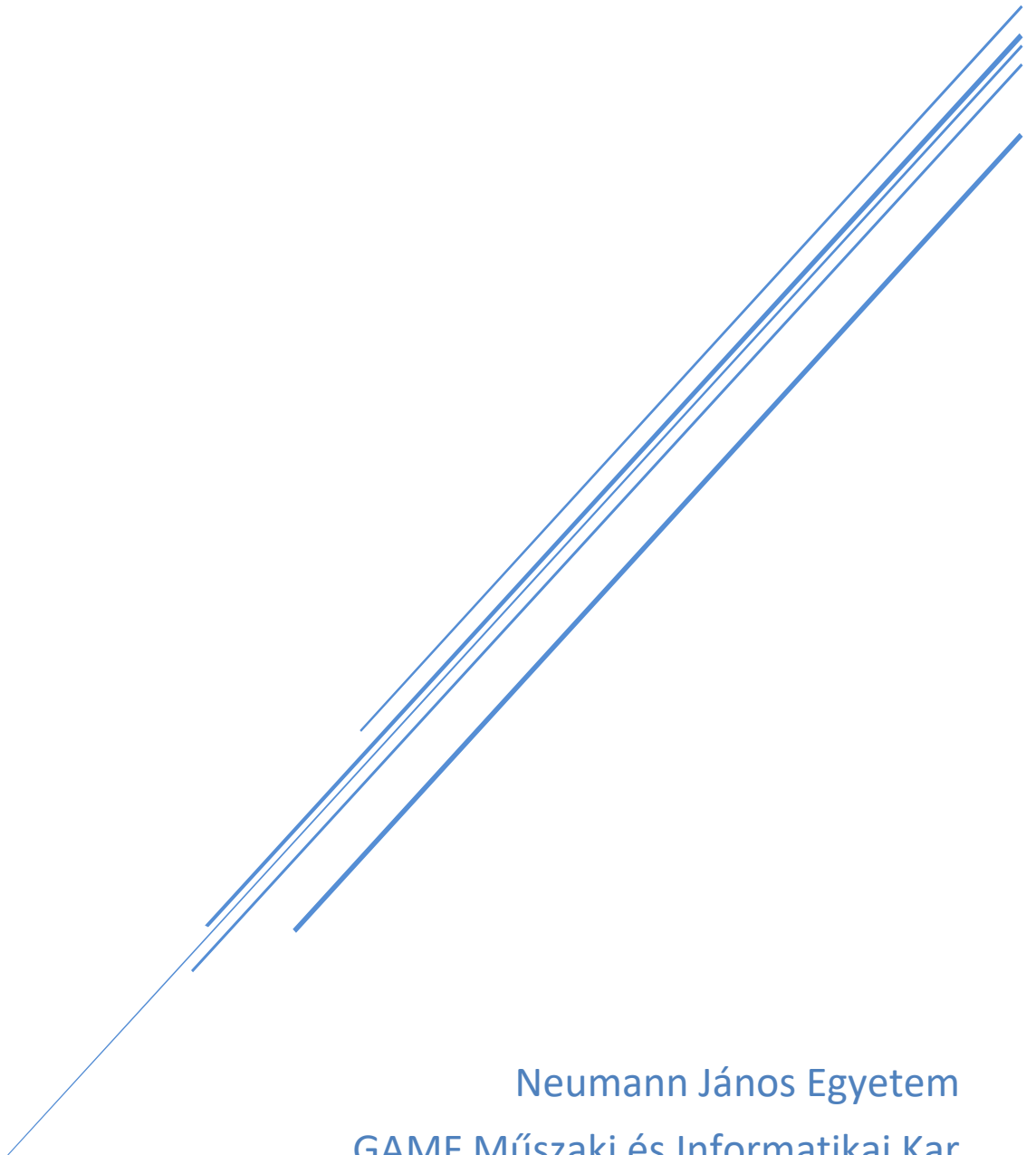


# VIZUÁLIS PROGRAMOZÁS

Gyakorlati jegyzet

Johanyák Zsolt Csaba



Neumann János Egyetem  
GAMF Műszaki és Informatikai Kar

2019

Copyright © 2008-2020 Johanyák Zsolt Csaba  
<http://johanyak.hu>  
e-mail: [johanyak.csaba@gamf.uni-neumann.hu](mailto:johanyak.csaba@gamf.uni-neumann.hu)

### **Köszönetnyilvánítás**

Köszönettel tartozom Jaczina Józsefnek, aki segítséget nyújtott a 2019-es Visual Studio-nak megfelelő frissítésekhez.

---

## Tartalomjegyzék

<b>1. Nyelvi Alapok .....</b>	<b>6</b>
<b>1.1. Komplex számok .....</b>	<b>6</b>
<b>1.2. Virtuális metódusok C#-ban.....</b>	<b>15</b>
<b>1.3. Kivételkezelés .....</b>	<b>29</b>
1.3.1. Nem kezelt kivétel .....	29
1.3.2. Kivételek kezelése .....	30
1.3.3. Kivételek előidézése .....	36
1.3.4. Jótanácsok .....	37
1.3.5. Ellenőrző kérdések .....	38
<b>1.4. Eseménykezelés – Lottójáték.....</b>	<b>38</b>
1.4.1. Események, közzetevő-feliratkozó modell .....	38
1.4.2. Feladat és főbb lépések.....	39
1.4.3. Paraméterek átadása .....	40
1.4.4. Események és eseménykezelő típusok .....	42
1.4.5. A lottóhúzást megvalósító osztály (közzetevő) .....	43
1.4.6. A játékosokat modellező osztály .....	47
1.4.7. A Program osztály és a Main metódus.....	49
1.4.8. További feladatok.....	52
<b>1.5. Sorosítás (szerializáció) és helyreállítás.....</b>	<b>53</b>
1.5.1. Bináris sorosítás és helyreállítás.....	53
1.5.2. SOAP-XML sorosítás és helyreállítás .....	53
1.5.3. XML sorosítás és helyreállítás .....	55
1.5.4. Példa .....	55
<b>2. Windows Forms .....</b>	<b>64</b>
<b>2.1. Gyümölcsárazó automata.....</b>	<b>64</b>
2.1.1. A feladat megoldása .....	64
2.1.2. A főablak grafikus felületének létrehozása .....	65
2.1.3. Eseménykezelő a Kilép nyomógombhoz .....	67
2.1.4. Asszociatív tömb a szerkesztőmező nyomógomb párosokhoz .....	67
2.1.5. Közös eseménykezelő a négy gyümölcs nyomógombhoz.....	69
2.1.6. Egységárak megváltoztatása .....	70
2.1.7. Házi feladat.....	72
<b>2.2. Ugráló Gomb .....</b>	<b>73</b>
2.2.1 Felület kialakítása .....	73
2.2.2. Adattagok definiálása.....	74
2.2.3. Kezdőérték adás a konstruktorban. ....	74
2.2.4. Az ablak fejlécében feliratot megjelenítő metódus definiálása .....	75
2.2.5. A játékot elindító Start gomb eseménykezelőjének elkészítése. ....	75
2.2.6. Eseménykezelő készítése a csúszka mozgásához .....	76
2.2.7. Eseménykezelő készítése a Kapj El! gombhoz.....	76
2.2.8. Eseménykezelő készítése a mozgásidő Tick eseményéhez .....	76
2.2.9. Eseménykezelő készítése a játékidő időzítőhöz.....	77

2.2.10. Érvénytelen találatok kiszűrése.....	77
2.2.11. Házi Feladat .....	79
<b>2.3. Képnézegető Program .....</b>	<b>80</b>
2.3.1. A feladat megoldása .....	80
2.3.2. Grafikus felület létrehozása.....	80
2.3.3. Képnevek beolvasása a program indulásakor .....	81
2.3.4. Kiválasztott kép betöltése .....	82
2.3.5. Kép megjelenítése .....	83
2.3.6. Képmegjelnítés már a program indulásakor .....	83
2.3.7. Osztálydiagram.....	84
2.3.8. Házi feladat.....	84
<b>2.4. Szöveg elhelyezése bittömbbe és kiolvasása program .....</b>	<b>84</b>
2.4.1. A feladat megoldása .....	85
2.4.2. Grafikus felület létrehozása.....	85
2.4.3. Eseménykezelő a Kilép nyomógombhoz .....	85
2.4.4. Az SzBArray osztály hozzáadása a projekthez és kisebb módosítása .....	85
2.4.5. SzBArray típusú adattag létrehozása az ablak osztályában.....	86
2.4.6. Kódolás megvalósítása .....	87
2.4.7. Kiolvasás megvalósítása .....	88
<b>2.5. Analóg óra .....</b>	<b>89</b>
2.5.1. Felület kialakítása .....	89
2.5.2. Időkezelés .....	90
2.5.3. Rajzolás (mutatók).....	91
2.5.4. Esc billentyű lenyomására kilépés a programból .....	92
2.5.5. Legyen az óra ablak kör alakú .....	92
2.5.6. Legyen mozhatható az ablak az egér segítségével.....	92
2.5.7. Névjegy panel készítése .....	94
2.5.8. Gyorsmenü készítése.....	94
2.5.9. Eseménykezelő készítése a gyorsmenühöz.....	95
2.5.10. Főablak kezdőpozíciója.....	95
2.5.11. Házi feladat.....	95
<b>3. Windows Presentation Foundation .....</b>	<b>97</b>
<b>3.1. Kétdimenziós rajzolás WPF-ben .....</b>	<b>97</b>
3.1.1. Feladat.....	99
3.1.2. Megoldás .....	99
3.1.3. Egyénileg megoldandó feladat.....	103
<b>3.2. Ugráló gomb .....</b>	<b>104</b>
3.2.1. Az alkalmazás vázának automatikus generálása: .....	104
3.2.2. Felület kialakítása: .....	104
3.2.3. Adattagok definiálása .....	105
3.2.4. Kezdőérték adás a konstruktorban .....	106
3.2.5. Eseménykezelő készítése a Tick eseményéhez. ....	107
3.2.6. Az ablak fejlécében feliratot megjelenítő metódus definiálása. ....	108
3.2.7. Eseménykezelő készítése a Kapj El! Gombhoz .....	108

3.2.8. A játékot elindító Start gomb eseménykezelőjének elkészítése.....	109
3.2.9. Eseménykezelő készítése a csúszka mozgatásához. ....	110
3.2.10. Házi feladat.....	110
<b>3.3. Képnézegető alkalmazás WPF alapú felülettel.....</b>	<b>111</b>
3.3.1. Megoldás .....	111
3.3.2. A felület elkészítése.....	111
3.3.3. A feladatot megvalósító kód .....	113
3.3.4. Eseménykezelő a mappaválasztáshoz.....	117
3.3.5. Házi feladat.....	117
<b>4. Adatbáziskezelés – Model First Entity Framework .....</b>	<b>118</b>
<b>4.1. Telefonszámok konzol alkalmazás .....</b>	<b>118</b>
4.1.1. Az Entity Framework modell és az adatbázis létrehozása.....	118
4.1.2. Adatfelvitel közvetlenül az adattáblákba .....	131
4.1.3. Adatfelvitel programból .....	132
4.1.4. Lekérdezések programból .....	137
<b>4.2. Telefonszámok WPF alkalmazás .....</b>	<b>139</b>
4.2.1. Projekt és alapbeállítások.....	139
4.2.2. A felület elkészítése.....	141
4.2.3. Egyszerű lekérdezés .....	143
4.2.4. Komplex lekérdezés.....	144
4.2.5. Helységadatok módosítása.....	149
<b>4.3. Az adatbázisban tárolt adatok lementése SQL szkriptbe az adatbázis szerkezettel együtt, majd az adatbázis újbóli létrehozása.....</b>	<b>152</b>
<b>5. Adatbáziskezelés – Code First Entity Framework .....</b>	<b>161</b>
<b>5.1. CikkAdatok WPF alkalmazás .....</b>	<b>161</b>
5.1.1. Adatbázis állomány létrehozása.....	161
5.1.2. EF osztályok elkészítése.....	162
5.1.3. Adatfelvitel .....	166
5.1.4. Egyszerű lekérdezések.....	167
<b>6. Windows Forms – Adatkötés, adatbáziskezelés.....</b>	<b>169</b>
<b>6.1. Access adatbázis elérése OLE DB-n keresztül.....</b>	<b>169</b>
6.1.1. A felhasználói felület létrehozása.....	169
6.1.2. Adatforrás megadása és a típusos adatkezelő osztályok legenerálása.....	171
6.1.3. A kód elkészítése .....	174
<b>6.2. Adatbáziselérés ODBC-n keresztül utasításokkal, C#-ban .....</b>	<b>182</b>
6.2.1. Előkészítés – Access adatbázis lemásolása, ODBC DSN létrehozása.....	182
6.2.2. Alkalmazás létrehozása .....	185
6.2.3. Házi Feladat .....	192

# 1. Nyelvi Alapok

## 1.1. Komplex számok

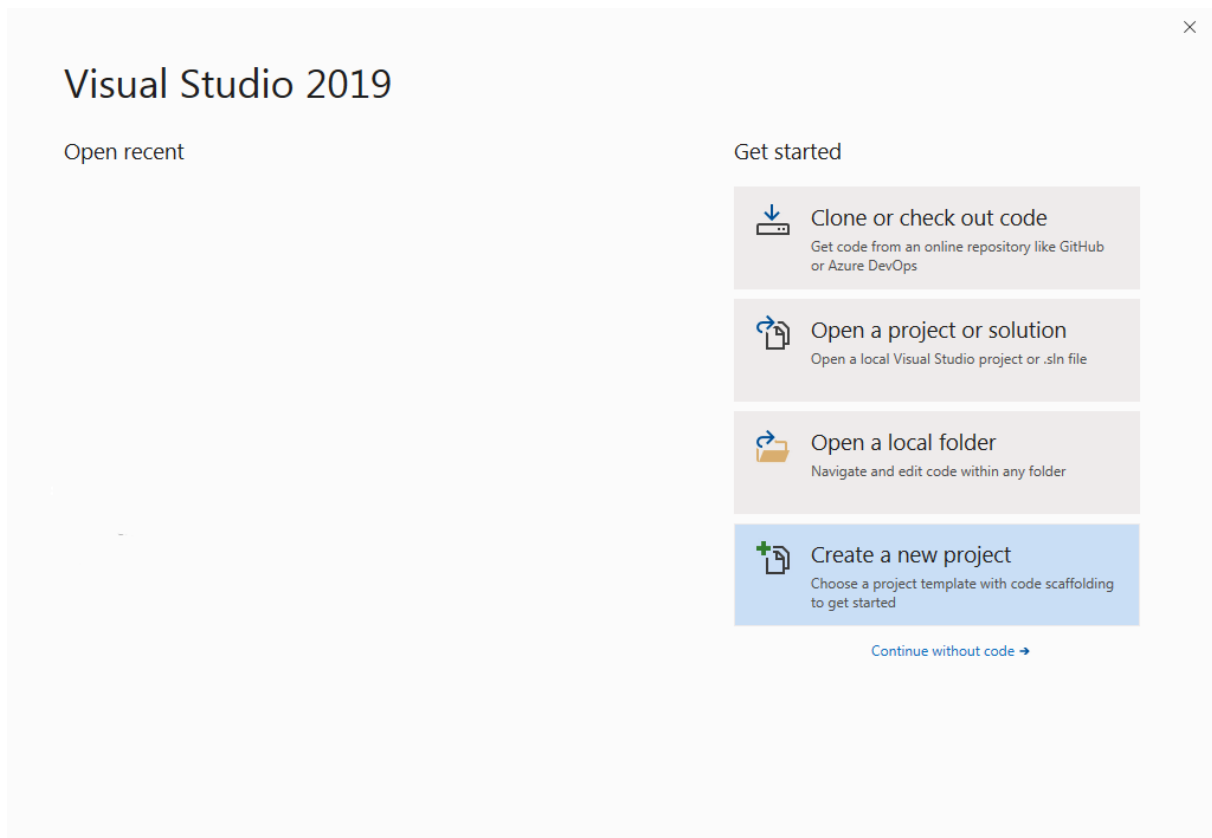
Célok:

- Ismerkedés a Visual Studio 2019 Professional fejlesztőrendszerrel (fordítás, konfigurációtípusok, néhány beállítás)
- Egyszerű konzolalkalmazás készítése vizuális eszközökkel
- Véletlenszám generálás
- ToString metódus átdefiniálása

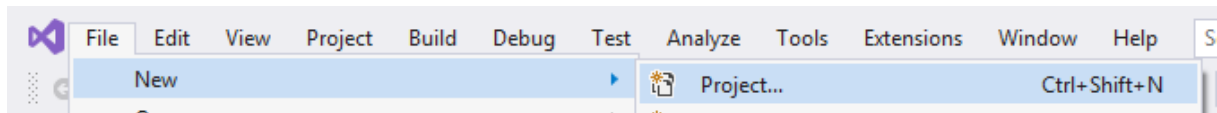
Készítsen egy Komplex számokat reprezentáló osztályt.

- Adattagok: valós, képzetes.
- Lehessen inicializálni a valós és a képzetes rész megadásával egy kétparaméteres konstruktor segítségével.
- Lehessen sztringben megkapni a komplex számot *valós+képzetes\*i* alakban (ToString metódus átdefiniálása).
- Lehessen megnövelni a komplex számot egy másik komplex számmal. Lehessen megnövelni a komplex számot egy valós számmal.
- Lehessen két komplex szám különbségét képezni. Lehessen csökkenteni a komplex számot egy valós számmal.
- A főfüggvényt tartalmazó osztályban hozzon létre két véletlenszerűen generált valós és képzetes részű komplex számot (objektumot), majd írassa ki a bennük tárolt értékeket, valamint az összegüket és a különbségüket. Írassa ki egy komplex szám és egy valós szám összegét és különbségét.
- Hozzon létre egy tömböt, amelynek elemei Komplex típusúak, és töltse fel a tömböt véletlenszerűen előállított értékekkel.
- Írassa ki a tömb elemeit.
- Adja össze a tömb elemeit, és írassa ki az eredményt.

Minden alkalmazáshoz projektet kell készíteni. Ennek menete a következők lehetnek, vagy az indításkor választjuk ki a "Create a new project"-et (1. ábra) vagy pedig a "File" menüpont alatt találjuk (2. ábra):

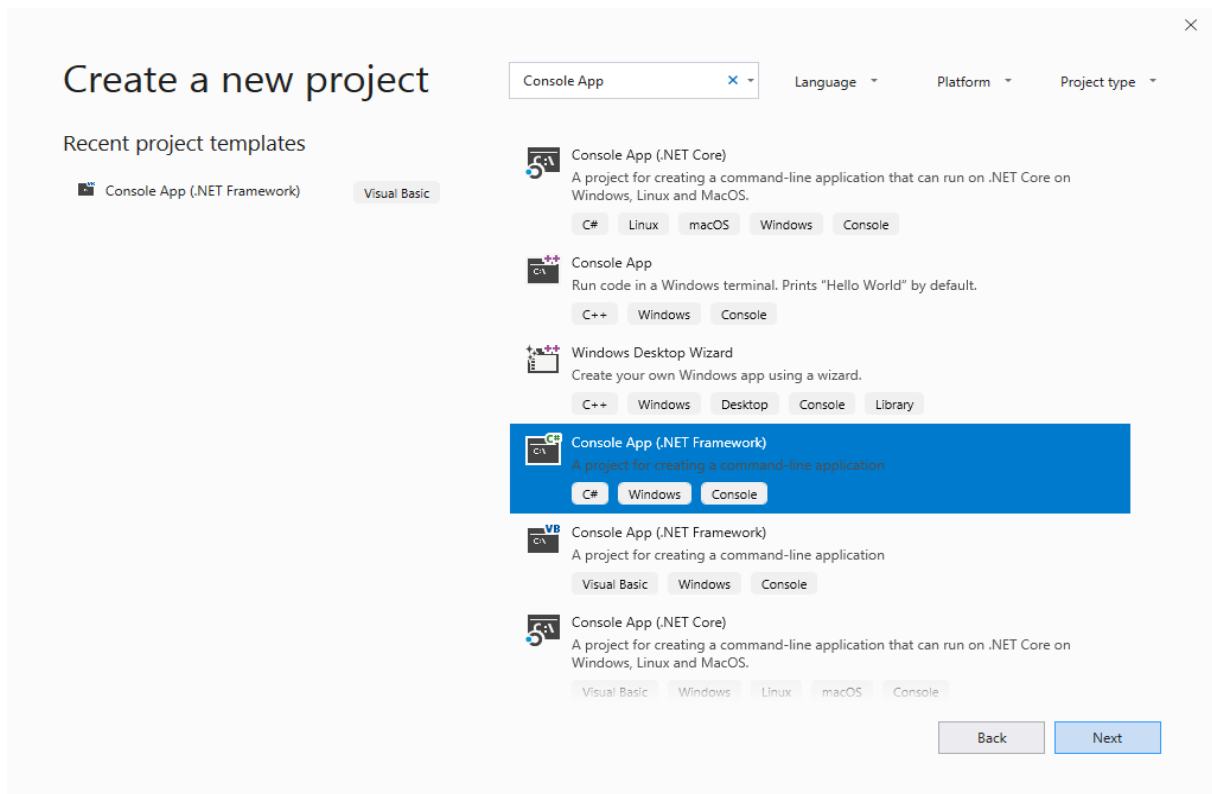


1. ábra



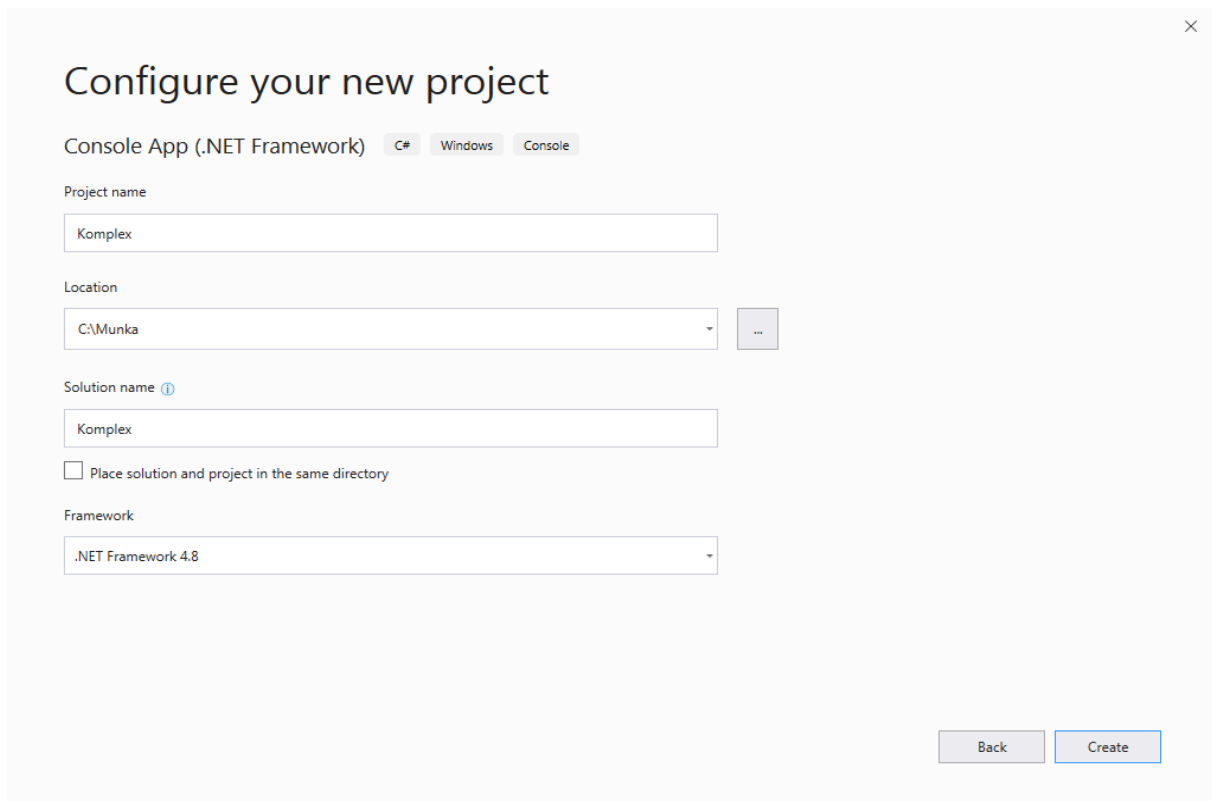
2. ábra.

A mai gyakorlaton konzolalkalmazást készítünk (3. ábra).



3. ábra

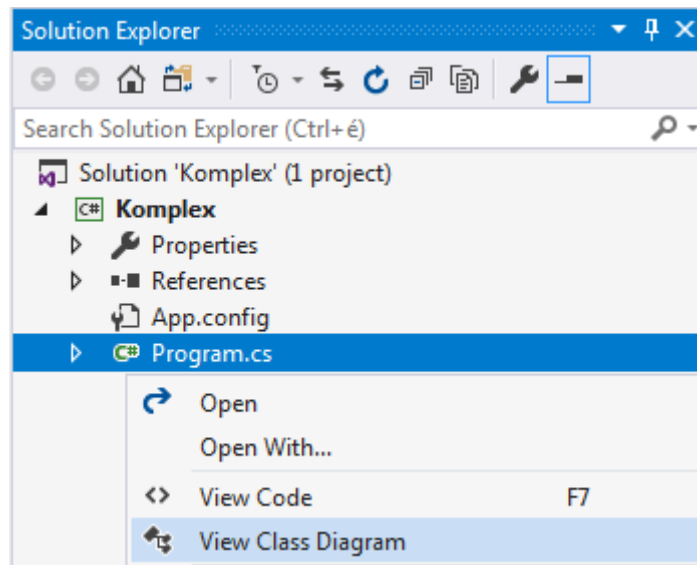
Ha a fejlesztőkörnyezet úgy van beállítva, hogy megadhatjuk a projekt helyét, akkor a C:\munka könyvtárban belül helyezzük el azt (4. ábra).



4. ábra.

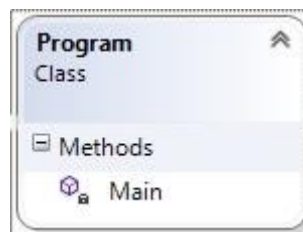


Az UML-szerű osztálydiagram elkészítése és megjelenítése



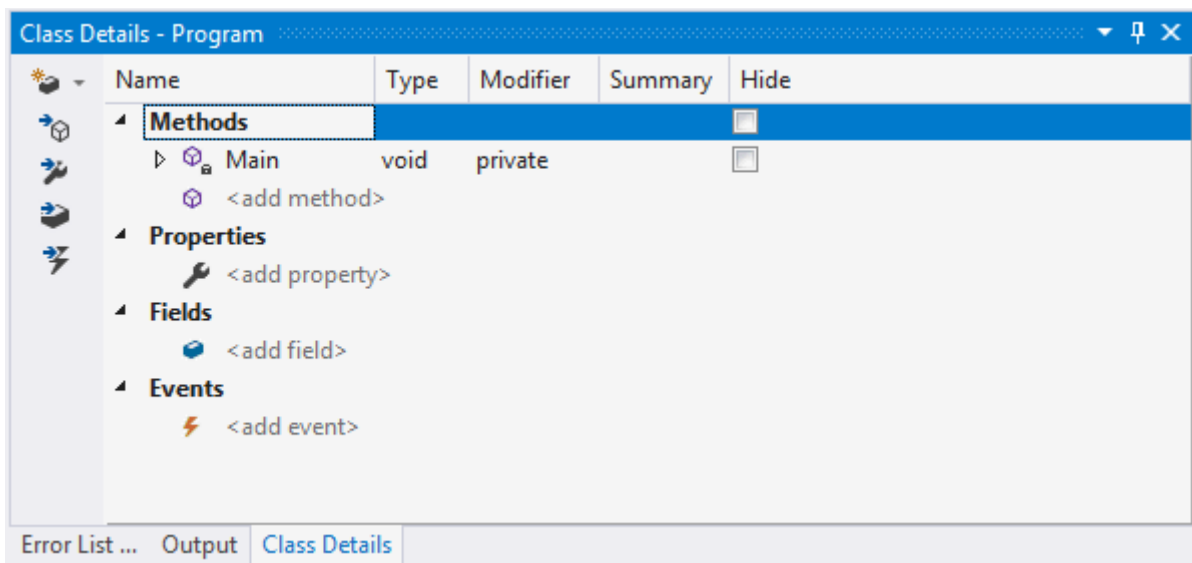
5. ábra.

