
    private List<string> zaszlok = new List<string>();
    private List<string> feliratok = new List<string>();

    private void ZaszloForm_Load(object sender, EventArgs e) {
        ZaszloKigyujt();
        ZaszloFelrak();
    }

    internal void Fogad(List<Versenyzo> versenyzok) {
        this.versenyzok = versenyzok;
    }

    private void ZaszloKigyujt() {
        foreach (var versenyzo in versenyzok) {
            if (!zaszlok.Contains(versenyzo.ZaszloNev)) {
                zaszlok.Add(versenyzo.ZaszloNev);
                feliratok.Add(versenyzo.OrszagNev);
            }
        }
    }
```

```
private void ZaszloFelrak() {
    PictureBox pctBox;
    Label felirat;
    int sor = 0, oszlop = 0;

    for (int i = 0; i < zaszlok.Count; i++) {
        pctBox = new PictureBox();
        pctBox.Location = new Point(bal + oszlop * (tavX), fent + sor * (tavY));
        pctBox.Size = new Size(zaszloX, zaszloY);
        pctBox.SizeMode = PictureBoxSizeMode.StretchImage;
        pctBox.Image = Image.FromFile(zaszlok[i] + ".gif");

        felirat = new Label();
        felirat.Location = new Point(pctBox.Location.X, pctBox.Location.Y + zaszloY);
        felirat.Size = new Size(zaszloX, labelY);
        felirat.TextAlign = ContentAlignment.MiddleCenter;
        felirat.Text = feliratok[i];
        pnlKozponti.Controls.Add(pctBox);
        pnlKozponti.Controls.Add(felirat);
        oszlop++;
        if (oszlop == oszlopSzam) {
            oszlop = 0;
            sor++;
        }
    }
}
```

Megjegyzések:

1. A zászlóképek helye: A projekt bin\Debug mappája.

2. Nem túl elegáns megoldás az, hogy egymástól független listákban tároljuk a zászlót és a feliratot. Ezeket nyilván együtt kellene kezelni, vagyis az lenne az igazi, ha a zászlónév–felirat párosból létrehoznánk egy osztályt, és ilyen objektumokat tárolnánk egy listában. Ekkor azonban nem tudnánk közvetlenül alkalmazni a `Contains()` metódust (ez vizsgálja meg, hogy van-e már ilyen elem a listában), mert ehhez is meg kellene adnunk azt, hogy mi alapján tekintünk egyformának két elemet.

Ha önállóan végigcsinálta a feladatot, akkor örüljön, mert talán már elég sok mindent tud. Ha úgy csinálta végig, hogy közben olvasta ezt az útmutatót, és ez alapján rakta össze, akkor gondolja végig, hogy értette-e, amit csinál. Ha értette, akkor jó, mert remélhetőleg más hasonló feladatot is meg tud oldani, de mindenképpen próbálkozzon önálló megoldással is.

Ha elolvasta, de közben nem csinálta, akkor sajnos nem megy sokra az olvasottakkal. Csaknem biztos viszont, hogy ebben az esetben nem lehetett türelme végigolvasni. Az ilyesmit csak akkor lehet élvezni, ha csinálja is. Sok mindenhez hasonlóan, programozni sokkal könnyebb úgy megtanulni, ha örül is a munkája eredményének. Reméljük, hogy ebben a jegyzetben is örömét lelte! ☺

4. Kitekintés – továbbfejlesztési lehetőségek *(Szendrői E.)*

A jegyzet fejezeteiből megismerkedhetett az olvasó az objektumorientált programozás alapjaival, s ha együtt haladt a tananyaggal, akkor remélhetőleg sikeresen és önállóan is megoldotta a bemutatott feladatokat.