Kezdetben egy osztályunk van, ami a főfüggvényt tartalmazza:



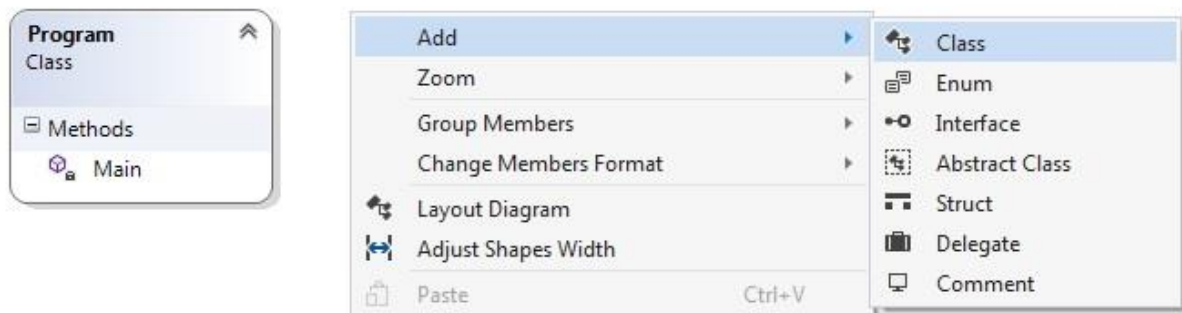
6. ábra.

A Class Details ablakban olvashatjuk el az osztály tagjaira vonatkozó információkat, illetve itt vehetünk fel újabb elemeket az osztályba. Először válasszuk ki az osztályt a diagramon.



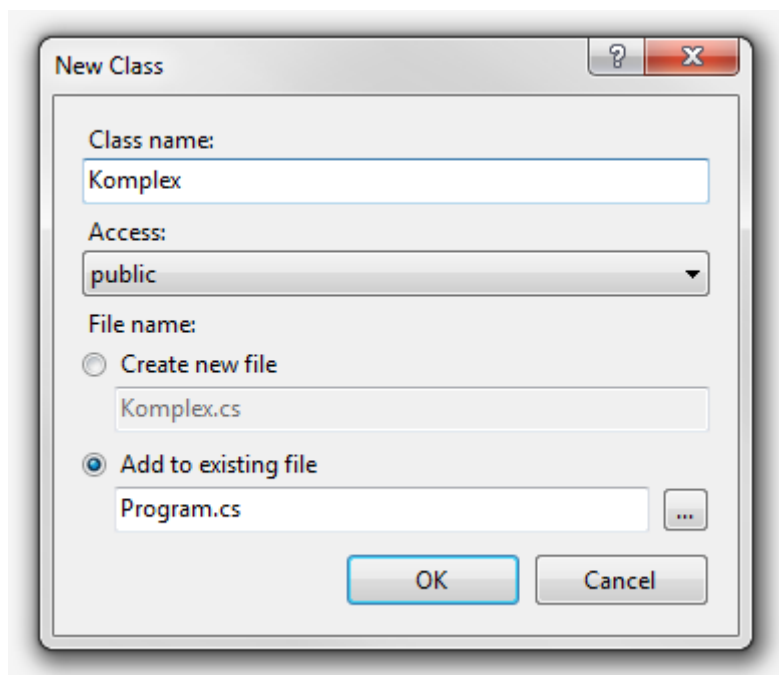
7. ábra.

Új osztály létrehozása a komplex számok számára:



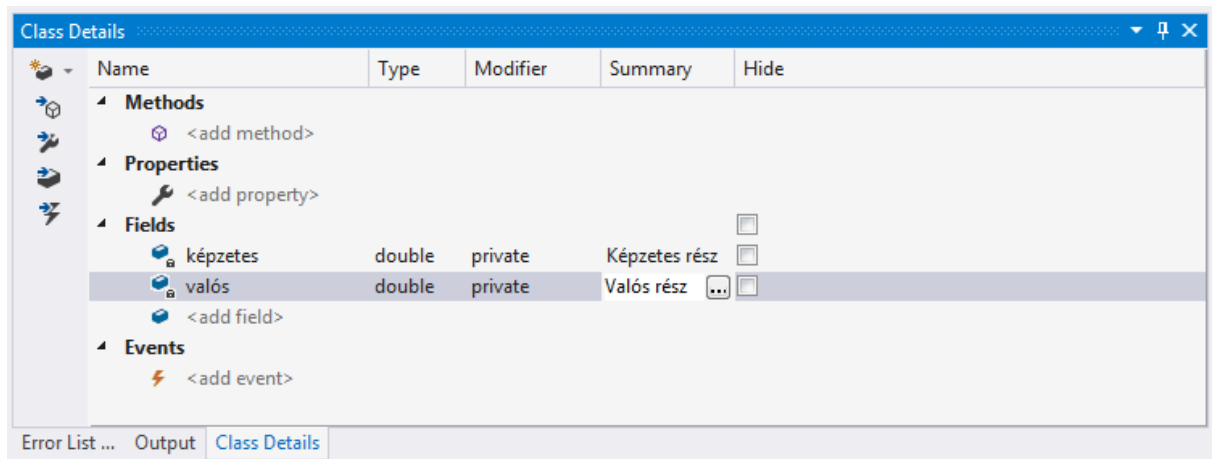
8. ábra.

Az új osztályt is az aktuális állományba helyezük el.



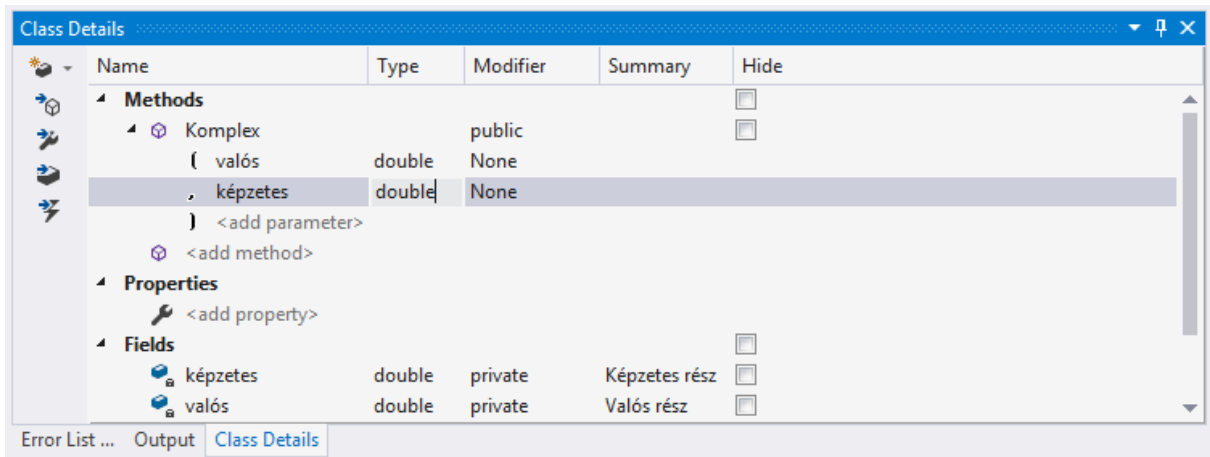
9. ábra.

Adattagok létrehozása a Fields részben:



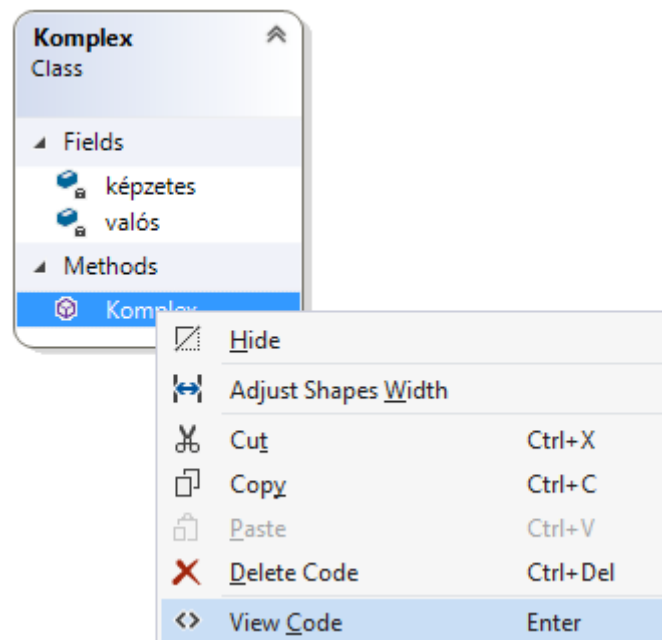
10. ábra.

Konstruktor definiálása:



11. ábra.

Áttérés kód nézetbe:



12. ábra.

Az automatikusan létrehozott metódus váz, ami kezdetben csak egy kivétel előírdézést tartalmaz:

```
public Komplex(double valós, double képzetes)
{
    throw new System.NotImplementedException();
}
```

Utasítások:

```
public Komplex(double valós, double képzetes)
{
    this.valós = valós;
    this.képzetes = képzetes;
}
```

Az object őszosztályban definiált ToString() metódus eredetileg az osztály nevét adja vissza egy sztringben. Ehelyett mi azt szeretnénk, hogy egy olyan sztringet definiáljon, ami *valós+képzetes\*i* alakban tartalmazza az objektumban tárolt adatokat.

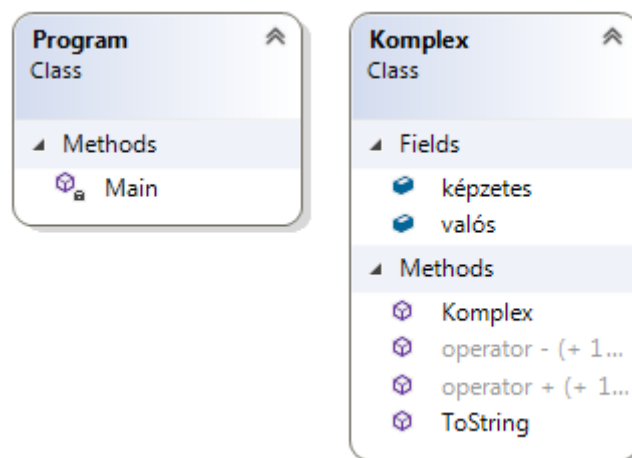
Örökölt ToString() átdefiniálásához a Komplex osztályban egy új ToString() metódust hozunk létre az override jelzővel:

```
public override string ToString()
{
    return string.Format("{0,8:F4}+i*{1,8:F4}", valós, képzetes);
}
```

A komplex szám +/- komplex szám és a komplex szám +/- valós szám összegek/különbségek számításához átdefiniáljuk a megfelelő operátorokat. Az átdefiniált operátorok az alábbiak lesznek:

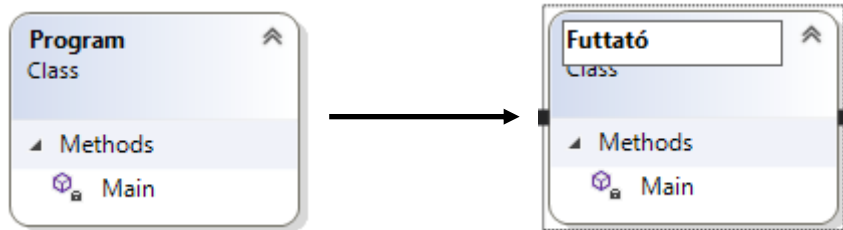
```
public static Komplex operator +(Komplex a, Komplex b)
{
    return new Komplex(a.valós + b.valós, a.képzetes + b.képzetes);
}
public static Komplex operator +(Komplex a, double b)
{
    return new Komplex(a.valós + b, a.képzetes);
}
public static Komplex operator -(Komplex a, Komplex b)
{
    return new Komplex(a.valós - b.valós, a.képzetes - b.képzetes);
}
public static Komplex operator -(Komplex a, double b)
{
    return new Komplex(a.valós - b, a.képzetes);
}
```

Visszatérve az osztálydiagramhoz:



13. ábra.

A program osztály átnevezéséhez válasszuk ki a Program osztályt (egy kattintás), majd kattintsunk egyszer a diagramon a Program feliraton. Ha dupla kattintást hajtunk végre, akkor kód nézetben szerkeszthetjük a programot.

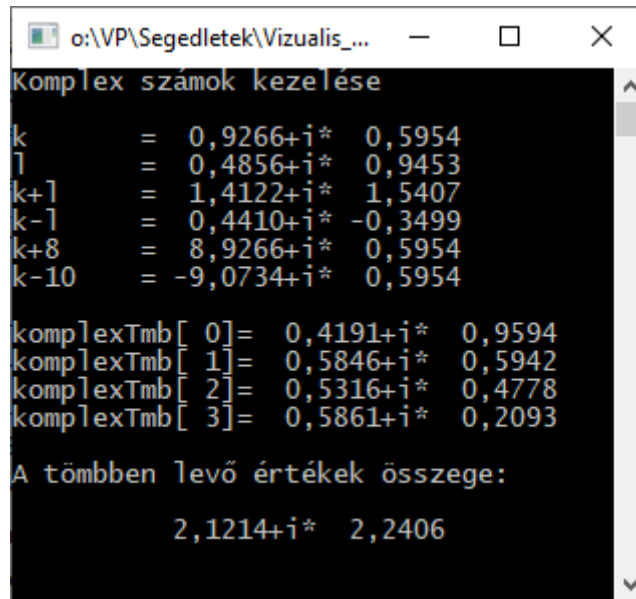


14. ábra.

A főfüggvény kódját kézzel kell beírni az automatikusan generált függvényvázba:

```
//Futtató osztály
class Futtató
{
    static void Main(string[] args)
    {
        Console.WriteLine("Komplex számok kezelése\n");
        Random r = new Random();
        Komplex a = new Komplex(r.NextDouble(), r.NextDouble());
        Komplex b = new Komplex(r.NextDouble(), r.NextDouble());
        Console.WriteLine("k\t={0}", a);
        Console.WriteLine("l\t={0}", b);
        Console.WriteLine("k+l\t={0}", a + b);
        Console.WriteLine("k-l\t={0}", a - b);
        Console.WriteLine("k+{0,-6}={1}", 8, a + 8);
        Console.WriteLine("k-{0,-6}={1}\n", 10, a - 10);
        Komplex[] komplexTmb = new Komplex[4];
        Komplex összeg = new Komplex(0, 0);
        for(int i = 0; i < komplexTmb.Length; i++)
        {
            komplexTmb[i] = new Komplex(r.NextDouble(), r.NextDouble());
            Console.WriteLine("komplexTmb[{0,2}]= {1}", i, komplexTmb[i]);
            összeg = összeg + komplexTmb[i];
        }
        Console.WriteLine("\nA tömbben levő értékek összege:\n\n{0,27}", összeg);
        Console.ReadLine();
    }
}
```

A program futásának eredménye:



```
o:\VP\Segedletek\Vizualis_...
Komplex számok kezelése
k      = 0,9266+i* 0,5954
l      = 0,4856+i* 0,9453
k+l    = 1,4122+i* 1,5407
k-l    = 0,4410+i* -0,3499
k+8    = 8,9266+i* 0,5954
k-10   = -9,0734+i* 0,5954

komplexTmb[ 0]= 0,4191+i* 0,9594
komplexTmb[ 1]= 0,5846+i* 0,5942
komplexTmb[ 2]= 0,5316+i* 0,4778
komplexTmb[ 3]= 0,5861+i* 0,2093

A tömbben levő értékek összege:
      2,1214+i* 2,2406
```

15. ábra.

### Feladat:

Tanulmányozza át a RacionalisSzamok könyvtárban levő projektet.

Érdekesebb elemek:

- Egyik konstruktor meghívja a másikat. Alapelv: egy kódrészlet csak egy példányban szerepeljen a kódban.
- Egyenlőségvizsgálat érdekében definiálni kell az == operátort, át kell definiálni az Equals metódust.
- Relációs operátorok csak párban definiálhatók át: == és !=, < és >

## 1.2. Virtuális metódusok C#-ban

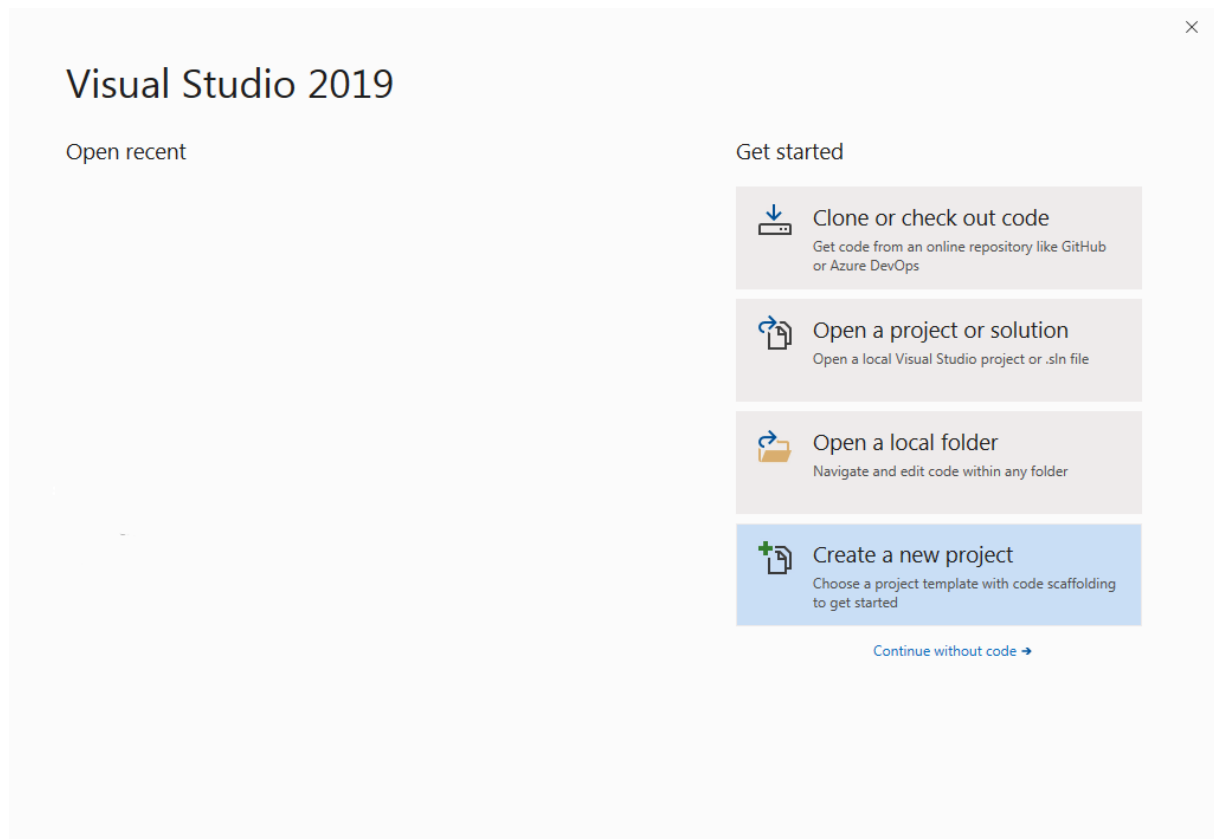
Célok:

- Virtuális metódusok használatának és készítésének gyakorlása.
- Véletlenszám generálás

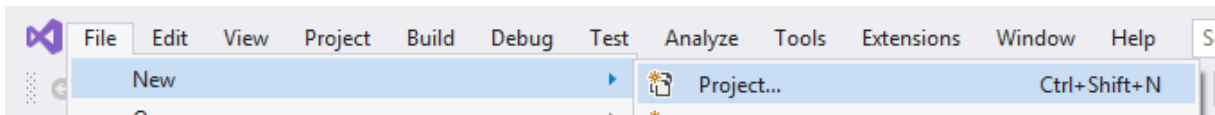
Készítsünk egy osztályt (Ős) az alábbi funkcionalitást megvalósító metódusokkal:

- Egész elemekből álló tömb létrehozása és feltöltése véletlenszámokkal.
- Adatok kiírása sztringbe úgy, hogy az elemek egymástól vesszővel elválasztva jelenjenek meg.
- Sztring kiírása a konzolra.
- Két egész szám összeadása (Művelet metódus)
- A művelet végrehajtása két tömb minden elemére (Számít metódus).
- Készítsünk egy leszármazott (Leszármazott) osztályt az előző osztályhoz, amelyben a Művelet metódus két szám különbségét számítja.

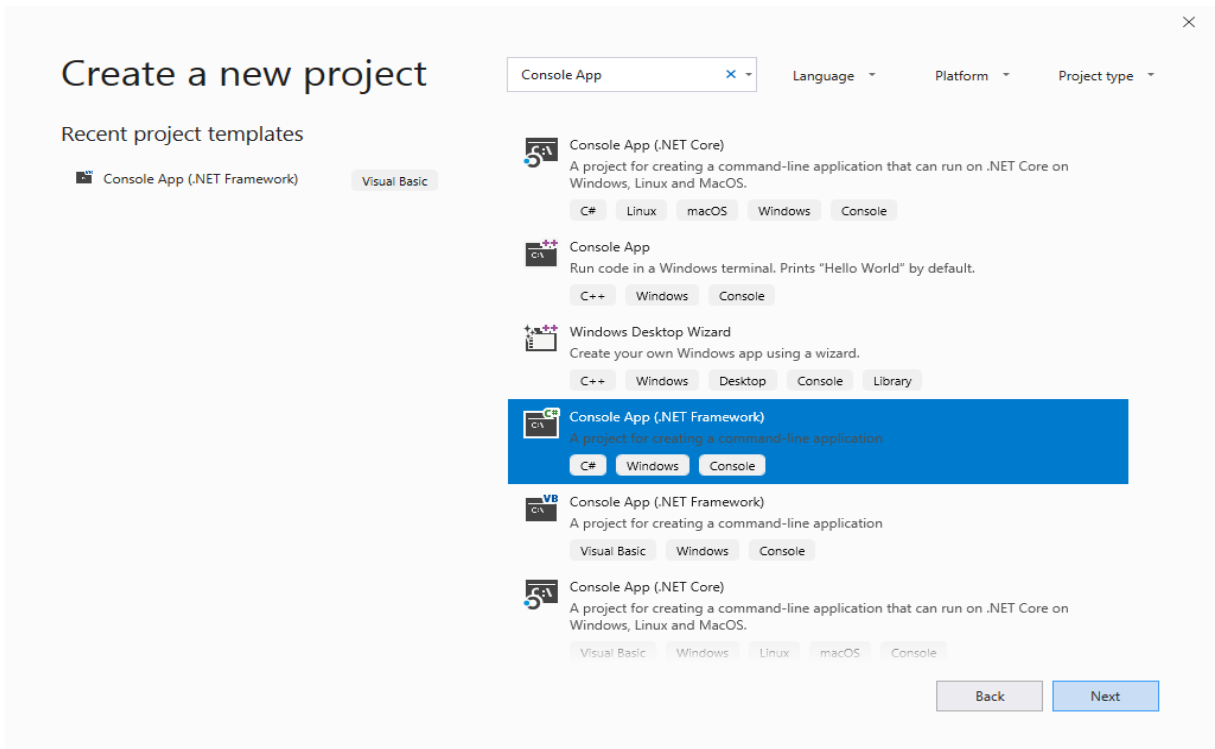
Egy konzol alkalmazással indítunk. Emlékeztetőül:



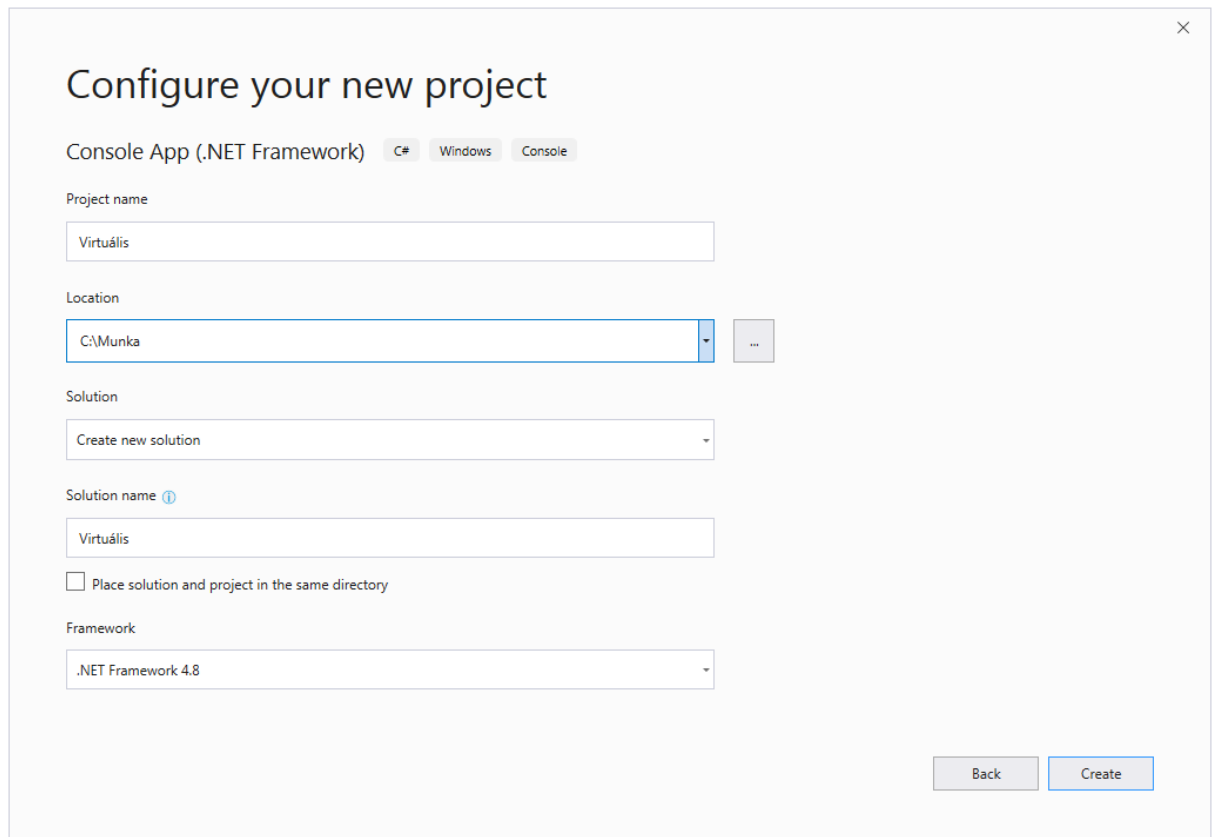
16. ábra



17. ábra.



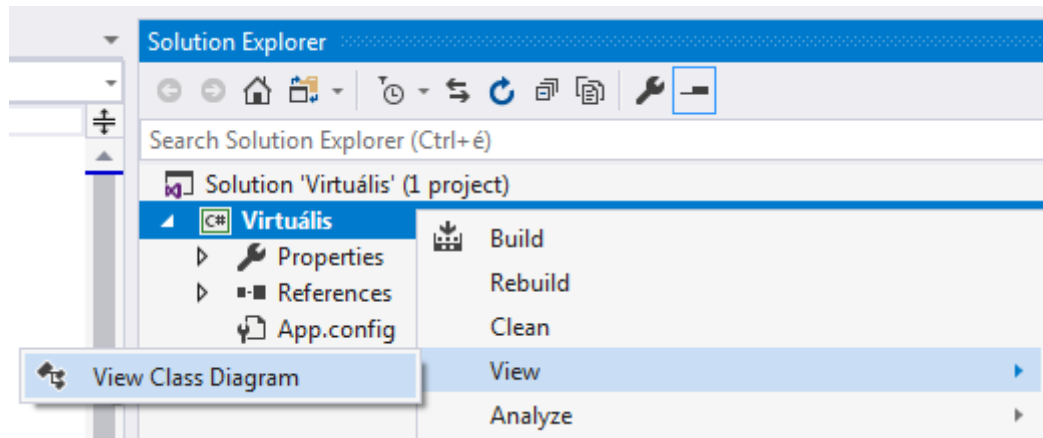
18. ábra



19. ábra.