A tankönyv korlátai miatt minden létező témakört nem tudtunk érinteni. Zárszóként néhány továbblépési, továbbfejlődési lehetőségre szeretnénk felhívni a figyelmet.

A .NET Framework, ahogy a bevezető fejezetekben is olvashatták, rendkívül sok technológiai réteget tartalmaz. Lehetőséget ad arra, hogy a klasszikus Windows Form alkalmazások helyett szebb, korszerűbb grafikus megjelenítési felületeket hozzunk létre a WPF technológia alkalmazásával. Ma fontos szerepe van az internetes alkalmazásoknak, így abban is gondolkodhatunk, hogy az ASP.NET technológiával kellene bővebben megismerkedni. Azt is beláthatjuk, hogy az üzleti alkalmazások nagy adattömeggel dolgoznak, nem fájlokból nyert információkat dolgoznak fel. Ehhez az ADO.NET technológia ismeretére van szükség. Ezeket a témaköröket ebben a jegyzetben nem tudtuk részletezni.

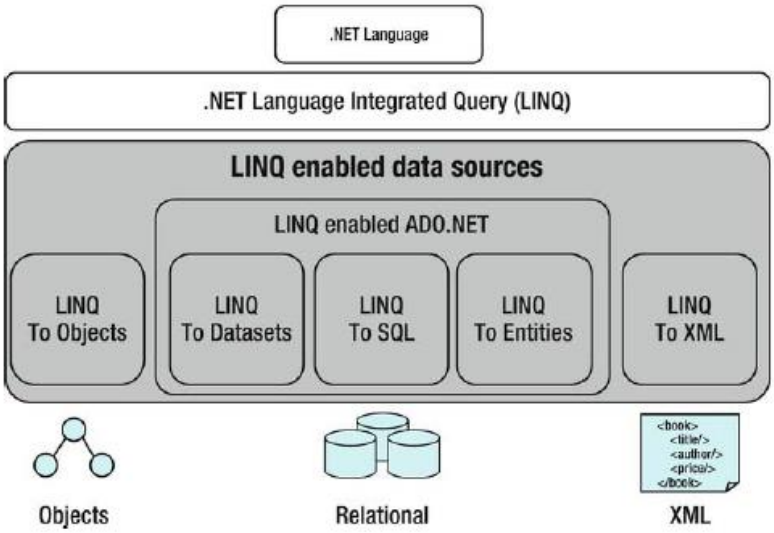
Az adatok kezelése nagyon fontos témakör, így egy kis ízelítőül röviden az adatkezelés egy fontos eszközét, a LINQ-t mutatjuk be.

A LINQ

A LINQ egy egységes programozási modellt ad a fejlesztő számára, bármilyen adatforrást is használ. Lehetőséget biztosít arra, hogy egységes módon kérdezzünk le és módosítsunk adatokat, függetlenül az adatforrástól. Segítségével SQL lekérdezéseket ágyazhatunk be a kódunkba. Röviden azt mondhatjuk, hogy a LINQ egy új adatabsztrakciós réteg.

A LINQ beépül a programozási nyelvbe és lehetővé teszi, hogy adatforrástól függetlenül, egységesen kezeljük az adatokat, vagyis pontosan úgy kezeljünk egy adatbázist, mint egy memóriában lévő gyűjteményt. Ebből az is következik, hogy a LINQ segítségével memóriában lévő gyűjteményekre vonatkozóan hasonló lekérdezéseket fogalmazzunk meg, mint a relációs adatbázisok SQL nyelve.

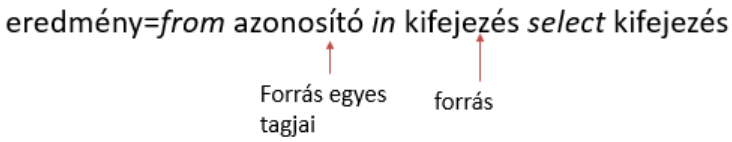
A LINQ lekérdezések erősen típusosak, vagyis a legtöbb hibát még fordítási időben el tudjuk kapni és ki tudjuk javítani.



2. ábra A LINQ architektúrája

Az 2. ábrát áttekintve, jól látszik az a törekvés, hogy bármilyen adatforrással is legyen szükséges dolgoznunk, ugyanazzal az eszközkészlettel, utasításokkal tudjuk kezelni az adatokat.

Egy LINQ lekérdezés legegyszerűbb formájában a következő alakú:

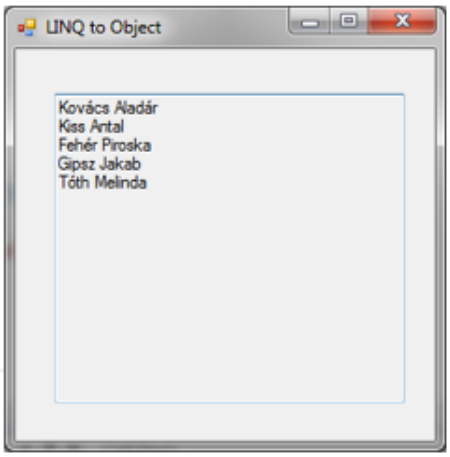


A LINQ működésének szemléltetésére egy nagyon egyszerű példát szeretnénk bemutatni.

Legyen az a feladatunk, hogy töltsünk fel egy string típusú tömböt nevekkel. Majd gyűjtsük ki a tömbből azokat az elemeket, ahol a név hossza kisebb, mint 13. Az alkalmazás ablakában egy, csak olvasható többsoros szövegdoboz van, ebben szeretnénk a kigyűjtés eredményét megjeleníteni. Mivel a tömb elemeit most az egyszerűség kedvéért a tömb deklarálásakor inicializálással adjuk meg, ezért semmilyen felhasználói interakcióra nem lesz szükségünk, s az ablak megjelenésekor már látni szeretnénk a kigyűjtés eredményét. Így a feladat megoldásának a kódját a Form Load() eseménykezelőjében adjuk meg.

A következő képen a megoldás és a futási eredmény látható.