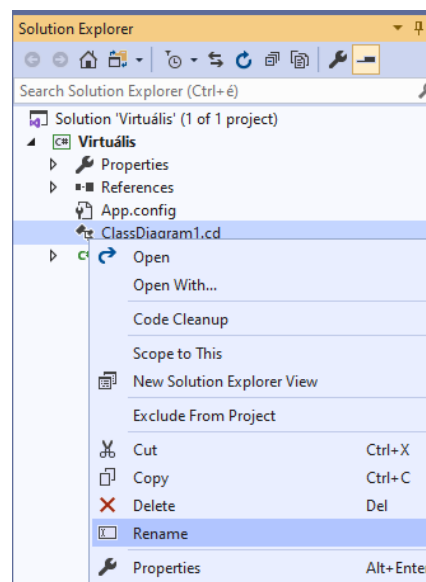


Létrehozzuk az osztálydiagramot, és elmentjük OsztD néven.



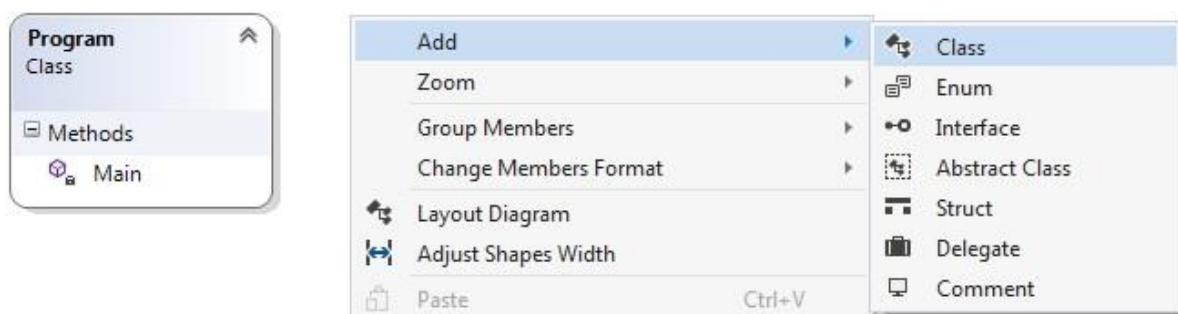
20. ábra.

Az átnevezéshez a Solution Explorerben kiválasztjuk az osztálydiagram fájlt, majd jobb egérgomb és a felbukkanó gyorsmenüben Rename menüpontot választjuk.

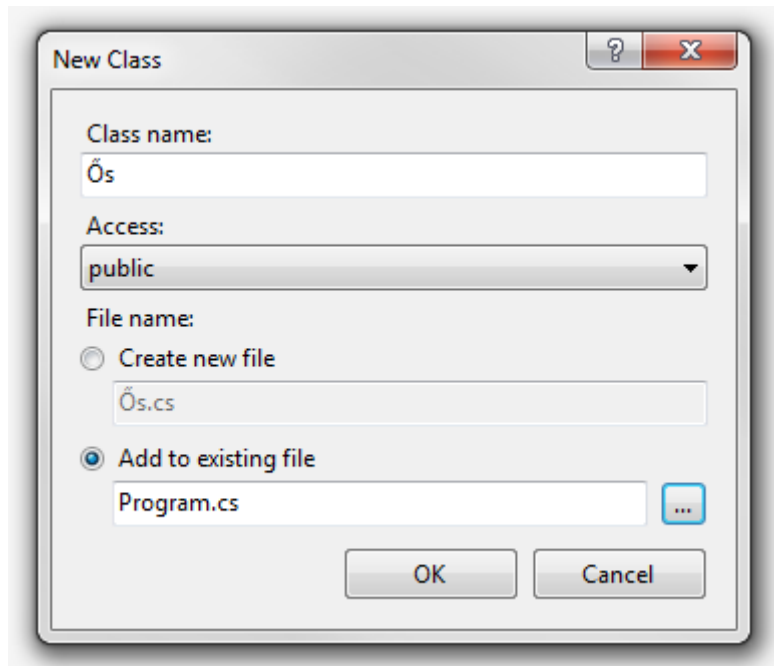


21.

Létrehozuk az Ős osztályt az aktuális állományban.

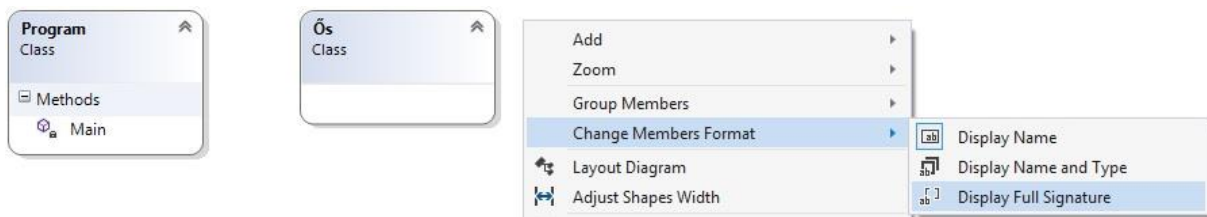


22. ábra.



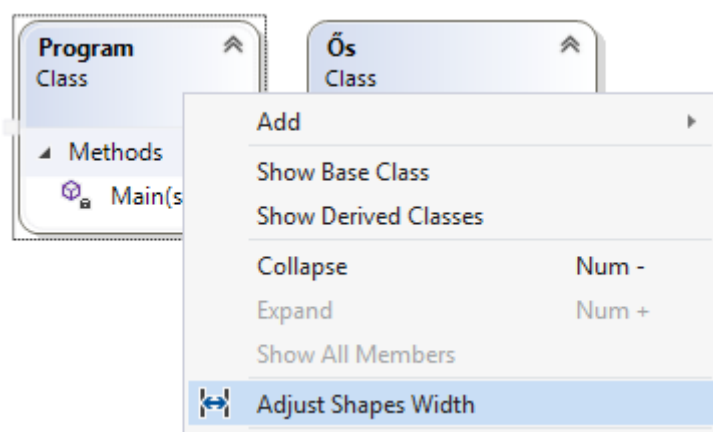
23. ábra.

Az osztálydiagramban váltsunk át a tagok részletes megjelenítésére.



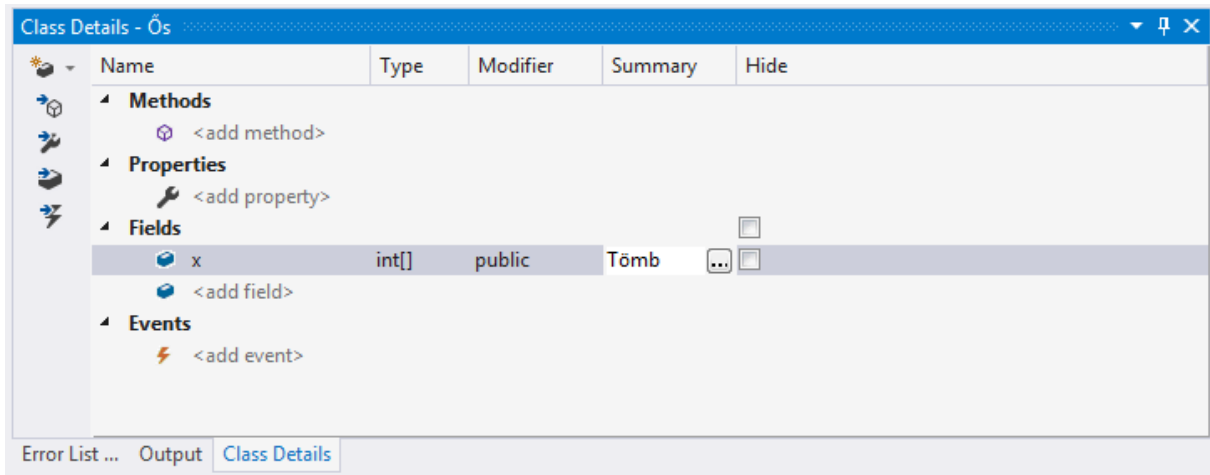
24. ábra.

Igazítsuk az osztályok szélességét a tartalomhoz.



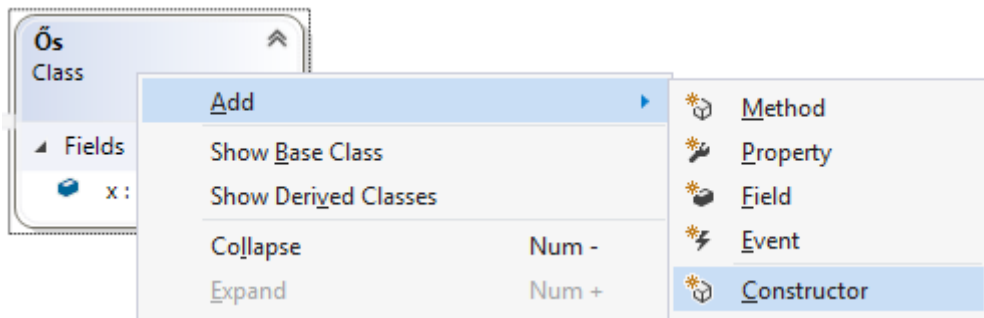
25. ábra.

Hozzuk létre az Ős osztályban az x nyilvános adattagot, ami int elemekből álló tömb referenciájának tárolására szolgál.



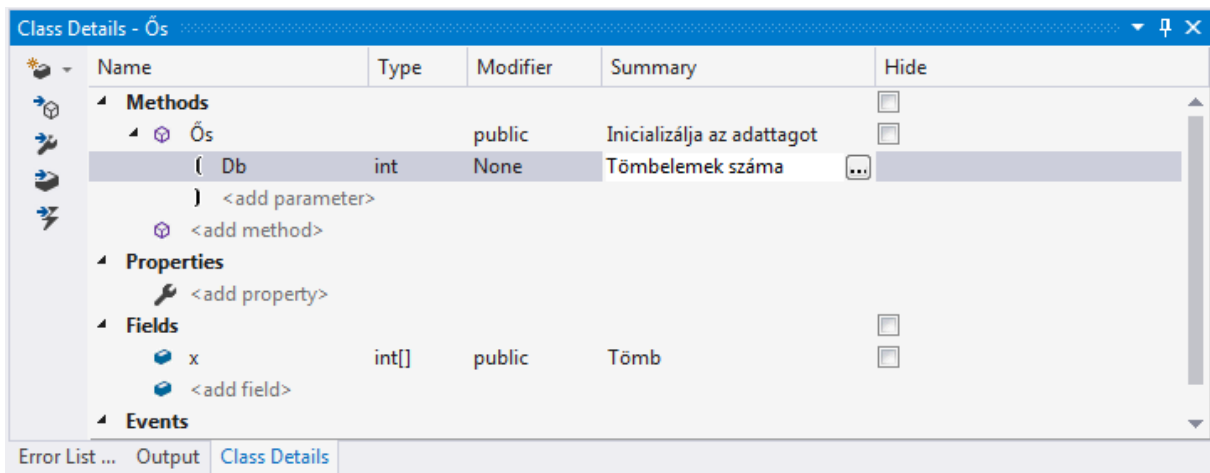
26. ábra.

Készítsünk egy konstruktort, ami paraméterként átveszi a tömb elemszámát, és létrehozza a tömböt.



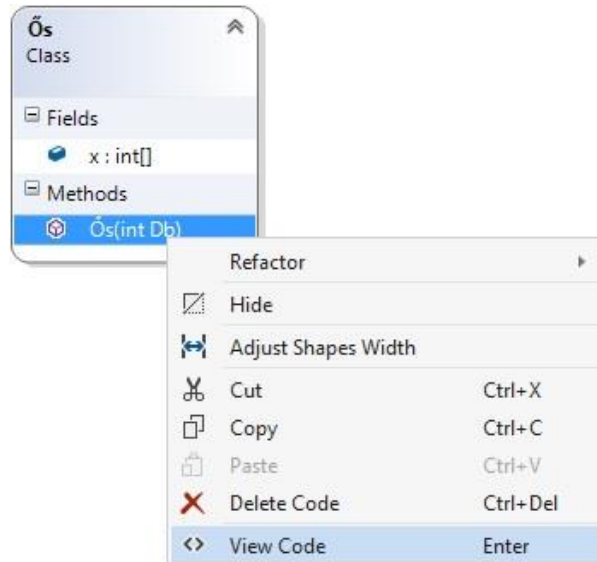
27. ábra.

A Class Details ablakban adjuk meg az argumentumot.



28. ábra.

Váltunk át kódnézetbe, és írjuk meg a függvénytörzset.



29. ábra.

```

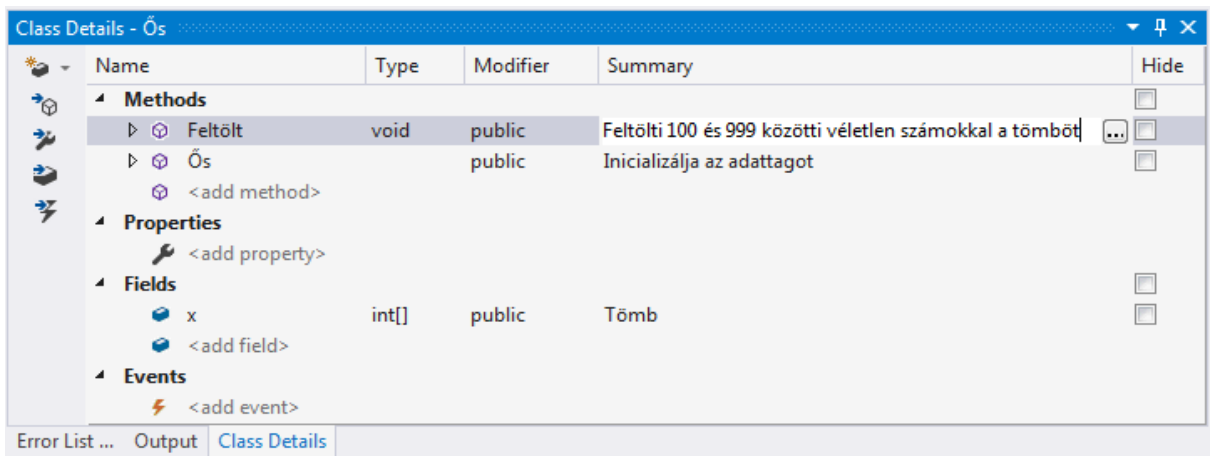
/// <summary>
/// Tömb
/// </summary>
public int[] x;
/// <summary>
/// Inicializálja az adattagot
/// </summary>
/// <param name="Db">Tömbelemek száma</param>
public Ős(int Db)
{
    throw new System.NotImplementedException();
}
    
```

Módosítsuk a kódot:

```

public Ős(int Db)
{ //Tömb létrehozása
    x = new int[Db];
}
    
```

Hozunk létre egy Feltölt nevű metódust, ami feltölti 100 és 999 közötti véletlen számokkal a tömböt.



30. ábra.

Váltsunk át kódnézetbe, és írjuk meg a függvénytorzset. Generált kód:

```
/// <summary>
/// Feltölti 100 és 999 közötti véletlen számokkal a tömböt
/// </summary>
public void Feltölt()
{
    throw new System.NotImplementedException();
}
```

Átírt kód:

```
public void Feltölt()
{ //Véletlen számokat előállító objektum létrehozása
    Random n = new Random(GetHashCode());
    //A tömb feltöltése adatokkal
    for(int i = 0; i < x.Length; i++)
    {
        x[i] = n.Next(100, 999);
    }
}
```

A ToString metódust úgy definiáljuk át, hogy egy olyan string-et hozzon létre, ami a tömbben tárolt adatokat tartalmazza, egymástól vesszővel elválasztva. A függvénytorzs újra írásához váltsunk át kódnézetre.

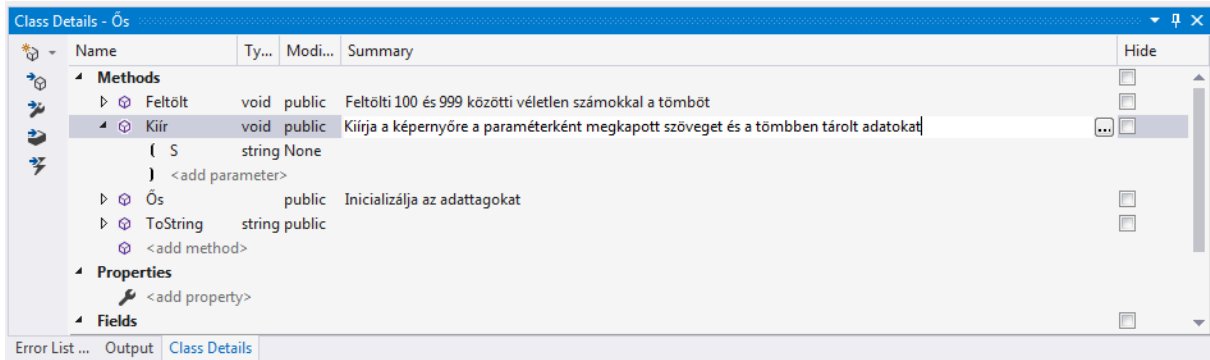
Alap függvénytorzs:

```
/// <summary>
/// Létrehoz egy sztringet, ami a tömbben tárolt adatokat tartalmazza egymástól
vesszővel elválasztva.
/// </summary>
/// <returns>A tömb adatai.</returns>
public override string ToString()
{
    throw new System.NotImplementedException();
}
```

Átírt függvénytorzs:

```
public override string ToString()
{
    string S = "";
    string Z;
    // Felfűzzük egy sztringbe az adatokat egymástól
    // vesszővel elválasztva
    // az utolsó adat kivételével, mert utána nem kell vessző
    // álljon.
    for(int i = 0; i < x.Length-1; i++)
    {
        Z = string.Format("{0,4:D}, ", x[i]);
        S += Z;
    }
    // Ha a tömb nem volt üres, akkor a sztring végéhez
    //hozzátesszük az utolsó adatot.
    if(x.Length - 1 >= 0)
    {
        Z = string.Format("{0,4:D}", x[x.Length - 1]);
        S += Z;
    }
    return S;
}
```

Definiáljunk egy metódust, ami kiírja a képernyőre a paraméterként megkapott szöveget és a tömbben tárolt adatokat.



31. ábra.

Váltunk át kódnézetbe, és írjuk meg a függvénytorzset.

```

/// <summary>
/// Kiírja a képernyőre a paraméterként megkapott szöveget és a
/// tömbben tárolt adatokat
/// </summary>
public void Kiír(string S)
{
    throw new System.NotImplementedException();
}

```

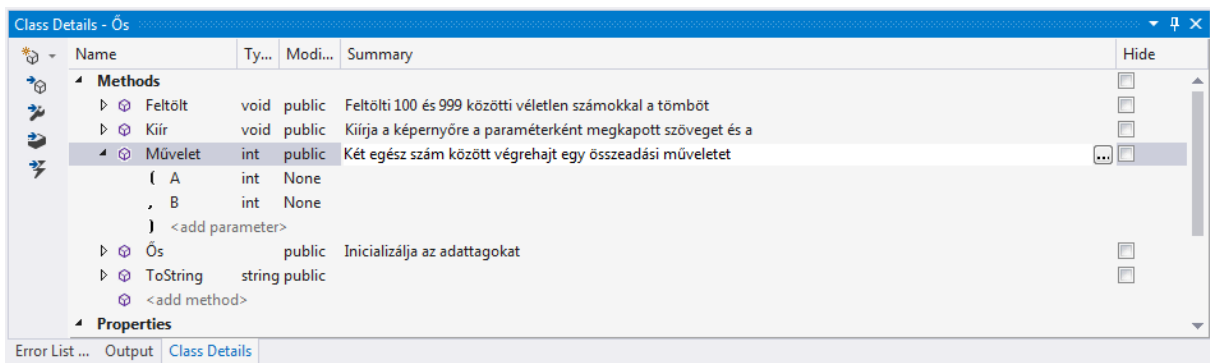
Átírt függvény törzs:

```

public void Kiír(string S)
{
    Console.WriteLine(S + this);
}

```

Definiáljunk egy Művelet nevű metódust, ami paraméterként átvesz két egész értéket és összeadja ezeket. Az eredmény lesz a visszatérési értéke.



32. ábra.

Váltunk át kódnézetbe, és írjuk meg a függvénytorzset.

```

/// <summary>
/// Két egész szám között végrehajt egy összeadási műveletet
/// </summary>
public int Művelet(int A, int B)
{
    throw new System.NotImplementedException();
}

```

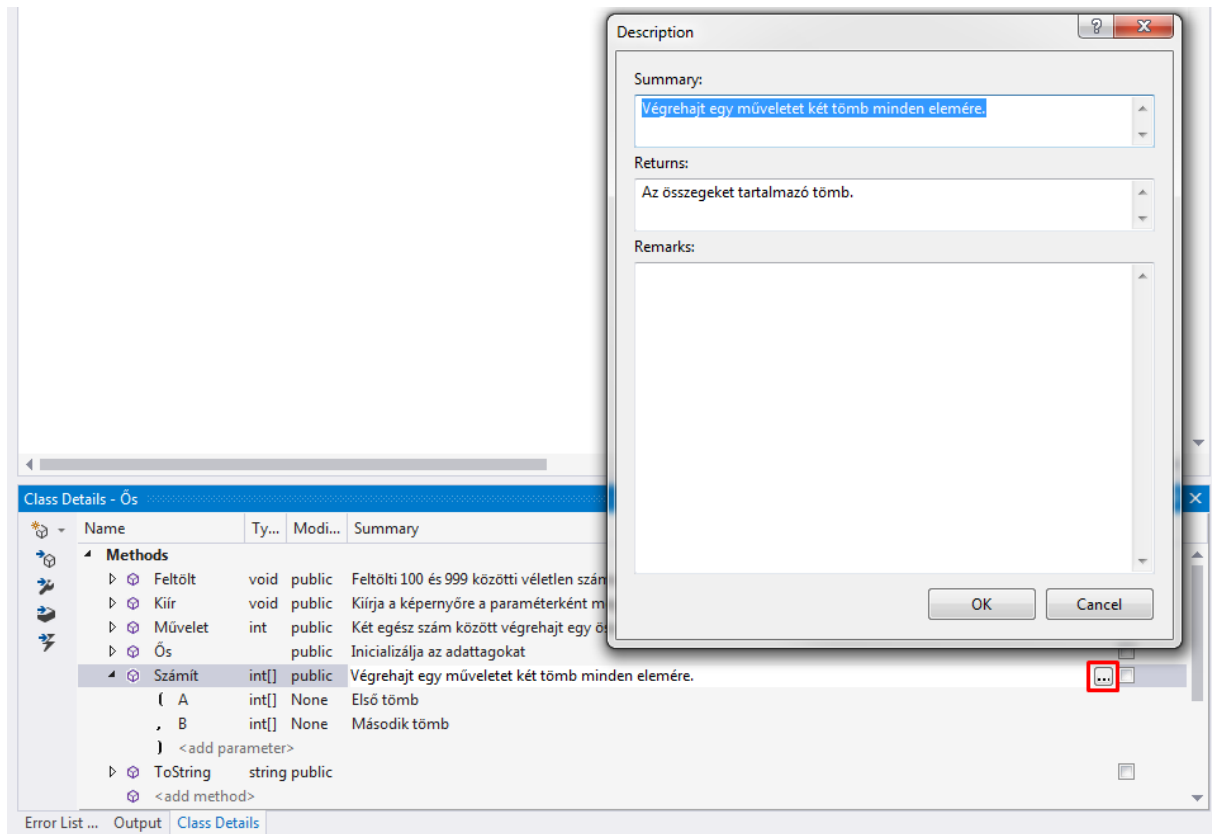
```
}

```

Jelöljük meg virtuálisként a metódust (virtual kulcsszó használata).

```
public virtual int Művelet(int A, int B)
{
    return A + B;
}
```

Készítsünk egy Számít nevű metódust, ami átvesz két, int elemekből álló tömbre vonatkozó referenciát, majd végrehajtja a műveletet a két tömbre, és visszaad egy ugyanilyen tömbre irányuló referenciát.



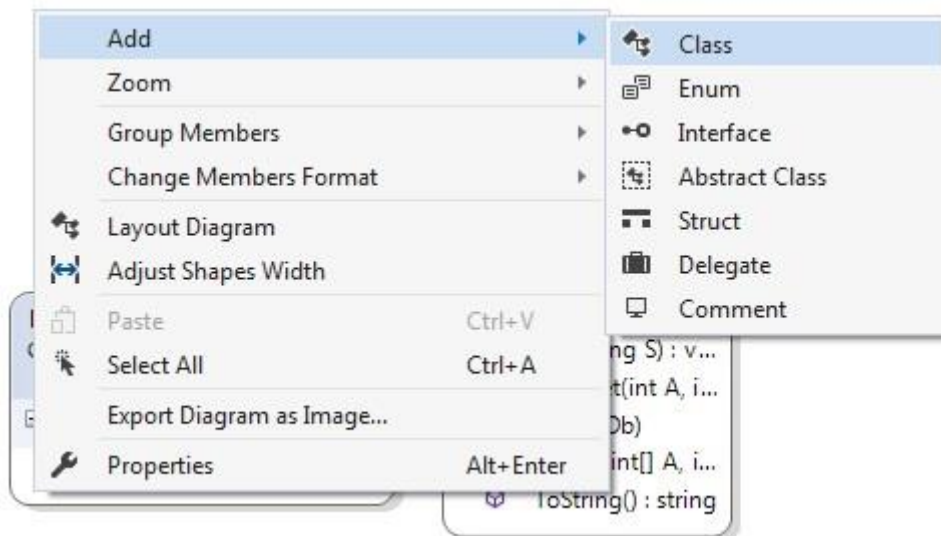
33. ábra.

```
/// <summary>
/// Végrehajt egy műveletet két tömb minden elemére.
/// </summary>
/// <param name="A">Első tömb</param>
/// <param name="B">Második tömb</param>
/// <returns>Az összegeket tartalmazó tömb.</returns>
public int[] Számít(int[] A, int[] B)
{
    if(A.Length != B.Length)
    {
        throw new Exception("A két tömb mérete eltérő!");
    }
    int[] C = new int[A.Length];
    for(int i = 0; i < C.Length; i++)
    {
        C[i] = Művelet(A[i], B[i]);
    }
    return C;
}
```

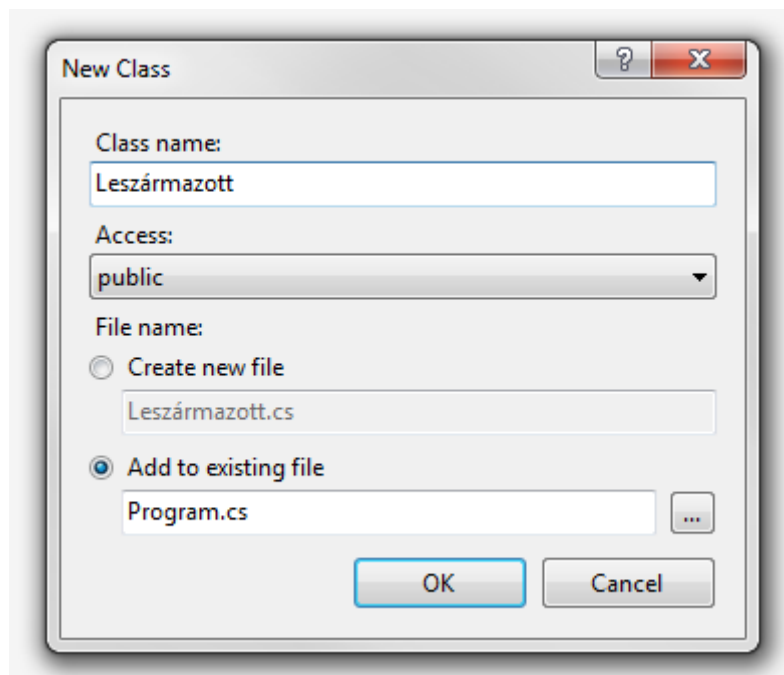
}

Készítsünk egy leszármazottat az Ős osztályhoz Leszármazott néven.

- Létrehozzuk az új osztályt a már ismert módon az aktuális állományban.

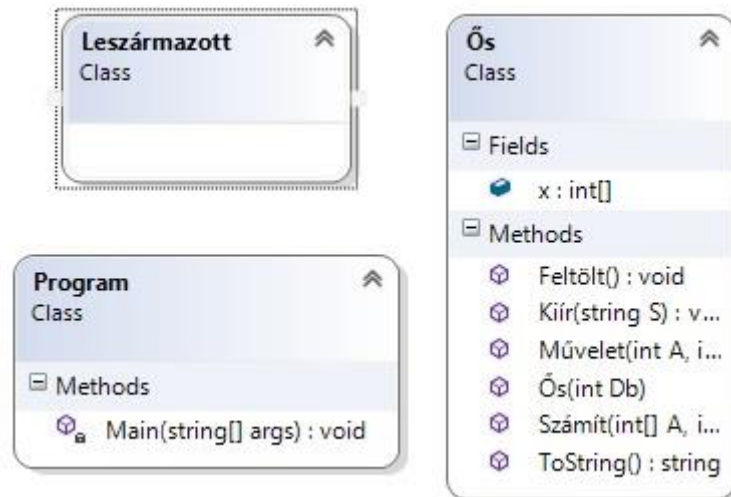


34. ábra.



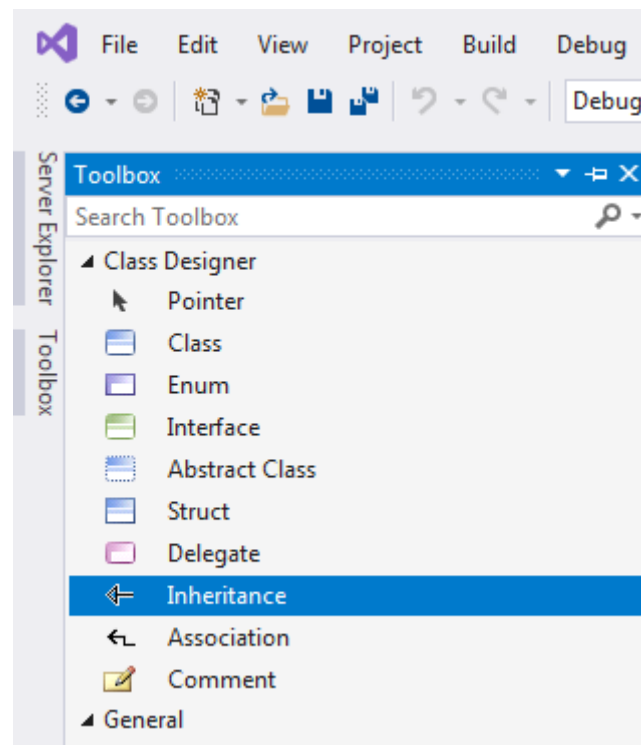
35. ábra.





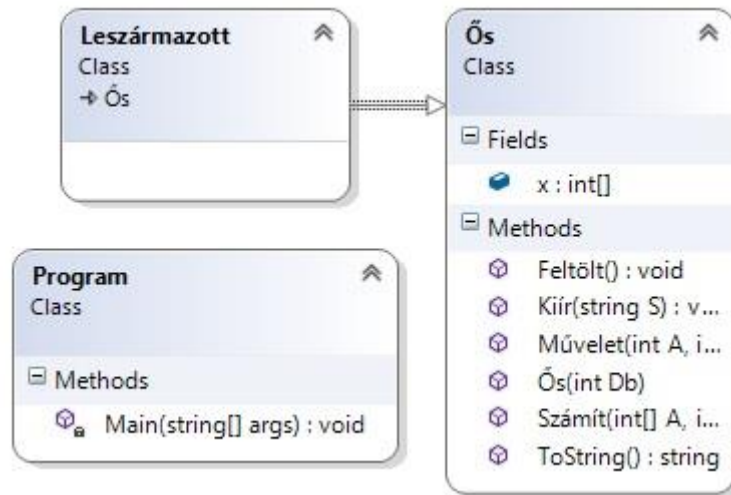
36. ábra.

- Láthatóvá tesszük a Toolbox eszköztárat a bal oldalon, és megnyitjuk a Class Designer csoportot.



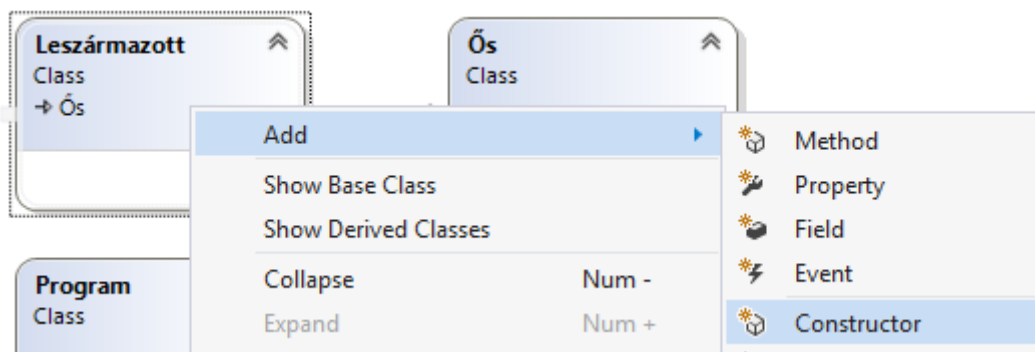
37. ábra.

- Kiválasztjuk az Inheritance-t (öröklődés) az eszköztáron.
- A Leszármazotton lenyomjuk a bal egérgombot, majd lenyomva tartva az egérgombot az Ősre húzzuk.

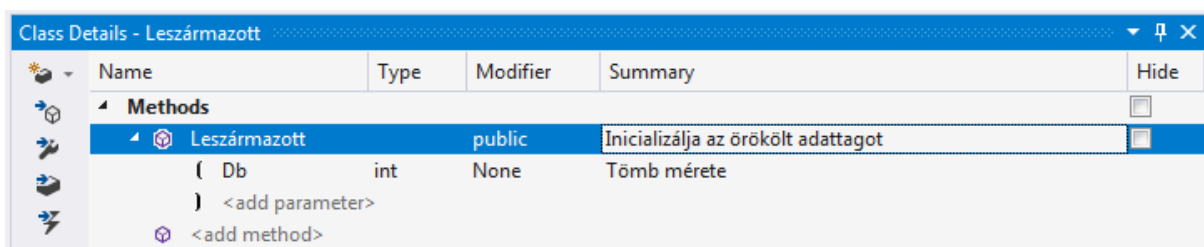


38. ábra.

A Leszármazott osztályban létrehozunk egy konstruktort, ami egy egész számot vesz át (darabszám) és *vele* az Ős osztály konstruktorának meghívása útján inicializálja az örökölt adatot.



39. ábra.



39. ábra.

Térjünk át kódnézetbe, és írjuk meg a metódus törzsét.

```
/// <summary>
/// Inicializálja az örökölt adattagot
/// </summary>
/// <param name="Db">Tömb mérete</param>
public Leszármazott(int Db) : base(Db) { }
```

Készítsük el az örökölt Művelet metódus átdefiniált változatát, melyben kivonást hajtunk végre. Térjünk át kódnézetbe, és írjuk meg a metódus törzsét.

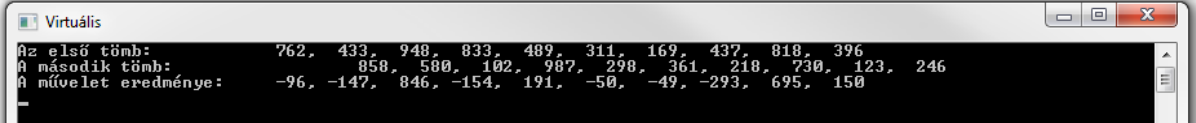
Az override kulcsszó után tegyük megjegyzésbe egy new kulcsszót.

```
/// <summary>
/// Két egész szám között végrehajt egy kivonási műveletet.
/// </summary>
/// <param name="A">Bal oldali operandus</param>
/// <param name="B">Jobb oldali operandus</param>
/// <returns>A két szám különbsége</returns>
public override /*new*/ int Művelet(int A, int B)
{
    return A - B;
}
```

Írjuk meg a Program osztály Main metódusának (főfüggvény) törzsét.

```
/// <summary>
/// Főfüggvény. Virtuális függvények meghívására példa.
/// </summary>
static void Main(string[] args)
{ //Létrehozunk két leszármazott típusú objektumot
    Leszármazott L1 = new Leszármazott(10);
    Leszármazott L2 = new Leszármazott(10);
    //Feltöltjük őket véletlen számokkal
    L1.Feltölt();
    L2.Feltölt();
    //Kiírjuk őket a konzolra
    L1.Kiír("Az első tömb:      ");
    L2.Kiír("A második tömb:      ");
    //Kiszámítjuk közöttük a műveletet
    Leszármazott L3 = new Leszármazott(10);
    L3.x = L3.Számít(L1.x, L2.x);
    //Kiíratjuk az eredményt
    L3.Kiír("A művelet eredménye:  ");
    Console.ReadLine();
}
```

Próbáljuk ki a programot.

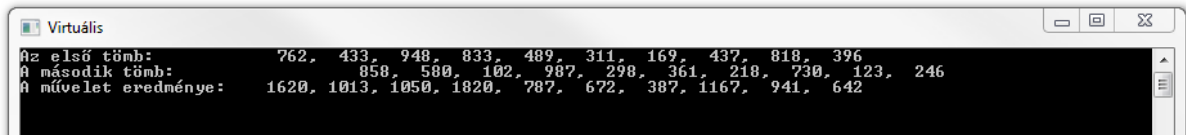


```
Virtuális
Az első tömb:      762, 433, 948, 833, 489, 311, 169, 437, 818, 396
A második tömb:   858, 580, 102, 987, 298, 361, 218, 730, 123, 246
A művelet eredménye: -96, -147, 846, -154, 191, -50, -49, -293, 695, 150
```

40. ábra.

Távolítsuk el a megjegyzéseket a new elől és mögül, valamint tegyük megjegyzésbe az override-ot. Fordítsuk és futtassuk le újból az alkalmazást.

```
/// <summary>
/// Két egész szám között végrehajt egy kivonási műveletet.
/// </summary>
/// <param name="A">Bal oldali operandus</param>
/// <param name="B">Jobb oldali operandus</param>
/// <returns>A két szám különbsége</returns>
public /*override*/ new int Művelet(int A, int B)
{
    return A - B;
}
```



41. ábra.

Mi okozza az eredmények közötti különbséget?

Az eredeti változatban a két Művelet metódus ugyanannak a virtuális láncnak voltak az elemei, így amikor a Leszármazott osztály objektumára meghívtuk az Ősben definiált Számít metódust, a Leszármazott osztály Művelet metódusa, azaz a kivonás hajtódott végre.

A második esetben a new kulcsszó alkalmazásával megszakítottuk a virtuális láncot, ezért az ősben definiált Művelet metódus, azaz az összeadás hajtódott végre.

### 1.3. Kivételkezelés

A C# nyelvben és általában a .NET keretrendszerben a hibajelzés és -kezelés széles körben alkalmazott formája a kivételek előírdézése és feldolgozása. Az alábbiakban megismerkedünk a nem kezelt kivétel fogalmával, a kivételek feldolgozási és továbbítási lehetőségeivel, valamint előírdézésükkel.

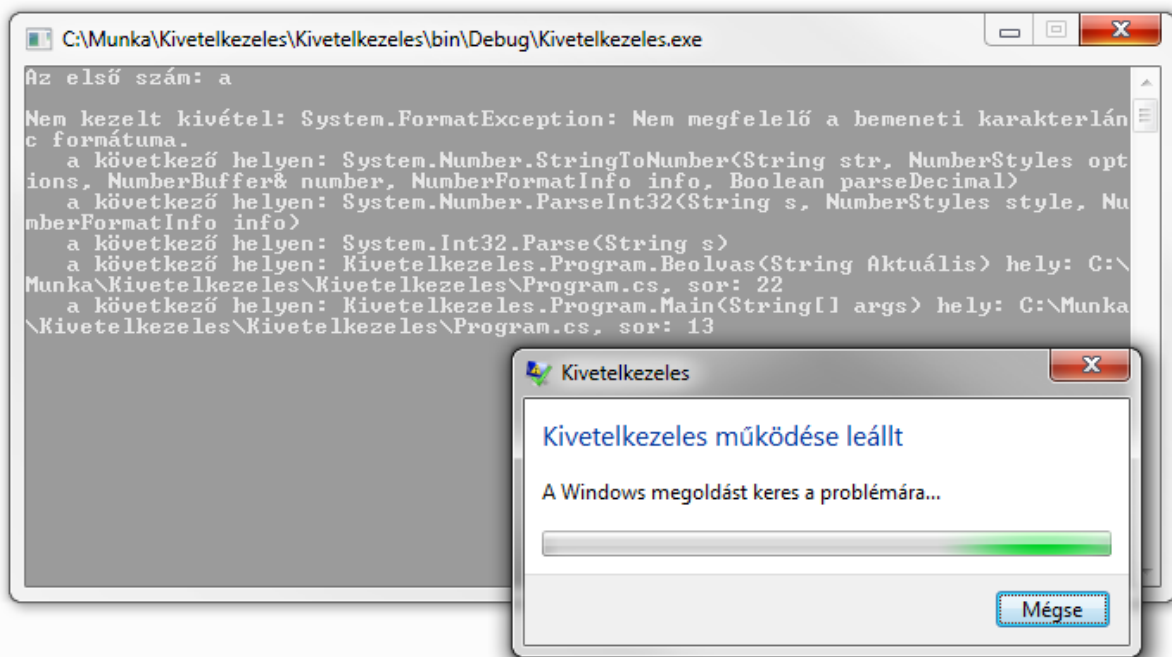
Bevezetésképpen tekintsünk egy kis programot, amelyben bekérünk két egész számot a konzolról, majd kiszámítjuk összegüket és megjelenítjük az eredményt a konzolablakban.

#### 1.3.1. Nem kezelt kivétel

A feladat első megoldását az alábbi kódrészlet tartalmazza. Az egyszerűség kedvéért az adatok beolvasását megvalósító metódust (`Beolvas`) statikus osztálytagnak választottuk.

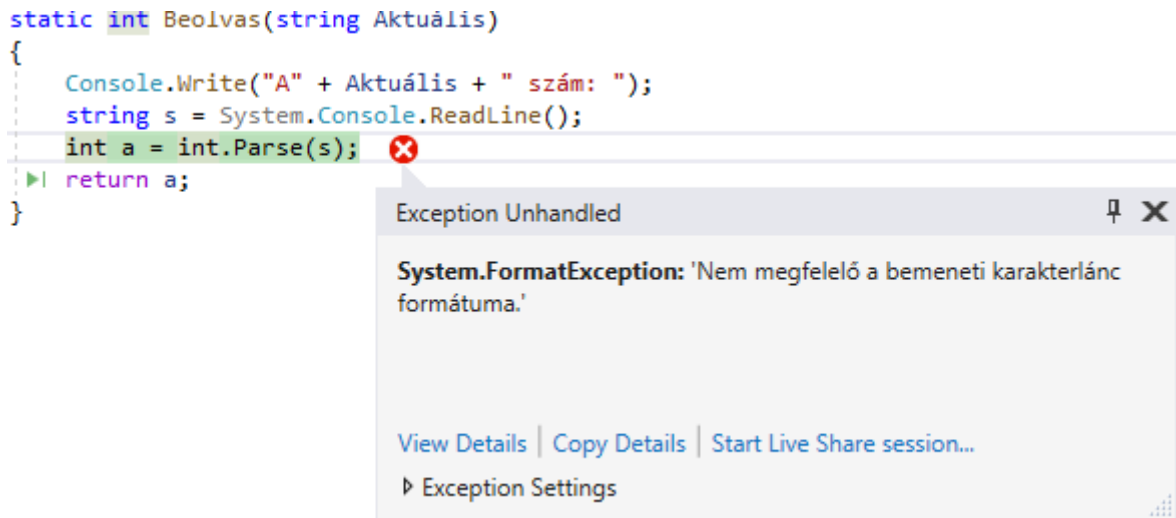
```
class Program
{
    static void Main(string[] args)
    {
        int a = Beolvas("z első");
        int b = Beolvas(" második");
        int c = a + b;
        Console.WriteLine("A két szám összege: {0}", c);
    }
    static int Beolvas(string Aktuális)
    {
        Console.Write("A"+Aktuális+" szám: ");
        string s = System.Console.ReadLine();
        int a = int.Parse(s);
        return a;
    }
}
```

Hibaüzenet a konzolon nem kezelt kivétel esetén:



42. ábra.

Kicsit eltérő eredményt kapunk akkor, ha debug módban indítjuk (F5) alkalmazásunkat. A hibás adatok megadása után a kódszerkesztőben az érintett utasítást sárga háttérrel kiemelve jelenik meg az Exception Assistant nem kezelt kivételre figyelmeztető ablaka:



43. ábra.

Itt is kaphatunk részletes információt a hibával kapcsolatban a View Details ... felírra kattintva. Az egyes változók értékeit ellenőrizhetjük a fejlesztőrendszer Locals és Watch ablakaiban.

Mi is történt valójában? A fenti két esetben az `s` változóban karakterláncként tárolt adatot a `Parse` metódus megpróbálta egész számmá alakítani, és a sikertelenséget egy kivétel esemény előidézésével jelezte. Emellett egy objektumot is létrehozott, aminek típusa és a benne tárolt adatok a hiba jellegéről és részleteiről adnak felvilágosítást. Első próbálkozásunknál a hiba bekövetkezése egyben az alkalmazás leállítását is jelentette.

### 1.3.2. Kivételek kezelése

Programunkat úgy szeretnénk továbbfejleszteni, hogy képes legyen kezelni a hibás felhasználói adatbevitelt a szám(ok) újbóli bekérésével. Ehhez olyan kódra van szükség, ami az alkalmazás leállása előtt érzékeli a kivételt, és gondoskodik a megfelelő vezérlésátadásról. A C# nyelvben a `try-catch-finally` szerkezet segítségével oldhatjuk meg a feladatot. Ez egy `try` blokkot, egy vagy több `catch` blokkot és nulla vagy egy `finally` blokkot tartalmaz.

#### A try-catch szerkezet

A `try` blokkban helyezük el azokat az utasításokat, amelyek végrehajtása során számíthatunk kivétel keletkezésére. Példánkban a konverziós utasítás kerül ide. A `catch` blokk(ok)ba helyezük el azokat az utasításokat, amelyekkel a hibára kívánunk reagálni. Példánkban átveszünk egy `FormatException` típusú kivétel objektumot, aminek feladata a hibához kapcsolódó információk hordozása. Most csak a `Message` tulajdonságot használjuk fel, ami egy rövid leírást tartalmaz a hibáról. Az újbóli adatbekérést a `Beolvas` metódus rekurzív meghívásával oldjuk meg.

## Try-catch blokk

```

static int Beolvas(string Aktuális)
{
    Console.WriteLine("A" + Aktuális + " szám: ");
    string s = System.Console.ReadLine();
    int a = 0;
    try {
        a = int.Parse(s);
    }
    catch(FormatException e){
        Console.WriteLine("Hibásan adta meg a számot!");
        Console.WriteLine(e.Message);
        a = Beolvas(Aktuális);
    }
    return a;
}

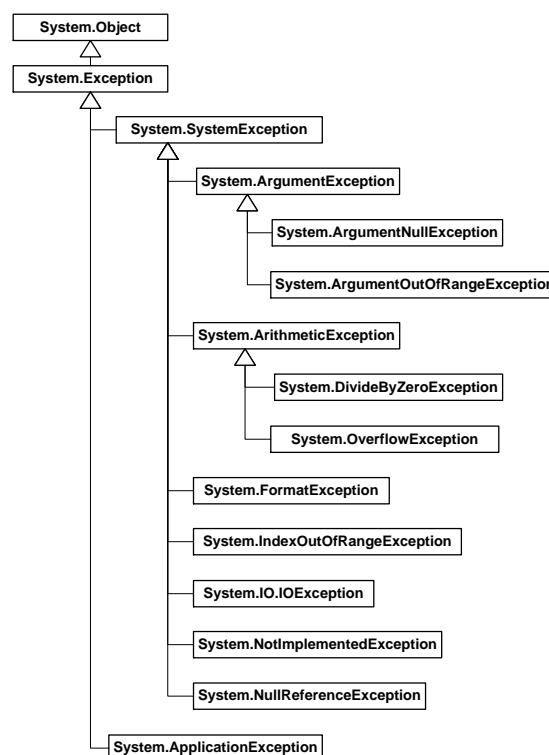
```

Lépésenként végrehajtva az alkalmazást, és a már megszokott „asd” betűsört megadva végigkövethetjük, hogy a `Parse` meghívása után a kivétel hatására a vezérlés a `catch` blokk fejlécére ugrik, majd sorban végrehajthatódnak az ott szereplő utasítások. Érvényes értéket megadva a `Beolvas` metódus újbóli meghívásakor a `Parse` sikeres konverziót hajt végre, a vezérlés átugorja a `catch` blokkot, és a metódus visszaadja `return`-el az a változó értékét.

## Egy vagy több catch blokk

Sok metódus esetében a futás során többféle hiba is előfordulhat, amelyekre néha teljesen eltérően kell reagálni. A hibatípusok elkülönítését jól szolgálják a kivétel osztályok. Ezeket témakörök szerint rendszerezték egy külön ágat kialakítva az osztályhierarchia `System` névterében. Az ág csúcsán az `Exception` osztály áll, ami közvetlen leszármazottja az `Object` osztálynak.

Az alábbi ábrán a teljesség igénye nélkül néhány gyakran alkalmazott kivételosztályt láthatunk jelezve hierarchiabeli elhelyezkedésüket is.



44. ábra.

Az alábbi táblázat röviden ismerteti jellegzetes alkalmazási területüket:

System.Exception	az alkalmazás végrehajtása során előforduló hibákhoz társított kivételek ősosztálya
System.SystemException	a System névtér előre definiált kivételtípusainak ősosztálya
System.ArgumentException	egy metódus valamely aktuális paramétere érvénytelen
System.ArgumentNullException	null referencia nem megengedett átadása
System.ArgumentOutOfRangeException	az átadott paraméter az érvényes tartományon kívülre esik
System.ArithmeticException	aritmetikai műveletek és típuskonverzió során előálló kivételek ősosztálya
System.DivideByZeroException	nullával történő osztás
System.OverflowException	túlcsordulási hiba
System.FormatException	a paraméter formátuma nem megfelelő
System.IndexOutOfRangeException	tömb túlindexelése
System.IO.IOException	fájlkezeléssel kapcsolatos kivételek ősosztálya
System.NotImplementedException	a meghívott metódus nem rendelkezik implementációval; például a fejlesztőrendszer a Class Details ablakban vizuálisan létrehozott metódusok vázába egy ilyen kivétel előidézését helyezi el
System.NullReferenceException	egy változón keresztül hivatkozunk egy objektum egy tagjára, és közben a változó null értékű
System.ApplicationException	a felhasználó által definiált kivételtípusainak ősosztálya

Egy alkalmazás fejlesztése során tisztában kell lennünk azzal, hogy milyen kivételeket idézhetnek elő a futtatókörnyezet (Common Language Runtime - CLR) vagy más forrásból származó osztályok/komponensek metódusai. A CLR esetében a fejlesztőrendszer sűgójában részletes információt találunk minden metódusról, továbbá a kódszerkesztőben a metódus neve felé helyezve az egérmutatót a felbukkanó gyorstippben is információt kapunk a lehetséges kivételekről. Bár az utóbbi megoldás sokkal kényelmesebb, de alkalmazhatósága korlátozott olyankor, amikor több azonos nevű metódus is található az osztályban/struktúrában. Így például az `int` (`Int32`) struktúra `Parse` tagja esetén a gyorstipp a három paraméteres változatot jeleníti meg, amihez négy kivételtípus tartozik. Az általunk alkalmazott egyparaméteres típus azonban három fajta kivételt idézhet elő. Ezek az `ArgumentNullException`, `FormatException` és az `OverflowException`.



Példaprogramunk futása közben az első típus gyakorlatilag nem fordulhat elő, így a megkívánt biztonság elérése érdekében a másik kettőre kell felkészítenünk alkalmazásunkat. Egész számként nem értelmezhető karaktorsor esetét eddig is kezelni tudta metódusunk, a továbbiakban a túlcordulás, azaz az abszolút értékben túl nagy szám esetével kell foglalkoznunk. Ez kétféleképpen oldható meg. Vagy mindkét hibatípushoz külön `catch` blokkot rendelünk vagy egyetlen közös `catch` blokkot alkalmazunk.

Amennyiben az első utat választjuk, az alább ismertetett kódrészletet kell elhelyeznünk az eredeti `catch` blokkot követően a `return` elé.

```
catch(OverflowException e)
{
    Console.WriteLine("Hibásan adta meg a számot!");
    Console.WriteLine(e.Message);
    a = Beolvas(Aktuális);
}
```

Lépésenként végrehajtva a programot nyomon követhetjük, hogy a szokásos „asd”-t megadva az első, míg a 2147483648 értéket megadva a második `catch` blokk hajtódik végre. A program működik, azonban ez a megoldás csak olyankor előnyös, ha a különböző hibák eltérő reakciót igényelnek.

Példánkban azonban a második blokk az elsővel azonos utasításokat tartalmaz, így inkább a második megoldást, azaz a közös `catch` blokkot alkalmazzuk. Tudva, hogy egy ős osztályhoz létrehozott referencia változó képes tárolni bármely leszármazott osztályból példányosított objektum referenciáját, megkeressük az osztályhierarchiában a legközelebbi olyan osztályt, amely mindkét kivételtípusnak őse. Esetünkben a `SystemException` felel meg e követelménynek. Ezért ezt az osztályt adjuk meg a közös `catch` blokk fejlécében kivételtípusként.

```
catch(SystemException e)
{
    Console.WriteLine("Hibásan adta meg a számot!");
    Console.WriteLine(e.Message);
    a = Beolvas(Aktuális);
}
```

Általános érvénnyel elmondható, hogy egy `try` blokkhoz több `catch` blokk is kapcsolható. Ezek közül mindig csak egy hajtódik végre, éspedig az, amelyikre elsőként teljesül felülről lefele haladva az, hogy formális paraméterének típusa vagy azonos a kivétel objektum típusával vagy őse annak. Amennyiben olyan kivétel esemény következik be, amelyikre a fenti két feltétel egyike sem teljesül, akkor az első példához hasonlóan nem kezeltnek minősül a kivétel. Amennyiben a `catch` blokkban nincs ugró utasítás (pl. kivétel továbbadása, új kivétel előidézése, kilépés a programból, stb.), akkor az alkalmazás végrehajtása az utolsó `catch` blokkot követő utasítással folytatódik.