```
public partial class Form1 : Form
{
    public Form1() {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e) {
        // string tömb
        string[] nevek = new string[] {"Kovács Aladár","Kiss Antal",
            "Fehér Piroska","Gipsz Jakab",
            "Tóth Melinda","Nagybányai Kelemen"};
        //Linq lekérdezés
        var kivNevek = from nev in nevek
                       where nev.Length <= 13
                       select nev;
        foreach (var nev in kivNevek)
        {
            txtLista.AppendText(nev + "\n");
        }
    }
}
```

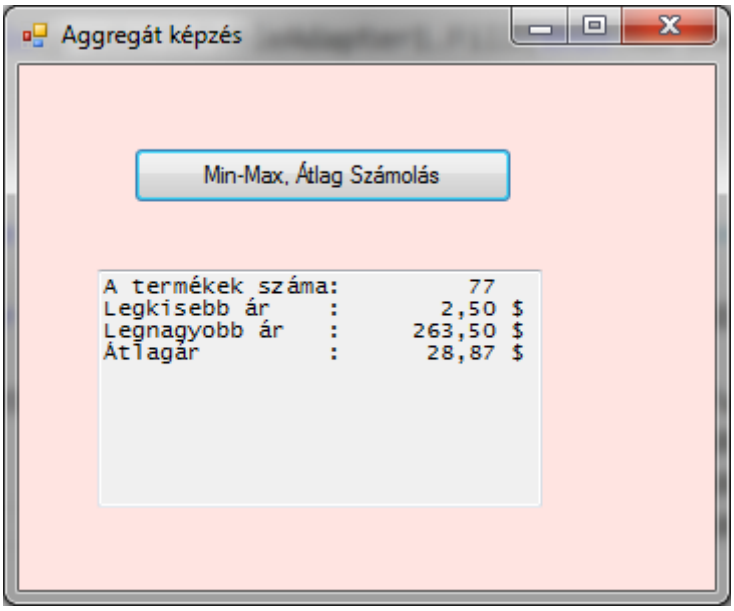