### Általános catch blokk

A fent bemutatott közös kivételkezelőnket elkészíthetjük paraméter nélküli változatban is az alábbi kódrészletnek megfelelően. Ez a megoldás a vezérlésátadás szempontjából egyenértékű azzal, mintha `Exception` típusú paramétert használnánk, tehát bármilyen kivétel esetén végrehajtódik. Az eltérés csak abban mutatkozik, hogy nem kapunk hozzáférést a kivétel objektumhoz, és nem rendelkezünk pontos információval az okra vonatkozólag.

```
catch
{
    Console.WriteLine("Hibásan adta meg a számot!");
    a = Beolvas(Aktuális);
}
```

### Kivétel továbbítása

A C# nyelv lehetőséget biztosít arra, hogy a kivételt ne csak annak keletkezési szintjén érzékeljük, hanem a hívási lánc magasabb szintjein elhelyezkedő metódusokban is. Ezt a kivétel továbbadásával érhetjük el a `throw` kulcsszó segítségével. Demonstrálásként szolgáljon az alább bemutatott metódus, ami átvesz egy `Graphics` típusú objektumot valamint két struktúrát, amelyek a szín és a befoglaló téglalapra vonatkozó információt hordozzák, majd rajzol egy kifestett téglalapot. A metódus Windows Forms típusú felülethez készült.

```
void Ellipszis(Graphics gr, Color cSzín, Rectangle rTéglalap)
{
    SolidBrush sbEcset = new SolidBrush(cSzín);
    try
    {
        gr.FillEllipse(sbEcset, rTéglalap);
    }
    catch (NullReferenceException e)
    {
        Console.WriteLine("Nincs hova rajzolni!" + e.Message);
        throw e;
    }
    finally
    {
        sbEcset.Dispose();
    }
}
```

A metódusban létrehozunk egy egyenletes színnel kifestő ecsetet, majd kísérletet teszünk a rajzolásra. Amennyiben az első paraméter null értékű, azaz nem rendelkezünk a festővászont megtestesítő `Graphics` típusú objektummal, akkor `NullReferenceException` típusú kivétel keletkezik, amit a szabványos kimenetre küldött hibaüzenettel jelzünk, majd a kivételt továbbadjuk az `Ellipszis` metódus hívójának. A metódus egy `finally` blokkot is tartalmaz, amelynek magyarázatát kicsit később ismertetjük.

Az `Ellipszis` metódus kipróbálása érdekében készítsünk egy Windows grafikus felületű alkalmazást a *Windows Forms App* (.Net Framework) sablon segítségével `Ellipszis` projektnévvel. Az ablak osztályába helyezzük el a fentiekben megismert `Ellipszis` metódust.

Az ablakon helyezzünk el egy nyomógombot (`btRajzol`), és készítsünk hozzá egy a kattintásra reagáló eseménykezelőt `btRajzol_Click` néven. Ebben határozzuk meg a befoglaló téglalapot, és hívjuk meg az `Ellipszis` metódust.

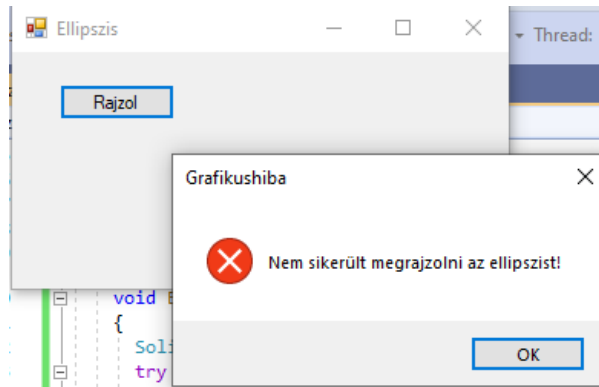
```
private void btRajzol_Click(object sender, EventArgs e)
{
    Rectangle rTéglalap = new Rectangle(0,0,200,100);
    try
    {
        Ellipszis(this.CreateGraphics(), Color.Red, rTéglalap);
    }
}
```

```

}
catch
{
    MessageBox.Show("Nem sikerült megrajzolni az ellipszist!",
        "Grafikushiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

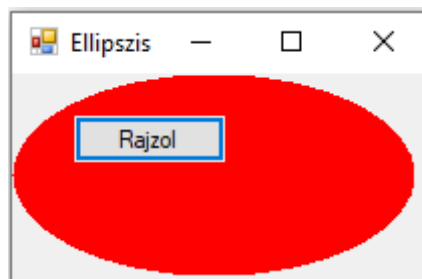
A kivételkezelés kipróbálása érdekében cseréljük le az Ellipszis metódus meghívásának első aktuális paraméterét, azaz a `this.CreateGraphics()` helyett írjunk `null`-t.



45. ábra.

Újból lefutattva a programot a nyomógomb eseménykezelőjének `catch` blokkjában előírt üzenetablak jelenik meg. A fejlesztőrendszer Output ablakában a „Show output from:” listában a Debug-ot kiválasztva láthatjuk, hogy a rendszer által generált hibaüzenetek között megjelent az általunk definiált „Nincs hova rajzolni!” sor is.

Helyes paraméterekkel lefutattva a programot, az alábbiakat kapjuk.



46. ábra.

### A finally blokk használata

Előfordulhat, hogy egy vagy több utasítást, például lefoglalt erőforrások felszabadítását, a kivétel bekövetkezésétől függetlenül mindenképpen végre kell hajtanunk a `try`-al vizsgált blokk után. Ez nem okoz különösebb gondot, ha egyik `catch` blokk sem tartalmaz olyan ugró utasítást, amelynek hatására az utolsó `catch` blokkot követő kódsort kihagyva valahol máshol folytatódna az alkalmazás futása. Amennyiben azonban fennáll az átugrás veszélye, akkor a végrehajtást csak úgy biztosíthatjuk, ha az említett utasításokat egy `finally` blokkban helyezük el.

A fentebb bemutatott Ellipszis metódusban létrehozunk egy `SolidBrush` típusú ecset objektumot. Használata után a `Dispose` metódus meghívásával fel kell szabadítanunk az általa lefoglalt erőforrásokat attól függetlenül, hogy a rajzolás sikeres volt-e vagy sem. A feladatot megoldó utasítást egy `finally` blokkba helyezük el. Lépésenként végrehajtva az

alkalmazást az `Ellipszis` metódus fentiekben bemutatott hibás paraméterezése mellett, láthatjuk, hogy a `throw` végrehajtásakor a metódusból történő kilépés előtt még végrehajtódik a `Dispose` is.

### Túlszordulási kivétel

Túlszordulásról beszélünk akkor, ha egy adatot a szükségesnél kisebb tárolóképességű változóba kívánunk elhelyezni. Konstans értékek esetén a fejlesztőrendszer ezt már fordítási időben jelzi, míg változóknál csak futási időben derül ki a probléma, amennyiben be van kapcsolva a túlszordulás figyelése.

Próbaképp futtassuk le újból a két szám bekérését és összeadását végző programunkat első számként az `int`-ben tárolható legnagyobb értéket, azaz 2147483647-et és második értéként 1-et megadva. Az eredmény -2147483648 lesz, ami egyértelműen utal a túlszordulás bekövetkeztére, kivétel azonban nem keletkezett.

A túlszordulás figyelését és bekövetkezésekor a kivétel előidézését kétféleképpen érhetjük el. A teljes megoldásra vonatkozóan bekapcsolja az ellenőrzést a fordító `/checked+` parancssori kapcsolója, amit a fejlesztőrendszerben úgy állíthatunk be, hogy kiválasztjuk a Project menü Projektnév Properties... menüpontját, majd az előbukkanó párbeszédpanelen a Build fület választjuk, ezt követően az Advanced gombon kattintunk, és végül bekapcsoljuk a Check for arithmetic overflow/underflow jelölőnégyzetet. A beállítás elvégzése után újból próbálkozva az alkalmazásunk `OverflowException` kivétellel leáll az összeadásnál.

A fordítóprogram kapcsolóitól függetlenül egy kifejezésre vagy egy utasításcsoportra is szabályozhatjuk a túlszordulás figyelését a `checked` és `unchecked` kulcsszavak alkalmazásával.

Túlszordulás figyelése egy kifejezés kiértékelése során:

```
int a = Beolvas("z első");
int b = Beolvas(" második");
int c = checked (a + b);
```

Utasításcsoportra kiterjedő túlszordulás-felügyelés:

```
int a,b,c,d;
checked{
    a = Beolvas("z első");
    b = Beolvas(" második");
    c = a + b;
    d = a * b;
}
```

### 1.3.3. Kivételek előidézése

Az eddigiekben az általunk meghívott metódusokban előidézett kivételek kezelésével és továbbadásával foglalkoztunk. Nézzük most meg, hogyan készíthetünk mi magunk is olyan kódrészletet, ami egy kivétel segítségével jelzi a hívó számára egy hiba bekövetkezését.

A feladat legyen egy átlagszámító metódus elkészítése, amely egy `double` típusú elemekből álló tömbre vonatkozó referenciát vesz át, és visszaadja az abban tárolt adatok átlagát. A lehetséges hibák jelzésére a következő kivétel osztályokat használjuk fel:

- `ArgumentNullException` a metódust `null` referenciával hívták meg, azaz nem létezik a tömb;

- `ArgumentException` a tömb elemeinek száma nulla vagy a tömb valamely eleme érvénytelen (nem szám vagy végtelen érték);
- `ArithmeticException` a számok összege meghaladja a `double` típusban maximálisan tárolható értéket (`double.MaxValue`).

A kivételek előírdézése a `throw` kulcsszó segítségével történik, amit egy kivétel objektum létrehozása követ. Az alábbi metódusban a kivételosztály konstruktorának paraméterként átadunk egy rövid hibaüzenetet.

```
static double ÁtlagSzámít(double[] Tömb)
{
    if (Tömb == null)
        throw new ArgumentNullException("A tömb nem létezik!");
    int Méret=Tömb.Length;
    if (Méret == 0)
        throw new ArgumentException("A tömb nem tartalmaz elemeket!");
    double Átlag = 0;
    for (int i = 0; i < Méret; i++)
    {
        if (double.IsNaN(Tömb[i]) || double.IsInfinity(Tömb[i]))
            throw new ArgumentException("A tömb " + i.ToString() + ". Eleme "+
                "érvénytelen értéket tartalmaz!");
        Átlag = Átlag + Tömb[i];
        if (double.IsInfinity(Átlag))
            throw new ArithmeticException("A tömb elemeinek összege" +
                " túl nagy érték!");
    }
    Átlag /= Méret;
    return Átlag;
}
```

#### 1.3.4. Jótanácsok

Az alábbiakban összefoglalunk néhány ajánlást a kivételek alkalmazásával és kezelésével kapcsolatosan.

- A kivételek előírdézése és kezelése jelentős erőforrás igényel jár és lassítja az alkalmazás végrehajtását az újabb osztályok betöltése és a veremkezelési feladatok következtében. Lehetőleg csökkentsük minimális mértékűre az általunk előírdézett kivételek számát.
- Csak olyan kivételeket dolgozzunk fel, amelyek esetén ismert az őket előírdező hiba kezelési módja. A többi kivétel feldolgozását engedjük át magasabb hívási szinten elhelyezett kivételkezelőknek.
- Gondoskodjunk mindig azon nem használt erőforrások felszabadításáról, amelyek nem tartoznak az automatikus szemétyűjtő mechanizmus hatálya alá.
- Kerüljük újabb kivétel előírdézését a `finally` blokkban, ugyanis ha egy kivételt nem fogunk el egyetlen `catch` blokkal sem, akkor végrehajtnak a `finally` blokkban elhelyezett utasítások, és az újabb kivétel előírdézése következtében az eredeti elvész.
- Amennyiben egy programszakaszon belül több egymás utáni utasításnál is elképzelhető kivétel keletkezése, úgy ne készítsünk ezekhez külön `try-catch-finally` szerkezeteket, mert nehezen olvashatóvá tennék a kódot. Helyezzük el inkább az érintett utasításokat egyetlen `try` blokkban, és készítsünk minden kivétel típushoz `catch` blokkot.

- Több catch blokkot tartalmazó kivételkezelésnél az egyes blokkokat az osztályhierarchia figyelembe vételével úgy kell sorba rendezni, hogy fentről lefele haladva a specifikusabb osztályok megelőzzék az általánosabbakat.

### 1.3.5. Ellenőrző kérdések

Mikor minősül nem kezeltnek egy kivétel?

Ellenőrizni tudjuk-e a fejlesztőrendszer segítségével, hogy egy kivétel bekövetkezésekor az érvényességi körükön belül levő változók milyen értékkel rendelkeznek?

Hogyan tudjuk kideríteni, hogy a .NET osztályhierarchia egy metódusa milyen kivételeket idézhet elő?

A kivétel objektum mely tagjából olvashatjuk ki a hiba rövid szöveges leírását?

Mikor célszerű közös catch blokkot készíteni több kivétel típushoz?

Ha több catch blokkot készítünk egy kivételkezeléshez, akkor bármilyen sorrendben elhelyezhetjük ezeket?

Mire szolgál a new kulcsszó a kivételt előidéző utasításban?

Mindig keletkezik kivétel túlcsondulási hiba esetén?

Megoldható-e, hogy egy kivétel feldolgozása ne abban a metódusban történjék, ahol azt elfogtuk?

Kötelező-e a finally blokk használata?

## 1.4. Eseménykezelés – Lottójáték

Célok:

Események, eseménykezelés háttérének, fogalmainak rövid áttekintése.

Eseménykezelést megvalósító nem grafikus felületű példaprogram készítése.

Objektum-, statikus- és nem statikus metódusok feliratkoztatása eseményre.

### 1.4.1. Események, közzetevő-feliratkozó modell

Az asztali alkalmazások (DesktopApplications) fejlesztésével kapcsolatos irodalomban gyakran találkozhatunk az események és az eseményvezérelt alkalmazások/programozás fogalmával. Mindenkinek van valamilyen elképzelése az esemény szó jelentéséről a mindennapi életben, de vajon mit jelent ez a szoftverfejlesztés világában? Itt esemény lehet például a kattintás egy nyomógombon, menüpontra vagy valami más vezérlő elemre, de ugyanígy eseményt idéz elő az, ha változtatunk valamit az állományrendszerben, vagy a felhasználó megpróbál bezárni egy futó alkalmazást. Emellett az események nemcsak külső behatásokhoz kapcsolódhatnak, például beállíthatunk egy időzítőt (Timer), ami meghatározott időközönként jelez, azaz Tick eseményt idéz elő, de akár mi magunk is írhatunk eseményeket előállító kódot.

Szoftverfejlesztési szempontból az eseményt mindig egy objektum idézi elő (pl. a kattintásos esetben a nyomógomb, menüpont, vagy más vezérlő; az állományrendszer esetében egy speciális FileSystemWatcher típusú objektum; az időzítésnél egyTimer objektum). Az eseményt előidéző objektumot szerepköre alapján *közzétevőnek* (Publisher) nevezzük.

Az események nem öncélúan keletkeznek, mindig kapcsolódik hozzájuk valamilyen feladat, amit végre kell hajtani a bekövetkezést követően. Ezen feladat végrehajtását eseménykezelésnek nevezzük, és a szükséges utasításokat egy metódus, az ún. eseménykezelő segítségével hajtjuk végre. Az eseménykezelő megírása mellett arról is gondoskodnunk kell, hogy megteremtsük a kapcsolatot közte és az eseményt előidéző metódus között, azaz biztosítsuk azt, hogy az esemény bekövetkezésekor végrehajtsódjon a metódus. A kapcsolat létrehozása a gyakorlatban azt jelenti, hogy a közzétevő objektumban beregisztráljuk metódusunkat. A regisztrációs lépést feliratkozásnak, míg a metódus szerepkörét *feliratkozónak* nevezzük.

A fentiekben megismert két szerepkörből származik az eseménykezelési modell elnevezése, a *közzétevő-feliratkozó modell* is. A későbbiekben látni fogjuk, hogy ez a modell egy rugalmas és kényelmes eszközt biztosít a számunkra. Így egy eseményre több metódus is feliratkozhat, valamint a metódus lehet az eseményt előidéző objektum osztályának tagja, egy másik objektum tagja vagy egy másik osztály statikus metódusa. Az esemény előidézésére ténylegesen csak akkor kerül sor, ha van legalább egy feliratkozó.

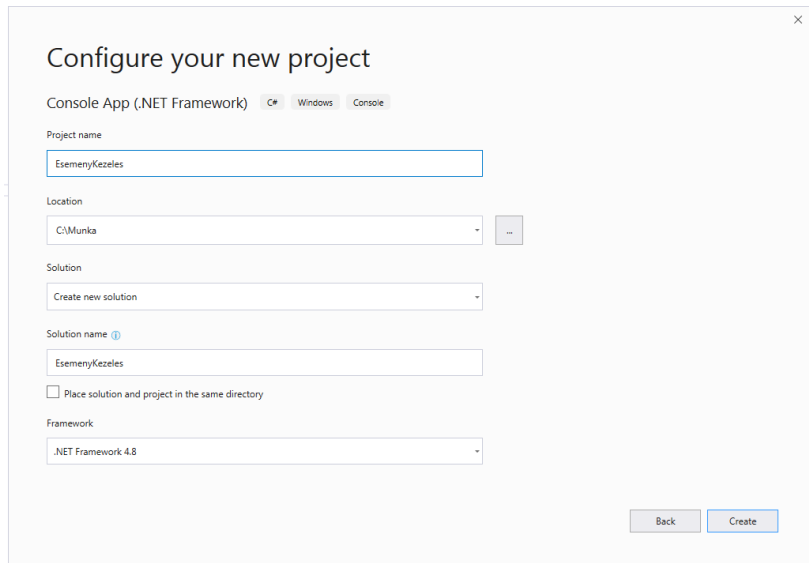
### 1.4.2. Feladat és főbb lépések

A feladat egy ötöslottó játékot szimuláló alkalmazás fejlesztése, ami megadott számú sorsolást hajt végre. Készíteni kell játékos objektumokat, amelyek megadott számokkal játszanak, és a játék során követni kell a találatok számát. A megadott számú sorsolást követően a játék véget ér, és erről értesíteni kell a játékosokat.

Az alkalmazás fejlesztése során az alábbi főbb lépések szükségesek.

- A sorsolásokat követően a játékosok számára átadott paraméterek megtervezése és a kapcsolódó implementációs lépések.
- Az esemény és eseménykezelő típusok deklarálása.
- A lottóhúzást megvalósító osztály (közzétevő) elkészítése.
- A játékosokat modellező osztály (feliratkozó) elkészítése.
- Objektumok létrehozása és a Main metódus elkészítése.

A feladatot egy konzolalkalmazásban oldjuk meg. A projekt és a megoldás neve legyen `EsemenyKezeles`. A továbbiakban az állománynevek mindig legyenek ékezet nélküliek, az azonosítók mindig legyenek ékezetesek. Ennek érdekében a New Project ablakban ékezet nélkül adjuk meg a projekt nevet.

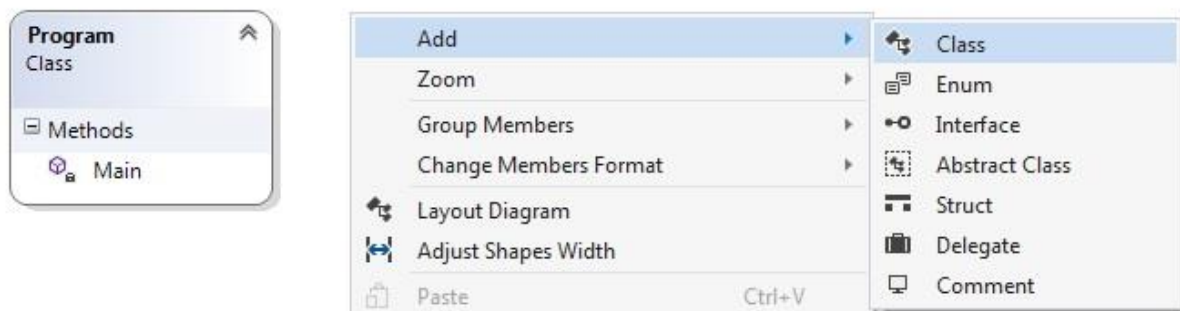


47. ábra.

Készítsünk egy osztálydiagramot is a projekthez. Ehhez a Solution Explorerben kattintsunk jobb egérgombbal projekt nevére, és a gyorsmenüben válasszuk ki a View/Class Diagram menüpontot. Válasszuk ki a teljes részletezettségű megjelenítést az eszköztáron a Display Full Signature ikonra kattintva.

### 1.4.3. Paraméterek átadása

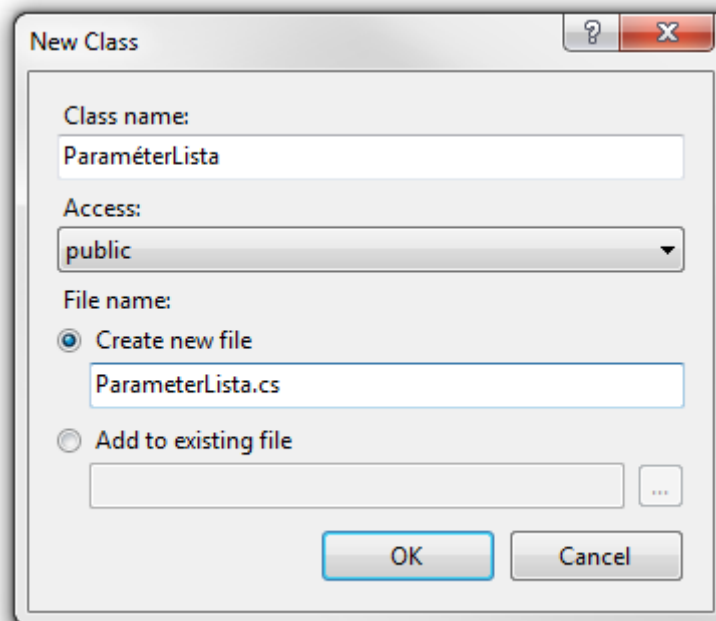
A sorsolás végrehajtását jelző esemény bekövetkezésekor át kell adni paraméterként az eseménykezelő metódusnak a húzás sorszámát (egész szám) és a kihúzott számokat (egész értékeket tartalmazó tömb). A paramétereket nem adhatjuk át közvetlenül, csak egy objektumba csomagolva, amely objektum az EventArgs osztály leszármazottja kell legyen.



48. ábra.

Ennek érdekében definiálunk egy ParaméterLista nevű osztályt, ami az EventArgs leszármazottja lesz. Az osztálydiagram fehér területén kattintsunk jobb egérgombbal, majd a gyorsmenüben válasszuk az Add/Class menüpontot. Az osztály neve legyen ParaméterLista (az állománynév ne legyen ékezetes!). A származtatást nem lehet vizuálisan megadni, ezért áttérünk kódnézetbe (jobb egérgomb az osztálynéven, majd ViewCode), és begépeljük a megfelelő kódot.

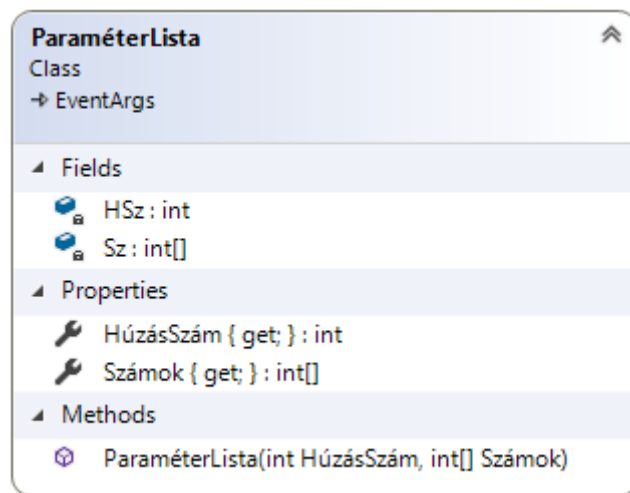




49. ábra.

```
public class ParaméterLista : EventArgs
{
}
```

Ezt követően visszatérünk az osztálydiagramba, kijelöljük a `ParaméterLista` osztályt, majd létrehozuk a szükséges adattagokat, tulajdonságokat és a konstruktort az ábrának megfelelően a `ClassDetails` ablakban.



50. ábra.

A tulajdonságok szerepe az lesz, hogy lekérdezhetővé teszik a háttérben levő korlátozott elérhetőségű két adattagot. Az adatagoknak csak egyszer (inicializáláskor – az objektum létrehozásakor) kell értéket adni, ezért `set` elérő nem szükséges, a kezdőérték megadása a konstruktorban történik. A `get` elérők és a konstruktor kódjának megírása után a `ParaméterLista` osztály a következő lesz:

```
///

```

```
/// </summary>
public class ParaméterLista : EventArgs
{
    /// <summary> A húzás sorszáma</summary>
    private int HSz;
    /// <summary> A kihúzott számokat tároló tömb</summary>
    private int[] Sz;
    /// <summary>
    /// Inicializálja az adattagokat
    ///</summary>
    /// <param name="HúzásSzám">A húzás sorszáma</param>
    /// <param name="Számok">A kihúzott számok</param>
    public ParaméterLista(int HúzásSzám, int[] Számok)
    {
        HSz = HúzásSzám;
        Sz = Számok;
    }
    /// <summary>
    /// Tulajdonság a húzásszám lekérdezésére
    /// </summary>
    public int HúzásSzám
    {
        get { return HSz; }
    }
    /// <summary>
    /// Tulajdonság a kihúzott számok lekérdezésére
    /// </summary>
    public int[] Számok
    {
        get { return Sz; }
    }
}
```

#### 1.4.4. Események és eseménykezelő típusok

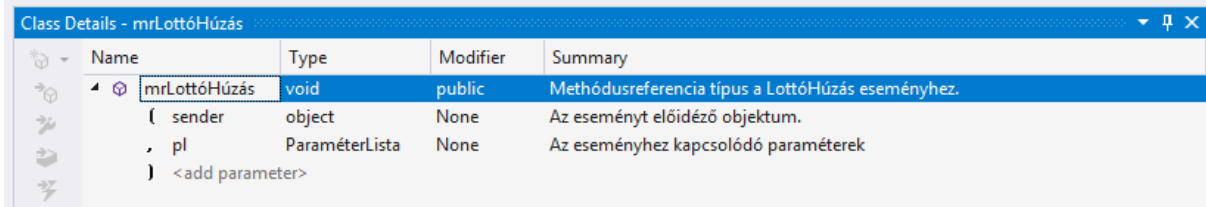
A játékhoz két eseményfajta szükséges:

- Minden lottóhúzást egy `LottóHúzás` esemény jelez.
- A játék végét `Vége` esemény jelzi.

Az eseménykezelő metódusok formális paraméterlistája alapvetően kétféle lehet. Mindkét esetben az első (vagy egyetlen) paraméter típusa `object` kell legyen, és ebben a paraméterben meg kell kapnia az eseménykezelőnek az eseményt előidéző objektum referenciáját. Amennyiben nem kell külön paramétereket átadnunk az eseménykezelőnek, akkor az eseménykezelőnk egyparaméteres lesz, és ha át kell adnunk paramétert, akkor pedig kétparaméteres lesz. A kétparaméteres esetben a második paraméternek az `EventArgs` osztály leszármazottjának kell lennie.

A `LottóHúzás` eseményhez kapcsolódóan kell átadnunk a lottóhúzásra vonatkozó adatokat (sorszám és kihúzott számok). Ezért itt az eseménykezelő kétparaméteres lesz. Az első paraméter az eseményt előidéző objektum referenciája, míg a második egy `ParaméterLista` típusú objektum referenciája.

Az alkalmazott eseménykezelő metódus típusok kiválasztása után metódusreferencia típusokat deklarálunk alkalmazásunk névterében a létező osztályokon kívül. A metódusreferencia típust új osztályhoz hasonlóan hozunk létre, csak most a Delegate menüpontot választjuk a gyorsmenüben. Neve legyen `mrLottóHúzás`. Állítsuk be a paraméterek nevét és típusát az ábra szerint a ClassDetails ablak segítségével.



51. ábra.

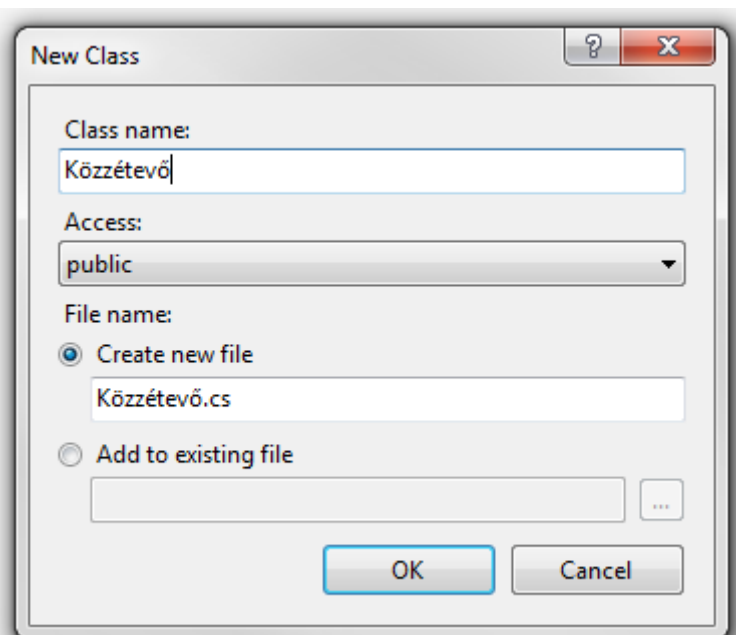


52. ábra.

A `Vége` eseménynél nem kell külön paramétereket átadni, ezért az eseménykezelő itt egyparáméteres lesz. Az előzőekhez hasonlóan hozunk létre egy Delegate komponenst az osztálydiagram fehér területén, majd definiáljuk a metódusreferencia típust az ábrának megfelelően.

#### 1.4.5. A lottóhúzást megvalósító osztály (közzétevő)

A lottóhúzást megvalósító osztály létrehozása érdekében adjunk hozzá egy Class komponenst az osztálydiagram fehér területéhez. Nevezzük el `Közzétevőnek`. Elsőként készítsük el a tárolni kívánt adatokhoz szükséges adattagokat.

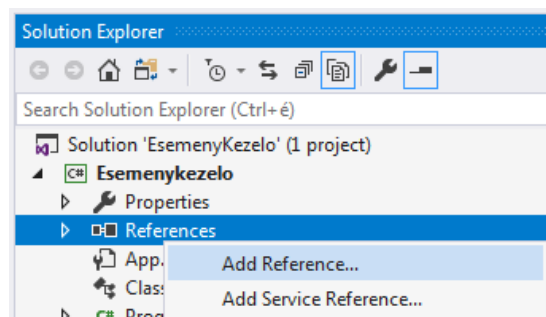


53. ábra.

Feladatunk megoldásához tárolnunk kell a lottóhúzások megengedett számát (MegengedettHúzásSzám) és az aktuális lottóhúzás sorszámát (HúzásSzám). Szükségünk lesz egy véletlenszám generátor objektumra (Véletlen) és egy időzítő Timer objektumra (Időzítő), ugyanis a sorsolásokat meghatározott időközönként hajtjuk végre.

A két esemény előidézéséhez két osztálytagra lesz szükségünk, amelyek általános elnevezése event (esemény). A két esemény (LottóHúzás és Vége) típusát az előzőekben deklarált metódusreferenciák határozzák meg.

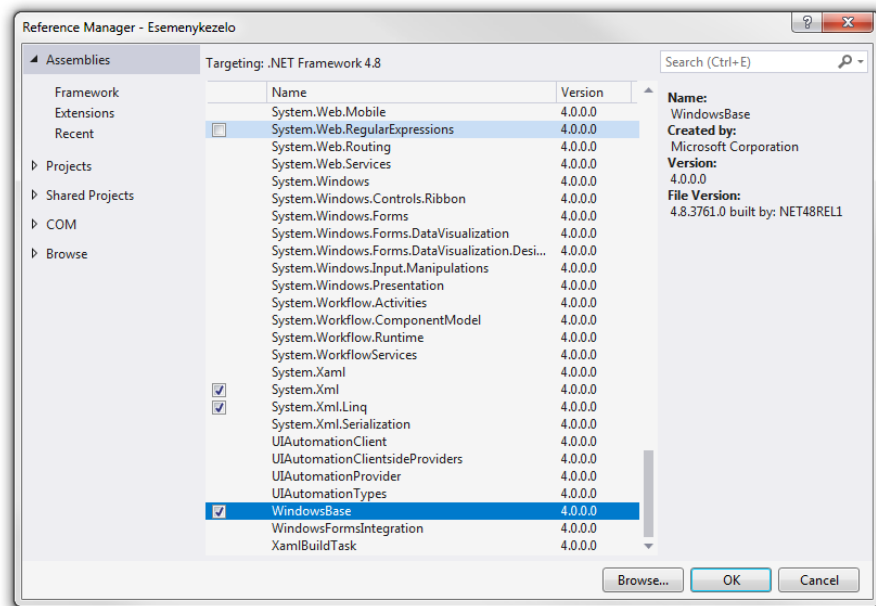
Az osztály két metódussal fog rendelkezni. A konstruktor biztosítja az adattagok inicializálását és az időzítés beállítását. A másik metódus (IdőzítésEseményKezelő) az időzítő Tick eseménye által aktivizálva véletlenszerűen előállítja a lottószámokat.



54. ábra.

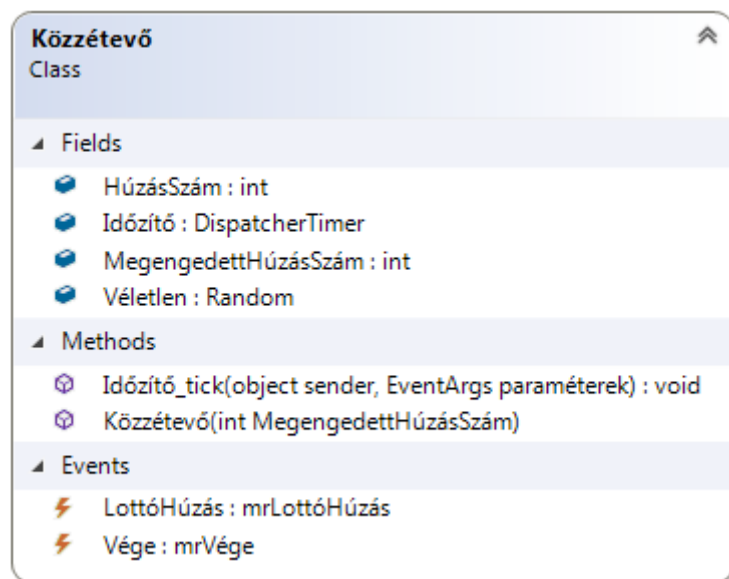
Az időzítéshez alkalmazott DispatcherTimer komponens a System.Windows.Threading névtérben található. Használatához azonban nem elegendő a névtér megadása, mivel az őt tartalmazó szerelvény (DLL) nem része a konzolalkalmazásokhoz automatikusan összeállított szerelvény csoportnak, ezért fel kell vennünk azt a referenciák közé. Ehhez a Solution Explorerben kattintsunk a References mappán jobb egérgombbal, majd válasszuk ki az Add Reference ... menüpontot. Kis várakozás után megjelenik az Add Reference párbeszédablak.

Itt kiválasztjuk az Assemblies menüpontot a bal oldalon, és azon belül a Framework-öt, majd megkeressük a listában a WindowsBase sort, bejelöljük, és kattintunk az OK gombon.



55. ábra.

A referencia kiegészítést követően készítjük el a ClassDetails ablak segítségével az alábbi ábrának megfelelően az osztályt az `Időzítő_Tick` eseménykezelő kivételével. Ezen eseménykezelő vázát a Visual Studioval fogjuk generáltatni a konstruktor megírása során.



56. ábra.

A konstruktor kódja a következő:

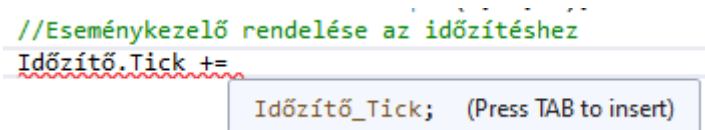
```

/// <summary>
/// Konstruktor. "Feliratkoztatja" Az IdőzítésEseménykezelő
/// függvényt a Timer objektum Tick(óraütés) eseményére. Indítja az időzítőt.
/// </summary>
/// <param name="MegengedettHúzásSzám">A lottóhúzások megengedett
/// száma.
///</param>

```

```
public Közzetevő(int MegengedettHúzásSzám)
{
    this.MegengedettHúzásSzám = MegengedettHúzásSzám;
    //Véletlenszámokat generáló objektum neve
    Véletlen = new Random();
    //Létrehozzuk az időzítő objektumot
    Időzítő = new DispatcherTimer();
    //Az időzítést 1 másodpercre állítjuk
    Időzítő.Interval = new TimeSpan(0, 0, 1);
    //Eseménykezelő rendelése az időzítéshez
    Időzítő.Tick += Időzítő_tick;
    //Időzítő indítása
    Időzítő.Start();
}
```

A konstruktor megírása során, amikor a „Időzítő.Tick +=” részhez érünk, akkor a Visual Studio felkínálja, hogy a TAB billentyű kétszeri megnyomása esetén automatikusan generálja a kapcsolódó eseménykezelő metódus vázát.



57. ábra.

A Közzetevő típusú objektum létrehozását követően előre beállított időközönként meghívódik az Időzítő eseménykezelője, és előállítja a lottószámokat. A feladatot megvalósító kód a következő:

```
/// <summary>
/// Eseménykezelő függvény a timer objektum időzítés eseményéhez. Ha
/// még nem értük el a maximális húzásszámot, akkor minden
/// "óraütéskor" lottóhúzást szimulál. A kisorolt öt számot a
/// Lottóhúzás esemény paraméterként teszi közzé. Ha elérjük a
/// maximális húzásszámot, akkor Vége eseményt generál. Ha elértük a
/// maximális húzásszámot, akkor leállítja a Timer objektumot
/// (időzítőt).
/// </summary>
/// <param name="sender">Az eseményt előidéző Timer
/// objektum.</param>
/// <param name="paraméterek">A Paramétereket tartalmazó
/// objektum.</param>
public void Időzítő_tick(object sender, EventArgs paraméterek)
{
    //Aktuális húzásszám meghatározása.
    HúzásSzám++;
    //Ha a húzásszám meghaladta a megengedett értéket, akkor
    //vége a játéknak
    if( HúzásSzám > MegengedettHúzásSzám )
    {
        //Leállítjuk az időzítőt.
        Időzítő.Stop();
        //Ha van feliratkozó a Vége eseményre, akkor előidézzük az //eseményt.
        if (Vége != null) Vége(this);
    }
    else
    {
        //Végrehajtjuk a sorsolást
        int[] Számok = new int[5];
```

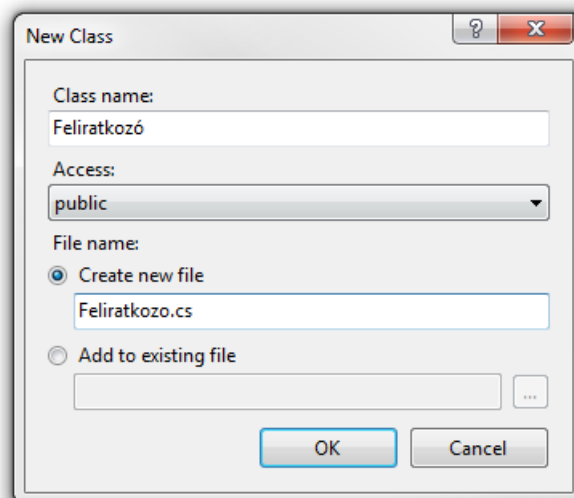
```

//Beletesszük a 90 számot egy listába
List<int> Összes = new List<int>();
for (int i = 1; i <= 90; i++) Összes.Add(i);
//Kiveszünk ötöt egyesével úgy, hogy a kivett értéket //eltávolítjuk a
listáról.
for(int i = 0; i < 5; i++)
{
    //Előállítjuk a soron következő érték sorszámát.
    int Aktuális = Véletlen.Next(Összes.Count);
    //Kimásoljuk az értéket.
    Számok[i] = Összes[Aktuális];
    //Eltávolítjuk a listából a kimásolt értéket.
    Összes.RemoveAt(Aktuális);
}
//Ha van feliratkozó, akkor létrehozuk a lottó eseményt.
if (LottóHúzás != null)
    LottóHúzás(this, new ParaméterLista(HúzásSzám, Számok));
}
}

```

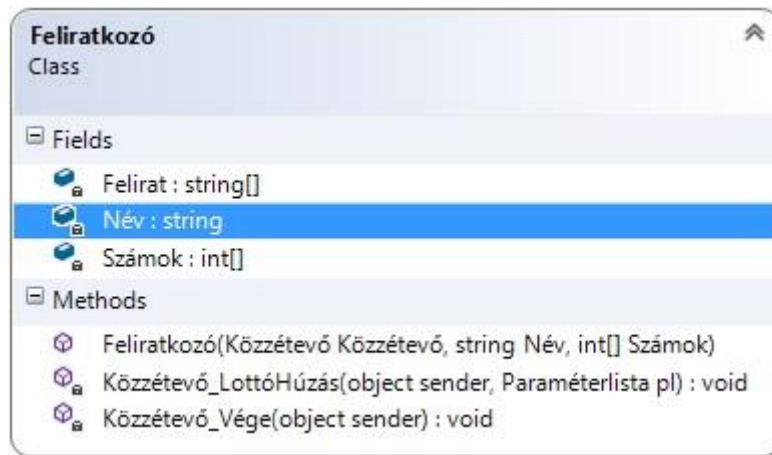
#### 1.4.6. A játékosokat modellező osztály

A következő osztály feladata egy játékos modellezése lesz. A játékosnak vannak saját számai, és feliratkozik a lottó játékra, azaz a két eseményre. Hozunk létre az osztálydiagramban egy új osztályt `Feliratkozó` néven.



58. ábra.

Hozunk létre adattagokat a játékos nevének (`Név`) és megjátszott számainak (`Számok`) tárolására. Továbbá egy `string` tömbre (`Felirat`) is szükségünk lesz, amiben tároljuk, hogy mit kell kiírni a konzolra az egyes találatzszámok értékeléseként.



59. ábra.

A két eseménykezelő metódus vázát az előzőekben látottaknak megfelelően automatikusan generáltassuk a Visual Studioval. A LottóHúzás eseményt a Közzétevő\_LottóHúzás metódus fogja kezelni. Feladata a találatok számának meghatározása és az elért találatszám kiírása a konzolra.

```

/// <summary>
/// Eseménykezelő a lottóhúzás eseményhez. Meghatározza a találatok /// számát.
/// Kiírja konzolra az eredményt.
/// </summary>
/// <param name="sender">Az eseményt előidéző objektum.</param>
/// <param name="pl">Kisorsolt számok.</param>
private void Közzétevő_LottóHúzás(object sender, Paraméterlista pl)
{
    int Találatok = 0;
    //Meghatározzuk a találatok számát.
    for (int i = 0; i < 5 && Találatok < 5; i++)
        for (int j = 0; j < 5 && Találatok < 5; j++)
            if (pl.Számok[i] == Számok[j])
                Találatok++;
    //Eredmény kiírása a konzolra.
    Console.WriteLine(pl.HúzásSzám + ". " + Név + " " + Felirat[Találatok]);
}

```

A Vége eseményt a Közzétevő\_Vége metódus fogja kezelni. Feladata csupán annyi, hogy a Vége esemény bekövetkezésekor kiírjon egy üzenetet a konzolra.

```

/// <summary>
/// Eseménykezelő a lottóhúzás végét jelző eseményhez. Kiírja a
/// konzolra, a vége eseményt.
/// </summary>
private void Közzétevő_Vége(object sender)
{
    Console.WriteLine(Név + " befejezte a játékot!");
}

```

A konstruktorban definiáljuk az egyes találatszámokhoz kapcsolódó üzeneteket, és ugyancsak itt iratkoztatjuk fel a két eseménykezelőt a megfelelő eseményekre. Ehhez a konstruktornak meg kell kapnia a Közzétevő típusú objektum referenciáját. Emellett még átveszi a játékos nevét és kedvenc számait is.

```

/// <summary>

```



```

/// Konstruktor. "Feliratkoztatja" a két eseménykezelő metódust a /// LottóHúzás és
a Vége eseményre.
/// </summary>
/// <param name="Közzétevő">A közzétevő objektum.</param>
/// <param name="Név">A játékos neve.</param>
/// <param name="Számok">A megadott számok.</param>
public Feliratkozó(Közzétevő Közzétevő, string Név, int[] Számok)
{
    this.Név = Név;
    this.Számok = Számok;
    //Eseménykezelő metódusok feliratkoztatása.
    Közzétevő.LottóHúzás += new mrLottóHúzás(Közzétevő_LottóHúzás);
    Közzétevő.Vége += new mrVége(Közzétevő_Vége);
    //Az eredményekhez kapcsolódó feliratok definiálása
    Felirat = new string[6];
    Felirat[0] = "Sajnos nem volt találat!";
    Felirat[1] = "Egy találat volt!";
    Felirat[2] = "Kettő találat volt!";
    Felirat[3] = "Három találat volt!";
    Felirat[4] = "Négy találat volt!";
    Felirat[5] = "Öt találat volt!";
}

```

#### 1.4.7. A Program osztály és a Main metódus

Konzol alkalmazásunk automatikusan kapott osztálya a `Program`. Feladata az lesz, hogy működésbe hozza programunkat, itt hozzuk létre az egyes objektumokat és indítjuk be a lottó játékot. Mivel a konzolalkalmazások alapból nem rendelkeznek eseménykezeléssel, ezért egy ciklust építünk majd a `Main` metódusba a feladat megoldására.



60. ábra.

A leállást úgy oldjuk meg, hogy a ciklusból történő kilépést egy logikai adattag (`Kilép`) értékéhez kötjük, aminek kiinduló értéke hamis, és készítünk egy metódust (`Kt_Vége`), amit feliratkoztatunk a `Vége` eseményre. Ez az eseménykezelő metódus meghívásakor átállítja igazra a logikai változó értékét. Mivel a `Program` osztályból nem készül objektum, ezért úgy az adattag, mint az eseménykezelő metódus statikus kell legyen.

Az előzőekhez hasonlóan a `Kt_Vége` metódus vázát generáltassuk le automatikusan a Visual Studioval. Az adattagot készítjük el grafikus eszközökkel, majd gépeljük be a még szükséges kódrészleteket. Az osztály teljes kódja az alábbi:

```
class Program
```

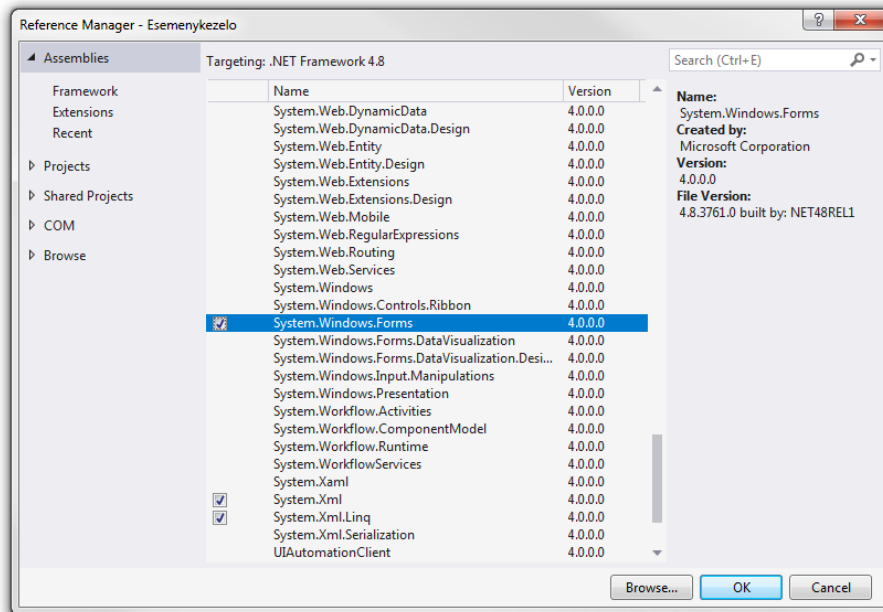
```
{
    /// <summary>
    /// Kiléphetünk e programból?
    /// </summary>
    static bool Kilép = false;
    /// <summary>
    /// Itt kezdődik a program végrehajtása
    /// </summary>
    static void Main(string[] args)
    {
        //Létrehozuk a közzétevő ( lottóhúzást szimuláló)
        //objektumot. 5 Lottóhúzást engedélyezünk.
        Közzétevő Kt = new Közzétevő(5);
        //Definiálunk három játékost
        Feliratkozó J1 = new Feliratkozó(Kt, "Sheldon Cooper ", new int[] { 1, 2, 3,
        4, 5 });
        Feliratkozó J2 = new Feliratkozó(Kt, "Jon Snow ", new int[] { 12, 13, 14, 15,
        16 });
        Feliratkozó J3 = new Feliratkozó(Kt, "The Imp", new int[] { 33, 34, 35, 36, 36
        });
        //Feliratkozttatjuk eseménykezelőnk a Vége eseményre.
        Kt.Vége += Kt_Vége;
        //Az eseménykezelő mechanizmus működtetése a lottóhúzás
        //vége eseményig.
        while (!Kilép)
        {
            Application.DoEvents();
        }
        Console.ReadLine();
    }
    /// <summary>
    /// Eseménykezelő a lottóhúzás végét jelző eseményhez.
    /// </summary>
    /// <param name="sender">Az eseményt előidéző objektum.</param>
    static void Kt_Vége(object sender)
    {
        Kilép = true;
    }
}
```

Az Application osztály a System.Windows.Forms névtérben található, ezért használata érdekében egészítsük ki a using csoportot a Program.cs elején egy

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

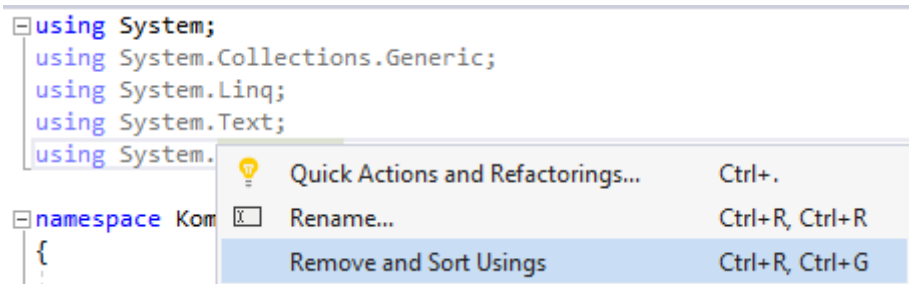
utasítással.

Ezután a felhasznált szerelvények listájába (References) vegyük fel a System.Windows.Forms szerelvényt.



61. ábra.

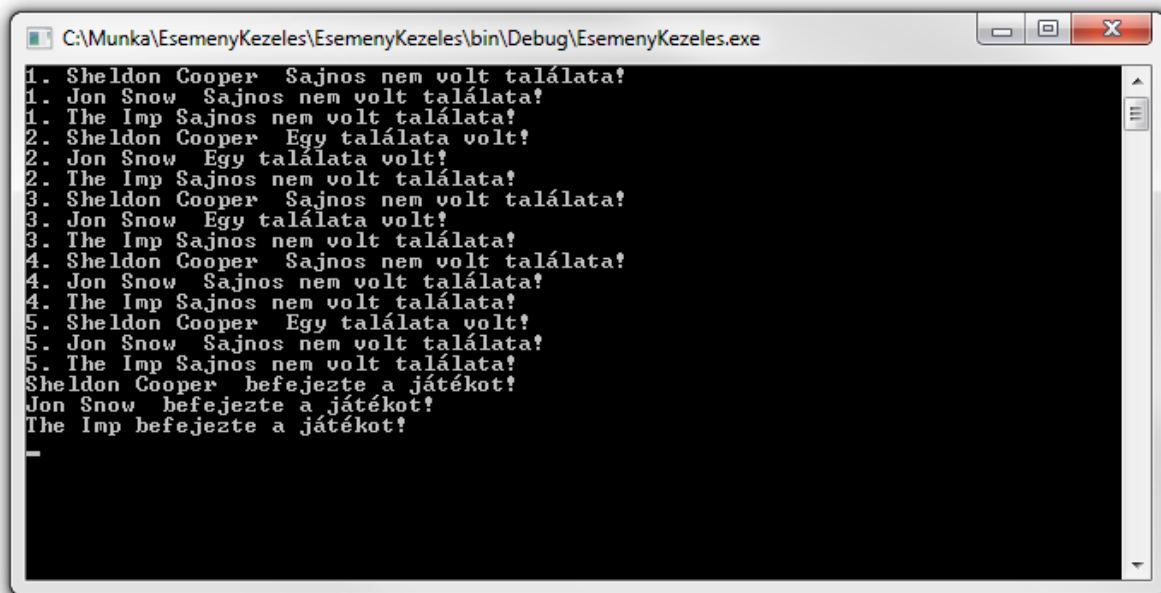
Programkódunk átláthatóságát azal is növelhetjük, hogy eltávolítjuk minden forrásállomány elejéről a felesleges `using` utasításokat. Ezt a legegyszerűbben úgy tehetjük meg, hogy a kódszerkesztőben egymás után megnyitunk minden C# forrásállományt, és a jobb egérgomb kattintására előjövő gyorsmenüben a “Remove and Sort Usings” pontot választjuk.



62. ábra.

Futtassuk le a programot.

A konzolon az alábbihoz hasonló eredmény jelenik meg:



```
C:\Munka\EsemenyKezeles\EsemenyKezeles\bin\Debug\EsemenyKezeles.exe
1. Sheldon Cooper Sajnos nem volt találata!
1. Jon Snow Sajnos nem volt találata!
1. The Imp Sajnos nem volt találata!
2. Sheldon Cooper Egy találata volt!
2. Jon Snow Egy találata volt!
2. The Imp Sajnos nem volt találata!
3. Sheldon Cooper Sajnos nem volt találata!
3. Jon Snow Egy találata volt!
3. The Imp Sajnos nem volt találata!
4. Sheldon Cooper Sajnos nem volt találata!
4. Jon Snow Sajnos nem volt találata!
4. The Imp Sajnos nem volt találata!
5. Sheldon Cooper Egy találata volt!
5. Jon Snow Sajnos nem volt találata!
5. The Imp Sajnos nem volt találata!
Sheldon Cooper befejezte a játékot!
Jon Snow befejezte a játékot!
The Imp befejezte a játékot!
```

63. ábra.

#### 1.4.8. További feladatok

Írjuk át úgy a programot, hogy

- a kihúzott számok jelenjenek meg a konzolon,
- a kihúzott számok legyenek növekvő sorba rendezve,
- minden játékos esetén az eltalált számok jelenjenek meg a konzolon.

## 1.5. Sorosítás (szerializáció) és helyreállítás

Cél: a memóriában tárolt adatok egyszerű lemezre mentése és visszatöltése.

A sorosítás során létrehozunk egy állományt és egy sorosítást kezelő objektumot. Ez gondoskodik arról, hogy az általunk kiválasztott objektumban tárolt adatok az állományba kerüljenek sorban egymás után. Láncolt lista és körkörös hivatkozások kezelésére is képes.

A helyreállítás során megnyitjuk a korábban lementett adatokat tartalmazó állományt, és létrehozunk egy sorosítást kezelő objektumot. Ez gondoskodik a lementett adatok visszatöltéséről. Ha a mentés óta megváltoztattuk az adatok típusát megadó osztályt, akkor a visszatöltés nem lehetséges.

Megoldások:

- Bináris
- SOAP-XML
- XML

### 1.5.1. Bináris sorosítás és helyreállítás

#### Szükséges névterek

```
using System.IO;
using System.Collection;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

#### Attribútumok

Sorosításra kijelölés osztály neve előtt:

[[Serializable](#)]

Sorosítani nem kívánt részek (adattagok, tulajdonságok) előtt:

[[NonSerialized](#)]

#### Mentés

```
Stream st = File.Create("fajlnév");
BinaryFormatter bf = new BinaryFormatter();
bf.Serialize(st, objektum); //ezt az objektumot akarjuk sorosítani
st.Close();
```

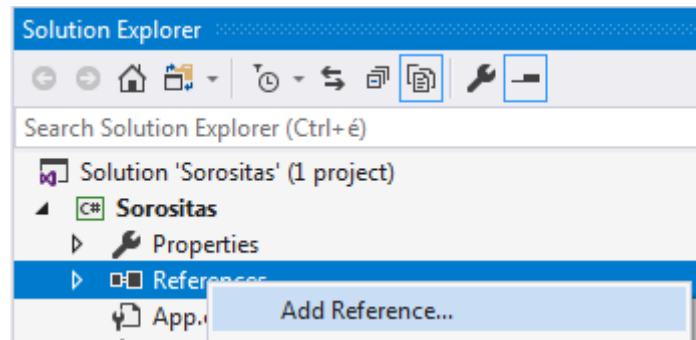
#### Betöltés

```
Stream st = File.OpenRead("fajlnév");
BinaryFormatter bf = new BinaryFormatter();
Osztály_Név objektum = (Osztály_Név)bf.Deserialize(st);
st.Close();
```

### 1.5.2. SOAP-XML sorosítás és helyreállítás

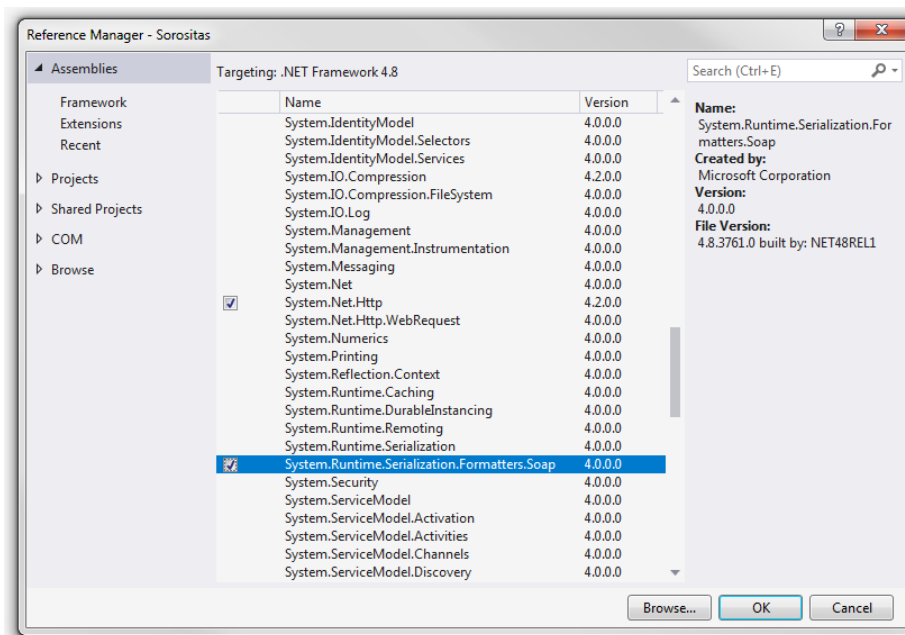
#### Szükséges névterek

```
using System.IO;
using System.Collection;
using System.Runtime.Serialization;
```



64. ábra.