```
SELECT nev
FROM nevek
WHERE nev.Length<=13
```

A képen látható, hogy feltöltjük a tömböt, amelynek 6 eleme lesz. A LINQ utasításban megadjuk a tömb nevét mint forrást, megadjuk, hogy a forrás mely elemére van szükségünk, esetünkben a névre, de nem minden elem kerülhet be az eredményhalmazba, ezért egy where utasításrésszel megadunk egy szűrési feltételt. A feltétel szerint csak azok a tömbelemek kerülnek be az eredményhalmazba, amelyek neve rövidebb mint 13 karakter. A select utasításrészben jelezzük, hogy csak a név elemet kívánjuk feldolgozni (most nincs is más a tömbben). A szűrés eredményét a **kivNevek** eredménylistába tesszük bele. Ezután már nincs más dolgunk, mint egy foreach ciklussal kiíratni a többsoros szövegdobozba az eredményt. Látható, hogy egy név nem felelt meg a követelményeknek, Nagybányai Kelemen neve.

Az ábrán az is látható, hogy milyen SQL utasítást használtunk volna, ha az adatok egy relációs adatbázis táblájában lettek volna tárolva.

Álljon itt egy másik egyszerű példa, amelyben a jól ismert *Northwind* adatbázishoz hozunk létre kapcsolatot, majd egy többsoros, csak olvasható szövegdobozba kilistázzuk a *Products* táblában lévő termékek számát, átlagos egységárát, valamint a legnagyobb és a legkisebb egységár értéket. A program felülete legyen a következő!



A megoldásban egy úgynevezett kapcsolati stringet kell létrehozni az adatbázishoz, majd a kapcsolat megnyitása után az adatbázisszerverről letöltjük a kliens oldal memóriájába, a DataSetbe az adatokat. (A DataSet ismertetésétől most eltekintünk.) Ezután a nyomógombra kattintás eseménykezelőjében elvégezzük a szükséges számításokat és az eredményt megmutatjuk a többsoros, csak olvasható szövegdobozban.

A megoldásból csak a nyomógomb eseménykezelőjét mutatjuk meg, ebben látható a LINQ lekérdezés. A lekérdezés eredményén meghívjuk a Count(), a Min(), a Max() és az Average() LINQ függvényeket, s megfelelően formázva megjelenítjük a szövegdobozban.

```
1 reference
private void btnSzamol_Click(object sender, EventArgs e)
{
    var eredmeny = from prod in this.northwindDataSet1.Products
                   select prod.UnitPrice;
    txtEredmeny.Text = "A termékek száma:  ".PadRight(20) + eredmeny.Count().ToString().PadLeft(8) + "\r\n" +
                       "Legkisebb ár    :".PadRight(20) + eredmeny.Min().ToString("F2").PadLeft(8) + " $\r\n" +
                       "Legnagyobb ár   :".PadRight(20) + eredmeny.Max().ToString("F2").PadLeft(8) + " $\r\n" +
                       "Átlagár         :".PadRight(20) + eredmeny.Average().ToString("F2").PadLeft(8)+" $";
}
```

A Visual Studio 2015

Pár napja került forgalomba a Visual Studio 2015 fejlesztőeszköz, amely rengeteg újítást tartalmaz. Számos kényelmi szolgáltatással bővítették a kódszerkesztőt, hogy megkönnyítsék a fejlesztők munkáját a kódolástól a tesztelésig, hibakeresésig bezárólag. Az Enterprise változat új diagnosztikai eszközzel is bővült.

A leglényegesebb változás, hogy a Visual Studio Tools for Apache Cordova bővítéssel lehetővé vált, hogy a Windowstól eltérő platformokra, Android és iOS tudjunk mobil alkalmazásokat fejleszteni. A munkát új mobileszköz emulátor is segíti.

A webalkalmazások fejlesztése terén is nagy változások történtek, hiszen az ASP.NET 5 változata Open Source, nyílt forrásúvá vált, így támogatva a technológia elterjedését Linux és Mac OSX környezetben.

Az Azure WebJob modul lehetővé teszi, hogy .exe és .cmd fájlok fussanak Azure web oldalakon.

Csak néhány újítást soroltunk fel a sok közül, remélve, hogy sikerült felkelteni az olvasó érdeklődését a programozás, és azon belül a Microsoft technológiák iránt.

5. 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. Benkő Tiborné, Tóth Bertalan, *C#, .Net programozás C# nyelven*, Computerbooks, 2008
3. John Sharp, *Microsoft Visual C# 2012 Step by Step*, Microsoft Press, 2012
4. Paolo Pialorsi, Marco Russo, *Programming Microsoft LINQ*, Microsoft Press, 2008
5. Reiter István: *C# jegyzet*, leölthető: http://devportal.hu/content/CSharpjegyzet.aspx
6. Trey Nash: *C# 2008*, Panem Kiadó, Bp., 2009
7. Wouter de Kort, *Programming in C# Exam Ref 70-483*, O'Reilly Media, Inc., 2013
8. MSDN oldal: http://msdn.microsoft.com
9. Microsoft Virtual Acdemy: http://mva.microsoft.com
10. Magyar fejlesztői oldal: http://devportal.hu