A Soap-ot fel kell venni a hivatkozások közé. A Solution Explorerben jobb egérgombbal kattintunk a References-en, Add Reference..., Framework fül, System.Runtime.Serialization.Formatters.Soap kiválasztása, majd kattintás az OK gombon.



65. ábra.

```
using System.Runtime.Serialization.Formatters.Soap;
```

### Attribútumok

Sorosításra kijelölés osztály neve előtt:

```
[Serializable]
```

Sorosítani nem kívánt részek (adattagok, tulajdonságok) előtt:

```
[NonSerialized]
```

### Mentés

```
Stream st = File.Create("fájlnev");  
SoapFormatter bf = new SoapFormatter();  
bf.Serialize(st, objektum);  
//ezt az objektumot akarjuk sorosítani  
st.Close();
```

**Betöltés**

```
Stream st = File.OpenRead("fájlnév");
SoapFormatter bf = new SoapFormatter();
Osztály_Név objektum = (Osztály_Név)bf.Deserialize(st);
st.Close();
```

**1.5.3. XML sorosítás és helyreállítás**

Nem tudja lementeni a korlátozott hozzáférésű tagokat és a két- vagy többdimenziós tömböket pl. string tömböt vagy ArrayList-et. Csak set elérővel is rendelkező tulajdonságot ment le.

**Szükséges névtér**

```
using System.Xml.Serialization;
```

**Attribútum**

Ha valamely adattagok vagy tulajdonságot nem akarunk sorosítani, akkor a definíciója elé írjuk:

```
[XmlIgnore]
```

**Mentés**

```
Stream st = File.Create("fájlnév.xml");
XmlSerializer xs = new XmlSerializer(typeof(Osztály_Név));
xs.Serialize(st, objektum);
// Az objektum típusa Osztály_Név
st.Close();
```

**Betöltés**

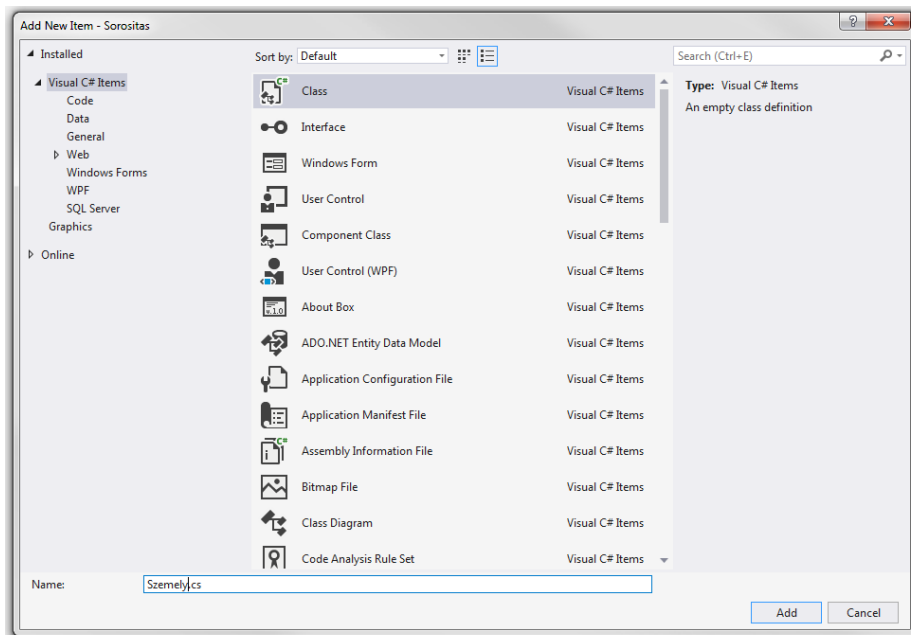
```
Stream st = File.OpenRead("fájlnév.xml");
XmlSerializer xs = new XmlSerializer(typeof(Osztály_Név));
objektum = (Osztály_Név)xs.Deserialize(st);
// Az objektum típusa Osztály_Név
st.Close();
```

**1.5.4. Példa**

Készítsünk egy alkalmazást a különböző sorosítási és helyreállítási módok bemutatására. Az alkalmazás típusa (Templates) legyen Windows Application, a neve: Sorositas. A formot nevezzük át frmSorositas-ra (Name=frmSorositas, Text=Sorosítás bemutatása), az őt tartalmazó állományt frmSorositas.cs-re.

**Mintaosztály**

Készítsünk egy mintaosztályt. Az ebből készült objektumokat fogjuk lementeni és visszatölteni. Project menü, Add Class ..., name: Szemely.cs, Add.



66. ábra.

Az XML típusú sorosításhoz helyezzük el a Személy osztályt tartalmazó kódállomány elején az alábbi két sort:

```
using System.Xml.Serialization;
using System.Xml;
```

Az osztály definícióját az alábbiak szerint készítjük el.

```
/// <summary>
/// A sorosítás bemutatásához készített mintaosztály
/// </summary>
/// Az alábbi attribútum jelzi, hogy az osztályt binárisan és SOAP-
/// XML sorosítással menthető
[Serializable]
public class Szemely
{
    //Ezt a tulajdonságot nem menti le az XML sorosítás a hozzáférés
    //korlátozás miatt
    private string Név;
    public string Neptun;
    public int Jegy;
    //Ha az alábbi sor nincs megjegyzésben, akkor a tulajdonságot
    //nem menti le az XML sorosítás
    //[XmlIgnore]
    public string Teljes
    {
        get
        {
            return Név + "-" + Neptun + "-" + Jegy.ToString();
        }
        set{;}
    }
}
//Paraméter nélküli konstruktor, meghívja a háromparaméteres
//konstruktor
public Szemely() : this("", "", 0) { }
//Háromparaméteres konstruktor
public Szemely(string Név, string Neptun, int Jegy)
{
```



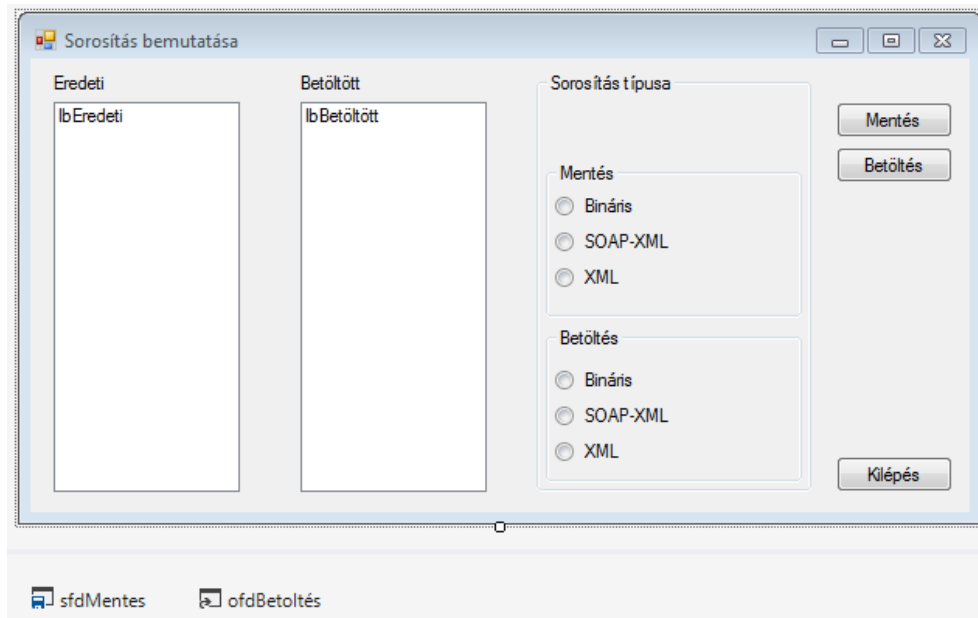
```

    this.Név = Név;
    this.Neptun = Neptun;
    this.Jegy = Jegy;
}
}

```

## Felület kialakítása

Nyissuk meg a frmSorositas-t tervezési nézetben, és helyezzük el rajta az ábrának megfelelően az alábbi komponenseket:



67. ábra.

- Címke (Label), Text=Eredeti
- Címke (Label), Text=Betöltött
- Listaablak (ListBox), Name=lbEredeti
- Listaablak (ListBox), Name=lbBetöltött
- Csoportablak (GroupBox), Text=Sorosítás típusa
- Csoportablak (GroupBox), Text=Mentés, ezt az első csoportablakra helyezzük el
- Csoportablak (GroupBox), Text=Betöltés, ezt az első csoportablakra helyezzük el
- Választógomb (RadioButton), Name=rbBinarisMent, Text=Bináris
- Választógomb (RadioButton), Name= rbSoap\_XmlMent, Text=SOAP-XML
- Választógomb (RadioButton), Name= rbXmlMent, Text=XML
- Választógomb (RadioButton), Name=rbBinarisBetolt, Text=Bináris
- Választógomb (RadioButton), Name= rbSoap\_XmlBetolt, Text=SOAP-XML
- Választógomb (RadioButton), Name=rbXmlBetolt, Text=XML
- Nyomógomb (Button), Name=btMentes, Text=Mentés

- Nyomógomb (Button), Name=btBetoltes, Text=Betöltés
- Nyomógomb (Button), Name=btKilepes, Text=Kilépés
- Állománymentési párbeszédablak (SaveFileDialog), Name=sfdMentes, Title=Mentés, InitialDirectory=.
- Állománymegnyitási párbeszédablak (OpenFileDialog), Name=ofdBetoltés, Title=Betöltés, InitialDirectory=.

### Szükséges névterek

Az frmSorositas kódjában a fejlesztőrendszer által generált névtérhivatkozások mellett az alábbi névterekre lesz szükség:

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;
using System.Xml.Serialization;
using System.Xml;
using System.Collections;
```

A 2. pontban ismertetett módon vegyük fel a hivatkozások közé a Soap-ot.

### A feladatot megvalósító kód

Definiáljunk egy felsorolás típust a sorosítás típusok beazonosítására. Ezt a Sorositas névtéren belül, de az osztálydefiníciókon kívül kell megtennünk.

```
public enum SorosításTípus { Bináris, SOAP_XML, XML};
```

A program működéséhez az alábbi adattagok definiálása szükséges a frmSorosítás osztályon belül.

```
/// <summary>
/// Az ablak mentésekor definiált objektumok tárolására szolgál.
/// </summary>
public ArrayList alEredetiAdatok;
/// <summary>
/// Betöltött objektumok tárolására szolgál.
/// </summary>
public ArrayList alBetöltöttAdatok;
/// <summary>
/// Mentés típusa.
/// </summary>
public SorosításTípus stMentésTípus;
/// <summary>
/// Betöltés típusa.
/// </summary>
public SorosításTípus stBetöltésTípus;
```

A frmSorosítás konstruktorában helyezzük el az alábbi kezdőértékadásokat.

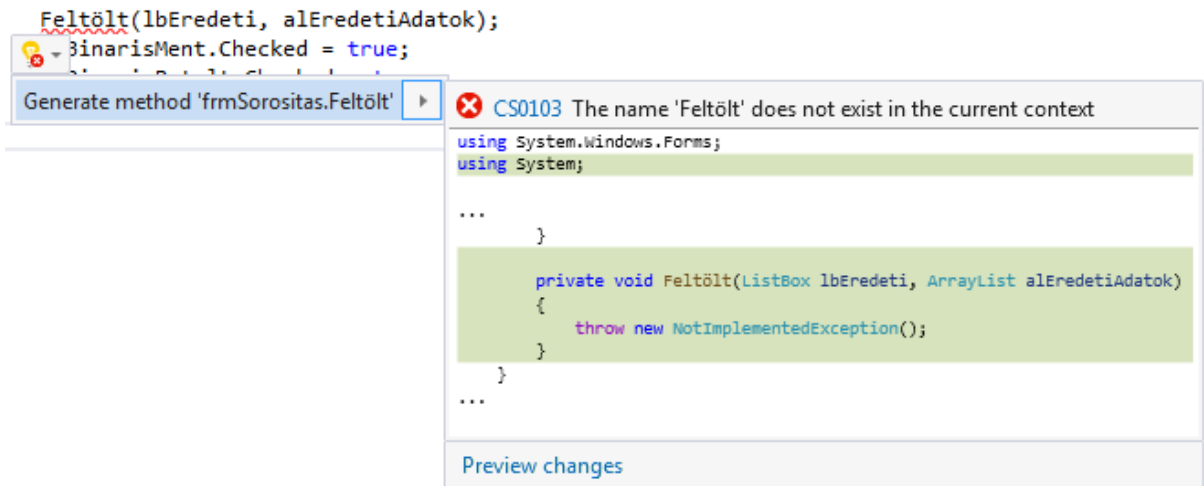
```
//Saját inicializálás
//Kezdeti és betöltött adatok tárolására szolgáló objektumok
//definiálása
alEredetiAdatok = new ArrayList();
alBetöltöttAdatok = new ArrayList();
//Kezdőadatok
alEredetiAdatok.Add(new Szemely("Kiss Ibolya", "AAA.BBB", 5));
```

```

alEredetiAdatok.Add(new Szemely("Nagy Bendegúz", "ABC.BBB", 4));
alEredetiAdatok.Add(new Szemely("Néhai Artúr", "ACA.BBB", 5));
//Feltölti az első paraméterként megadott listaablakot a második
//paraméterként megadott ArrayList adataival.
Feltölt(lbEredeti, alEredetiAdatok);
rbBinarisMent.Checked = true;
rbBinarisBetolt.Checked = true;

```

A Feltölt metódust meghívó utasítás kék hullámos vonallal jelenik meg, mivel ilyen nevű metódust még nem definiáltunk. A metódushívás neve alatt megjelenő jelen kattintva a fejlesztőrendszer felkínálja a metódus vázának generálását.



68. ábra.

A kapott kód az alábbi:

```

private void Feltölt(ListBox lbEredeti, ArrayList alEredetiAdatok)
{
    throw new NotImplementedException();
}

```

Írjuk meg a Feltölt metódust az frmSorosítás osztály definíciójában az alábbiak szerint:

```

/// <summary>
/// Feltölti az első paraméterként megadott listaablakot a második
/// paraméterként megadott ArrayList adataival
/// </summary>
/// <param name="lbEredeti">A feltölteni kívánt listaablak</param>
/// <param name="allista">A Szemely típusú elemeket tartalmazó
/// ArrayList</param>
private void Feltölt(ListBox lbEredeti, ArrayList allista)
{
    for(int i =0; i < allista.Count; i++)
    {
        Szemely sz = (Szemely)(allista[i]);
        lbEredeti.Items.Add(sz.Teljes);
    }
}

```

Készítsük el a Kilépés gomb eseménykezelőjének (Click esemény) definícióját.

```

/// <summary>
/// Kilépés az alkalmazásból
/// </summary>

```

```
private void btKilepes_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

Készítsük el a Mentéstípus kiválasztásakor aktivizálódó eseménykezelőt, amelyben beállítjuk az stMentésTípus változó értékét a kijelölt választógomb függvényében. Tervezési nézetben kiválasztjuk a Mentés választógomb csoportból a Bináris választógombot, a Properties ablakban az Events gombra kattintunk, és a CheckedChanged mezőbe beírjuk a rbMent\_CheckedChanged nevet, majd duplán kattintunk a mezőben. A kódszerkesztőben automatikusan megjelenik a függvény váza, ezt az alábbiak szerinti kóddal töltjük fel.

```
/// <summary>
/// Mentés típus kiválasztása a választógombok állapotának
/// függvényében
/// </summary>
private void rbMent_CheckedChanged(object sender, EventArgs e)
{
    if (rbBinarisMent.Checked)
        stMentésTípus = SorosításTípus.Bináris;
    if (rbSoap_XmlMent.Checked)
        stMentésTípus = SorosításTípus.SOAP_XML;
    if (rbXmlMent.Checked)
        stMentésTípus = SorosításTípus.XML;
}
```

Állítsuk be a Mentés csoport SOAP-XML és XML csoportjainál is CheckedChanged eseménykezelőként a rbMent\_CheckedChanged metódust.

Készítsük el a Betöltéstípus kiválasztásakor aktivizálódó eseménykezelőt, amelyben beállítjuk az stBetöltésTípus változó értékét a kijelölt választógomb függvényében. Tervezési nézetben kiválasztjuk a Betöltés választógomb csoportból a Bináris választógombot, a Properties ablakban az Events gombra *kattintunk*, és a CheckedChanged mezőbe beírjuk a rbBetölt\_CheckedChanged nevet, majd duplán *kattintunk* a mezőben. A kódszerkesztőben automatikusan megjelenik a függvény váza, ezt az alábbiak szerinti kóddal töltjük fel.

```
/// <summary>
/// Betöltés típus kiválasztása a választógombok állapotának
/// függvényében
/// </summary>
private void rbBetolt_CheckedChanged(object sender, EventArgs e)
{
    if (rbBinarisBetolt.Checked)
        stBetöltésTípus = SorosításTípus.Bináris;
    if (rbSoap_XmlBetolt.Checked)
        stBetöltésTípus = SorosításTípus.SOAP_XML;
    if (rbXmlBetolt.Checked)
        stBetöltésTípus = SorosításTípus.XML;
}
```

Állítsuk be a Betöltés csoport SOAP-XML és XML csoportjainál is CheckedChanged eseménykezelőként a rbBetölt\_CheckedChanged metódust.

Készítsük el a mentést megvalósító metódust. Ez a btMentés nyomógomb Click eseményének kezelője lesz. A metódust egy switch szerkezettel három részre tagoljuk. A mentés típusától függően beállítjuk a mentés párbeszédablak alapértelmezett állomány kiterjesztését és fájl típusait, megjelenítjük a párbeszédablakot, és ha Mentés gombbal zárta le a felhasználó, akkor végrehajtjuk az előzőekben megismert módon a sorosítást.

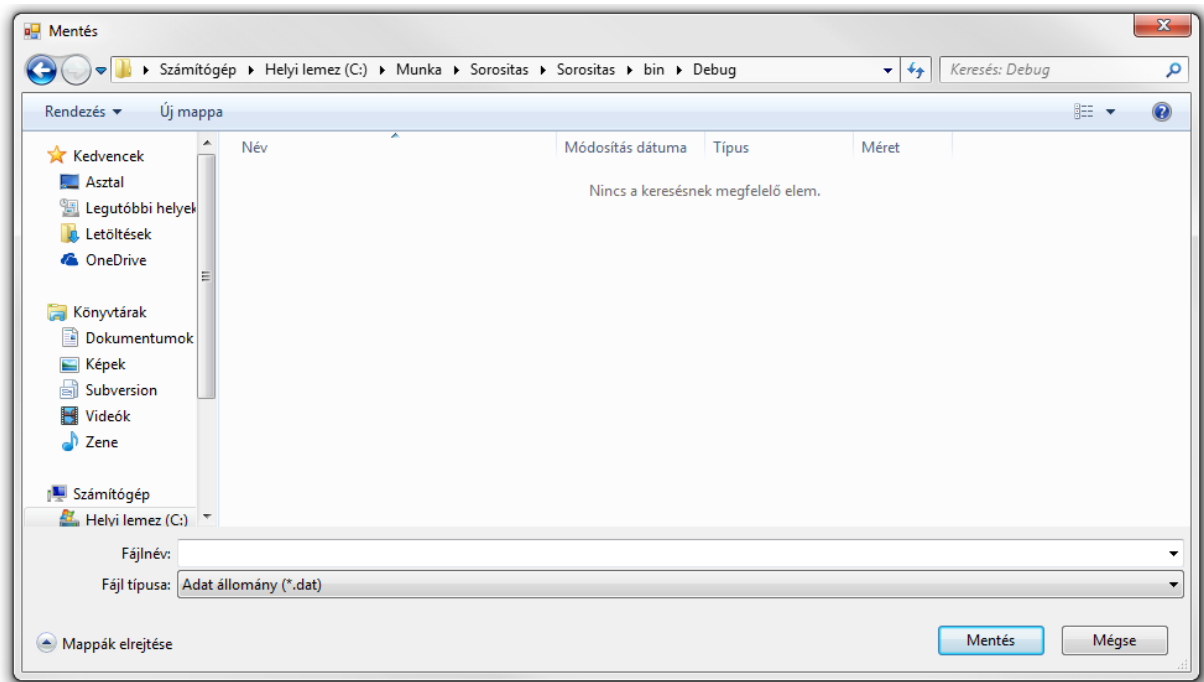
Mivel az ArrayList típus nem sorosítható XML sorosítással, ezért, ha a felhasználó az XML sorosítást választja, akkor létrehozunk egy Személy típusú objektumok tárolására alkalmas egydimenziós tömböt, ebbe belemásoljuk az ArrayList-ben tárolt Személy típusú objektumokat, majd a tömböt sorosítjuk.

A feladatot megoldó kód a következő:

```

/// <summary>
/// Az aktuális mentéstípusnak megfelelően sorosítja az
/// alEredetiAdatok ArrayList tartalmát
/// </summary>
private void btMentes_Click(object sender, EventArgs e)
{
    switch (stMentésTípus)
    {
        case SorosításTípus.Bináris:
            sfdMentes.DefaultExt = "*.dat";
            sfdMentes.Filter = "Adat állomány (*.dat)|*.dat|" + "Minden
            állomány (*.*)|*.*";
            if (sfdMentes.ShowDialog() == DialogResult.OK)
            {
                Stream st = File.Create(sfdMentes.FileName);
                BinaryFormatter sf = new BinaryFormatter();
                sf.Serialize(st, alEredetiAdatok);
                st.Close();
            }
            break;
        case SorosításTípus.SOAP_XML:
            sfdMentes.DefaultExt = "*.xml";
            sfdMentes.Filter = "XML állomány (*.xml)|*.xml|" +
            "Minden állomány (*.*)|*.*";
            if (sfdMentes.ShowDialog() == DialogResult.OK)
            {
                Stream st = File.Create(sfdMentes.FileName);
                SoapFormatter sf = new SoapFormatter();
                sf.Serialize(st, alEredetiAdatok);
                st.Close();
            }
            break;
        case SorosításTípus.XML:
            sfdMentes.DefaultExt = "*.xml";
            sfdMentes.Filter = "XML állomány (*.xml)|*.xml|" +
            "Minden állomány (*.*)|*.*";
            if (sfdMentes.ShowDialog() == DialogResult.OK)
            {
                Stream st = File.Create(sfdMentes.FileName);
                Szemely[] sz = new Szemely[alEredetiAdatok.Count];
                for (int i = 0; i < alEredetiAdatok.Count; i++)
                    sz[i] = (Szemely)alEredetiAdatok[i];
                XmlSerializer xs = new XmlSerializer (typeof(Szemely[]));
                xs.Serialize(st, sz);
                st.Close();
            }
            break;
    }
}

```



69. ábra.

Készítsük el a betöltést megvalósító metódust. Ez a `btBetöltés` nyomógomb `Click` eseményének kezelője lesz. A metódust egy `switch` szerkezettel három részre tagoljuk. Először töröljük az `lbBetölt` listaablak tartalmát, majd betöltés típusától függően beállítjuk a betöltés párbeszédablak alapértelmezett állomány kiterjesztését és fájl típusait, megjelenítjük a párbeszédablakot, és ha `Megnyitás` gombbal zárta le a felhasználó, akkor végrehajtjuk az előzőekben megismert módon a helyreállítást.

Mivel az `ArrayList` típus nem sorosítható XML sorosítással, ezért, ha a felhasználó ezt választja, akkor létrehozunk egy `Személy` típusú objektumok tárolására alkalmas egydimenziós tömböt, ebbe olvassuk be az adatokat a lemezről, majd ennek elemeit elhelyezzük az `alBetöltöttAdatok` `ArrayList`-ben, és a tartalmát megjelenítjük az `lbBetöltött` listaablakban a `Feltölt` metódus meghívásával.

A feladatot megoldó kód a következő:

```

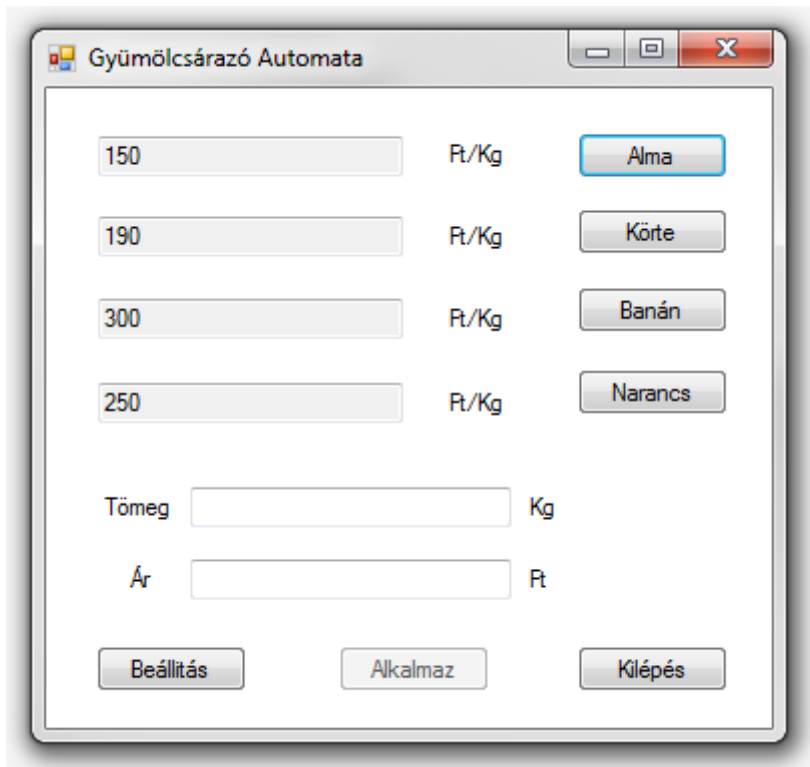
/// <summary>
/// Az aktuális betöltéstípusnak megfelelően beolvassa a lementett
/// adatokat az ArrayList típusú alBetöltöttAdatok objektumba.
/// Feltölti az lbBetöltött listaablakot.
/// </summary>
private void btnBetoltes_Click(object sender, EventArgs e)
{
    lbBetöltött.Items.Clear();
    switch (stBetöltésTípus)
    {
        case SorosításTípus.Bináris:
            ofdBetöltés.DefaultExt = "*.dat";
            ofdBetöltés.Filter = "Adat állomány (*.dat)|*.dat|" + "Minden állomány (*.*)|*.*";
            if (ofdBetöltés.ShowDialog() == DialogResult.OK)
            {
                Stream st = File.OpenRead(ofdBetöltés.FileName);
                BinaryFormatter bf = new BinaryFormatter();
            }
        }
    }

```

```
        alBetöltöttAdatok = (ArrayList)bf.Deserialize(st);
        st.Close();
    }
    break;
    case SorosításTípus.SOAP_XML:
        ofdBetöltés.DefaultExt = "*.xml";
        ofdBetöltés.Filter = "XML állomány (*.xml)|*.xml|" +
            "Minden állomány (*.*)|*.*";
        if (ofdBetöltés.ShowDialog() == DialogResult.OK)
        {
            Stream st = File.OpenRead(ofdBetöltés.FileName);
            SoapFormatter sf = new SoapFormatter();
            alBetöltöttAdatok = (ArrayList)sf.Deserialize(st);
            st.Close();
        }
        break;
    case SorosításTípus.XML:
        ofdBetöltés.DefaultExt = "*.xml";
        ofdBetöltés.Filter = "XML állomány (*.xml)|*.xml|" +
            "Minden állomány (*.*)|*.*";
        if (ofdBetöltés.ShowDialog() == DialogResult.OK)
        {
            Stream st = File.OpenRead(ofdBetöltés.FileName);
            XmlSerializer xs = new XmlSerializer(typeof(Szemely[]));
            Szemely[] sz = new Szemely[1];
            sz = (Szemely[])xs.Deserialize(st);
            for (int i = 0; i < sz.Length; i++)
                alBetöltöttAdatok.Add(sz[i]);
            st.Close();
        }
        break;
    }
    Feltölt(lbBetöltött, alBetöltöttAdatok);
}
```

## 2. Windows Forms

### 2.1. Gyümölcsárazó automata



70. ábra.

Készítsünk egy gyümölcsárazó automatát szimuláló alkalmazást. A program négy gyümölcsöt tud kezelni. A bal oldali oszlopban csak olvasható szövegmezőkben jelennek meg az egységárak. A jobb oldali nyomógombokon történő kattintás idézi elő az ár kiszámítását. A „Tömeg” felirat melletti szövegmezőbe kell beírnia a felhasználónak a mennyiséget. Az „Ár” felirat melletti szövegmezőben jelenik meg a fizetendő összeg.

A Beállít gombon történő kattintás hatására egy jelszóbekérő ablak jelenik meg. A „bla-bla” jelszó megadása után az egységárak szerkesztőmezői írhatóvá válnak. Az új értékek megadása után az Alkalmaz gombon történő kattintással rögzíthetjük az adatokat. Ennek hatására az egységár mezők újra csak olvashatóak lesznek.

#### 2.1.1. A feladat megoldása

A feladat megoldásának fontosabb lépései a következők.

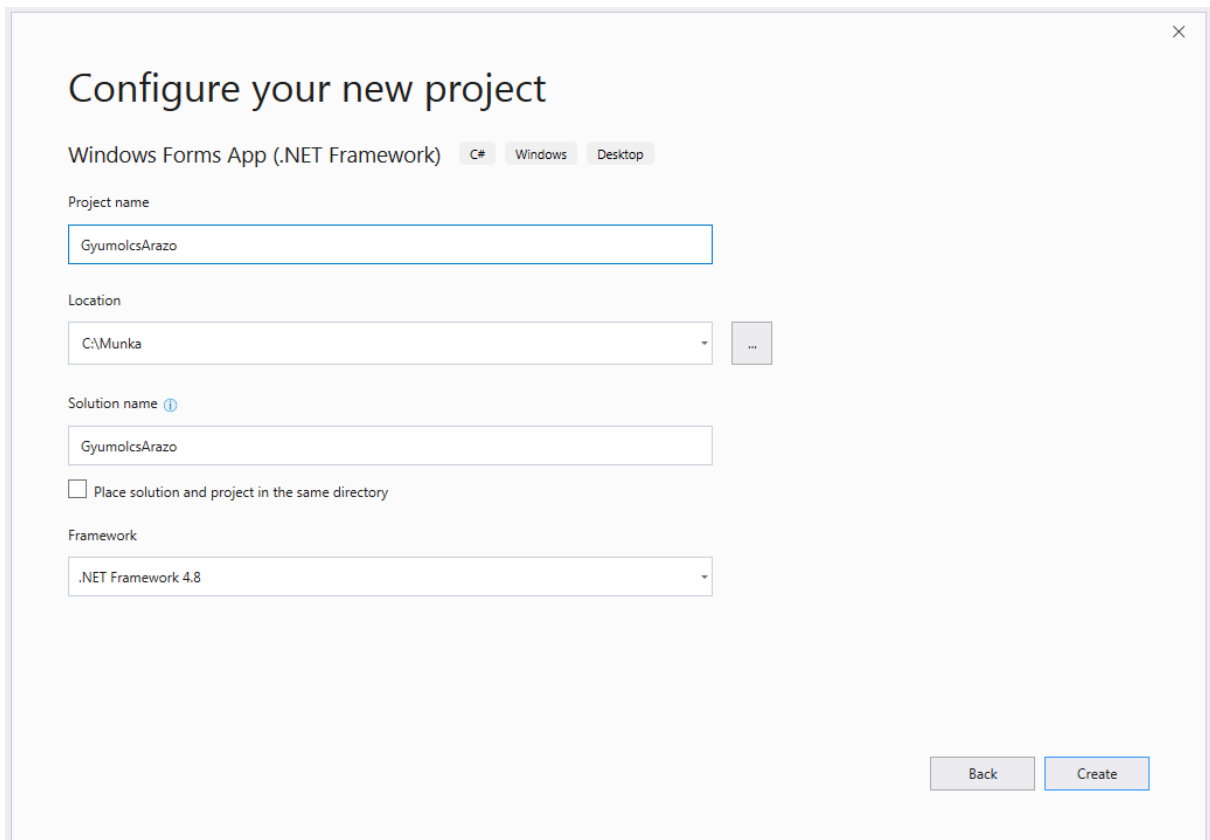
- A főablak grafikus felületének létrehozása.
- Eseménykezelő a Kilép nyomógombhoz.
- Asszociatív tömb a szerkesztőmező nyomógomb párosokhoz.
- Közös eseménykezelő a négy gyümölcs nyomógombhoz.
- Egységárak megváltoztatási lehetőségének megvalósítása.

A teljes osztálydiagram a megoldás végén található.

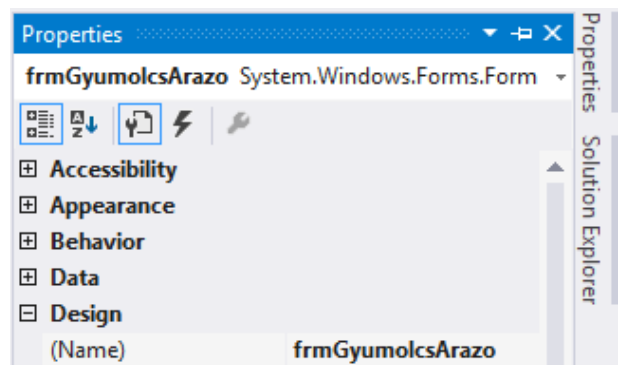


### 2.1.2. A főablak grafikus felületének létrehozása

Hozunk létre egy új projektet Gyumolcsarazo néven.



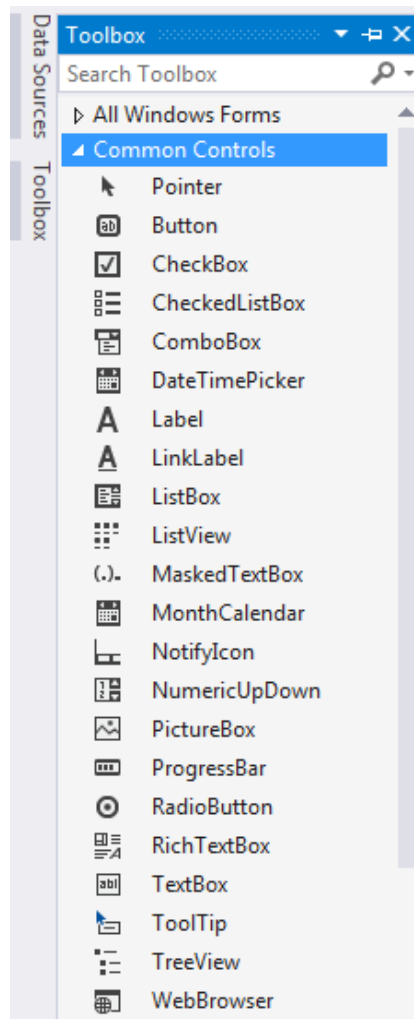
71. ábra.



72. ábra.

Az ablak objektumot nevezzük át **frmGyümölcsÁrazó**-nak, majd helyezzük el fejlécében a fenti képen látható szöveget. Ehhez tervezési nézetben (Design) válasszuk ki az ablakot (Form), majd nyissuk meg a Properties ablakot, ott válasszuk ki a (Name) tulajdonságot, és az eredeti Form1 értéket írjuk át frmGyümölcsÁrazó-ra. A feliratot az előzőhöz hasonló módon a Text tulajdonság segítségével módosíthatjuk.

A Solution Explorerben az ablak állományát Form1.cs-ről nevezzük át frmGyumolcsArazo.cs-rek (jobb egérgomb, Rename)



73. ábra.

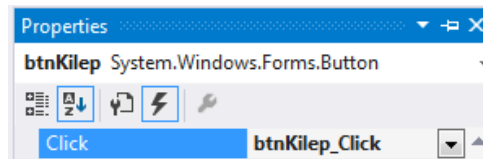
Következő lépésként helyezük el a komponenseket a formon (ablakon). Ehhez nyissuk meg a Toolbox palettát (a főablak keretében bal oldalon jelenik meg), majd kattintsunk a rajzszög gombon. Ebben a gyakorlatban olyan komponenseket használunk, amelyek a Common Controls csoportban vannak. Más esetekben egyszerűbb, ha az All Windows Forms csoportot nyitjuk meg, mert itt minden komponenst megtalálunk, neveik szerint sorba rendezve. A három használt komponens a nyomógomb (Button), szövegmező (TextBox) és a címke (Label). Ezeket az egér segítségével foghatjuk meg és húzhatjuk az ablak területére. Általános elnevezési konvenció lesz a továbbiakban az, hogy egy ablakra helyezett komponens nevét két részből állítjuk össze: az első rész kisbetűs és a típust tükrözi, a második rész nagy betűvel kezdődik és a típuson belüli egyedi azonosításra szolgál. Tekintsük át az elnevezést típusonként.

A nyomógombok neveit úgy alakítsuk ki, hogy bt-vel kezdődjenek, majd a név további része legyen azonos a felirattal (pl. btAlma).

A szerkesztőmezők neve kezdődjön tb-vel, majd ezt követően legyen azonos a jobb oldalukon levő nyomógomb feliratával az első négy esetben (pl. tbAlma), illetve a bal oldalukon látható felirattal az utolsó két esetben (pl. tbTomeg).

Az ötödik kivételével mindegyik szövegmező legyen csak olvasható (ReadOnly=True). Az Alkalmaz nyomógomb legyen letiltva (Enabled=False).

### 2.1.3. Eseménykezelő a Kilép nyomógombhoz



74. ábra.

Készítsünk egy eseménykezelőt a Kilép nyomógombhoz. PI. tervezési nézetben duplán kattintunk a nyomógombon. Alternatív lehetőség az, hogy tervezési nézetben kijelöljük a nyomógombot, majd a Properties ablakban kiválasztjuk a villámra emlékeztető Events nyomógombot, és a megjelenő eseménylistában kiválasztjuk a Click eseményt. Ezután duplán kattintunk a mellette levő üres mezőben.

```

/// <summary>
/// Kilépés a programból
/// </summary>
/// <param name="sender">Az eseményt előidéző nyomógomb</param>
/// <param name="e">Esemény paraméter.</param>
private void btnKilep_Click(object sender, EventArgs e)
{
    Application.Exit();
}

```

### 2.1.4. Asszociatív tömb a szerkesztőmező nyomógomb párosokhoz

A későbbi kényelmesebb programozás érdekében az összetartozó szerkesztőmező és nyomógomb referenciákat egy asszociatív tömbben szeretnénk tárolni.

Hozzunk létre egy olyan osztályt, ami asszociatív tömböt valósít meg. Az osztály neve legyen Párosok és származzon a DictionaryBase-ből. A Project menüben válasszuk ki az Add Class pontot, a sablonok közül az elsőt (Class) jelöljük be, majd állománynévként adjuk meg a Párosok.cs-t. A kódszerkesztőben az alábbi mintának megfelelően egészítsük ki az osztály kódját.

Az osztály létrehozásának alternatív módja az, hogy a Solution Explorerben kiválasztjuk a projektet, majd kattintunk a View Class Diagram gombon. Az ekkor megjelenő osztálydiagramban egy üres területen kattintva jobb egérgombbal a gyorsmenüben Add/Add class-t választunk.

```

/// <summary>
/// Asszociatív tömb megvalósítását támogató osztály
/// </summary>
public class Parosok: DictionaryBase
{
    /// <summary>
    /// Indexelő: feladata megadható/lekérdezhető, hogy
    /// egy adott nyomógombhoz melyik szerkesztőmező tartozik.
    /// </summary>
    /// <param name="kulcs">A nyomógomb neve</param>
    /// <returns>A szerkesztőmező neve</returns>
    public TextBox this[Button kulcs]
    {

```

```

    get
    {
        return ((TextBox)Dictionary[kulcs]);
    }
    set
    {
        Dictionary[kulcs] = value;
    }
}
}

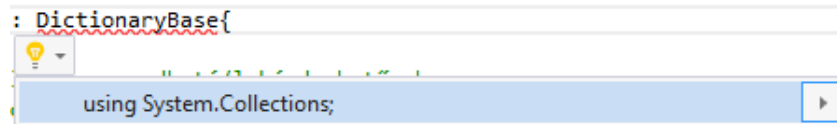
```

Fordítsuk le az alkalmazást (Ctrl+Shift+B).

Három hibaüzenetet kapunk.

Az első az alábbi:

The type or namespace name 'DictionaryBase' could not be found (are you missing a using directive or an assembly reference?)



75. ábra.

A hiba oka a DictionaryBase neve előtt a névtér elérési út megadásának elmaradása. A kódszerkesztőben a DictionaryBase kék hullámos vonallal van aláhúzva. Kattintsunk jobb egérgombbal rajta, majd a gyorsmenüben válasszuk a Resolve pontot, majd a using ...-t. Ennek eredményeképp az állomány elejére bekerül a megfelelő using direktíva.

A másik két hibát is hasonló módon kell megoldani. Mindkettő a `using System.Windows.Forms;` direktívát hiányolja.

Hozunk létre egy Párosok típusú adattagot Szótár néven az ablak osztályában (frmGyümölcsÁrazó) az osztálydiagram és a Class Details segítségével.

```
private Párosok Szótár;
```

Az ablakosztály konstruktorában adjunk kezdőértéket az egységáraknak, azaz határozzuk meg a négy szövegmező kezdeti tartalmát. Hozunk létre egy objektumot a Szótár adattaghoz, majd helyezzük el benne az összetartozó nyomógomb és szövegmező párosokat.

```

public frmGyumolcsArazo()
{
    InitializeComponent();
    tbAlma.Text = "150";
    tbBanan.Text = "256";
    tbKorte.Text = "190";
    tbNarancs.Text = "199";
    Szotar = new Párosok();
    Szotar[btnAlma] = tbAlma;
    Szotar[btnBanan] = tbBanan;
    Szotar[btnKorte] = tbKorte;
    Szotar[btnNarancs] = tbNarancs;
}

```

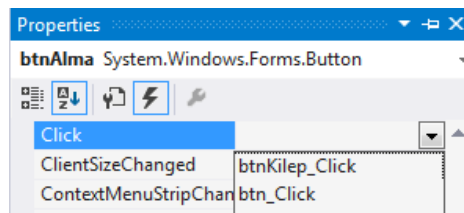
### 2.1.5. Közös eseménykezelő a négy gyümölcs nyomógombhoz

Készítsünk egy közös eseménykezelőt a négy gyümölcs nyomógombhoz. Az alábbi kódrészletet közvetlenül be kell gépelni az ablak osztályába.

```

/// <summary>
/// A négy gyümölcsnyomógomb közös eseménykezelője. Kiszámolja a
/// fizetendő árat, és elhelyezi azt a megfelelő szövegmezőbe
/// </summary>
/// <param name="sender">Az eseményt előidéző nyomógomb</param>
/// <param name="e">Esemény paraméter.</param>
private void btn_Click(object sender, EventArgs e)
{
    //Meghatározzuk a gyümölcsnek megfelelő szövegmezőt
    TextBox tbMunka = Szotar[(Button)sender];
    try
    {
        //Az aktuális egységár
        double egységAr = double.Parse(tbMunka.Text);
        //Az aktuális tömeg
        double tomeg = double.Parse(tbTomeg.Text);
        if (tomeg <= 0)
            throw new ApplicationException("A tömeg értéke negatív vagy nulla!");
        //A fizetendő ár
        double ar = egységAr * tomeg;
        //Ár kiírása
        tbAr.Text = ar.ToString();
    }
    catch(Exception ex)
    {
        MessageBox.Show("Hibás adatok!" + ex.Message, "Hiba", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}

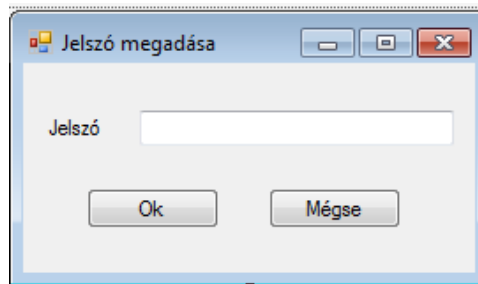
```



76. ábra.

Állítsuk be a négy nyomógombhoz a `btn_Click` függvényt eseménykezelőként. Ehhez tervezési nézetben jelöljük ki a négy nyomógombot egyszerre a Shift billentyű segítségével, majd a Properties ablakban kattintsunk az Events ikonra. A listában válasszuk ki a Click eseményt, és a mellette levő üres mezőben kattintsunk. A legördülő listából válasszuk aki a `btn_Click` elemet. Fordítsuk le, és próbáljuk ki a programot.

## 2.1.6. Egységárok megváltoztatása



77. ábra.

Az egységárok megváltoztatását jelszóhoz kívánjuk kötni, ezért hozunk létre egy új ablakot (Form) a jelszó bevitelhez. Válasszuk a Project menü Add Windows Form... menüpontját, majd a sablonok közül a Windows Form-ot. Az állomány neve legyen frmJelszoBekero. Kezdetben az új ablak osztálya is ugyanezt a nevet fogja viselni. A Properties ablak segítségével nevezzük át ékezetesre.

Az ablakot a mellékelt ábrának megfelelően alakítsuk ki. A nyomógomboknál és a szövegmezőnél a már megismert elnevezési konvenciót alkalmazzuk. A szövegmező esetében a PasswordChar=\* beállítás szükséges annak érdekében, hogy a jelszó begépelésekor csak csillagok jelenjenek meg. A btOK nyomógomb esetében a DialogResult=OK beállítás hatására az OK gombon történő kattintásra bezárul az ablak, és az ablak megjelenítését megvalósító ShowDialog metódus DialogResult.OK értékkel tér vissza. A btMegse nyomógomb esetében a DialogResult=Cancel beállítás hatására a Mégse gombon történő kattintásra bezárul az ablak, és az ablak megjelenítését megvalósító ShowDialog metódus DialogResult.Cancel értékkel tér vissza. Hozunk létre egy csak olvasható tulajdonságot az frmJelszoBekero osztályban Jelszo néven, aminek célja szövegmezőben megadott adatok lekérdezése.

```

/// <summary>
/// Csak olvasható tulajdonság a jelszóbekérő szövegmező
/// tartalmának lekérdezésére.
/// </summary>
public string Jelszo
{
    get
    {
        return tbJelszo.Text;
    }
}

```

Hozunk létre egy eseménykezelőt a főablak Beállít nyomógombjához, ami létrehozza és megjeleníti a jelszóbekérő ablakot, és érvényes jelszó (bla-bla) megadása esetén írhatóvá teszi az egységár szövegmezőket, majd engedélyezi az Alkalmaz nyomógombot és letiltja a Beállít nyomógombot.

```

/// <summary>
/// Bekéri a jelszót, és egyezés esetén írhatóvá teszi az egységár /// szövegmezőket.
/// </summary>
/// <param name="sender">Az eseményt előidéző nyomógomb</param>
/// <param name="e">Esemény paraméter</param>
private void btnBeallit_Click(object sender, EventArgs e)
{
    //Jelszóbekérő ablak objektum létrehozása
}

```

```

frmJelszoBekero jSz = new frmJelszoBekero();
//Jelszobekero ablak megjelenítése
DialogResult dR = jSz.ShowDialog();
//Ha az Ok nyomógommbal zárta be a felhasználó az ablakot
if(dR == DialogResult.OK)
{
    //jelszó ellenőrzése
    if(jSz.Jelszo == "bla-bla")
    {
        //A szövegmezők írhatóvá tétele
        tbAlma.ReadOnly = false;
        tbKorte.ReadOnly = false;
        tbNarancs.ReadOnly = false;
        tbBanan.ReadOnly = false;
        //Beállít nyomógomb letiltása
        btnBeallit.Enabled = false;
        //Alkalmaz nyomógomb engedélyezése
        btnAlkalmaz.Enabled = true;
    }
    else
    {
        MessageBox.Show("Hibás jelszó!", "Hiba", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
}

```

Készítsünk egy eseménykezelőt az Alkalmaz nyomógommbhoz, ami ellenőrzi a megadott egységárakat, és ha minden rendben, akkor újból csak olvashatóvá teszi a megfelelő szövegmezőket, engedélyezi a Beállít nyomógombot, majd letiltja az Alkalmaz nyomógombot.

```

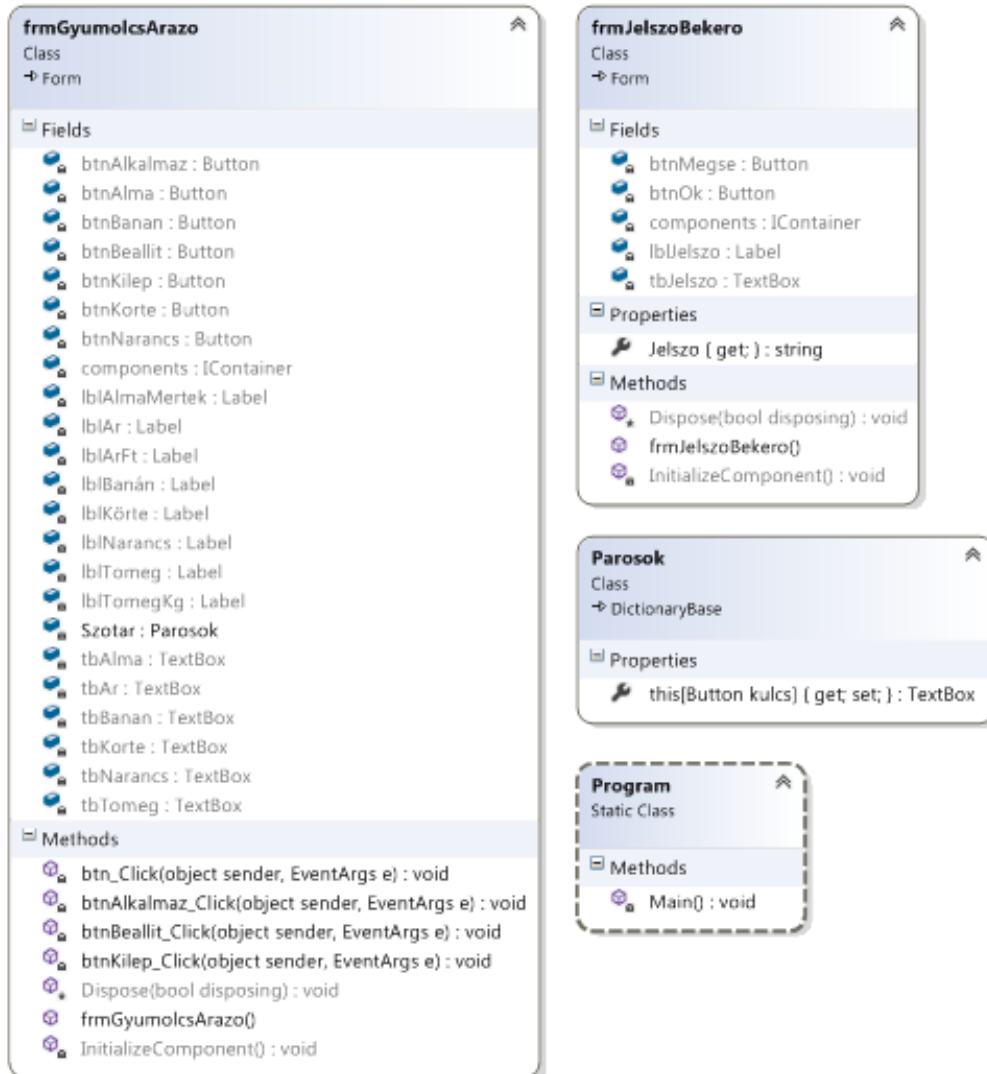
/// <summary>
/// Leellenőrzi a megadott egységárakat, és ha rendben
/// vannak, akkor újból csak olvashatóvá teszi az
/// egységár szövegmezőket.
/// </summary>
/// <param name="sender">Az eseményt előidéző nyomógomb</param>
/// <param name="e">Esemény paraméter</param>
private void btnAlkalmaz_Click(object sender, EventArgs e)
{
    try
    {
        double egysegAr = double.Parse(tbAlma.Text);
        if (egysegAr <= 0)
            throw new Exception("Az alma ára negatív");
        tbAlma.ReadOnly = true;
        egysegAr = double.Parse(tbKorte.Text);
        if (egysegAr <= 0)
            throw new Exception("A körte ára negatív");
        tbKorte.ReadOnly = true;
        egysegAr = double.Parse(tbNarancs.Text);
        if (egysegAr <= 0)
            throw new Exception("A narancs ára negatív");
        tbNarancs.ReadOnly = true;
        egysegAr = double.Parse(tbBanan.Text);
        if (egysegAr <= 0)
            throw new Exception("A banán ára negatív");
        tbBanan.ReadOnly = true;
    }
}

```

```

//Beállít nyomógomb engedélyezése
btnBeallit.Enabled = true;
//Alkalmaz nyomógomb letiltása
btnAlkalmaz.Enabled = false;
}
catch(Exception ex)
{
    MessageBox.Show("Hibás adatok!" + ex.Message, "Hiba", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

```

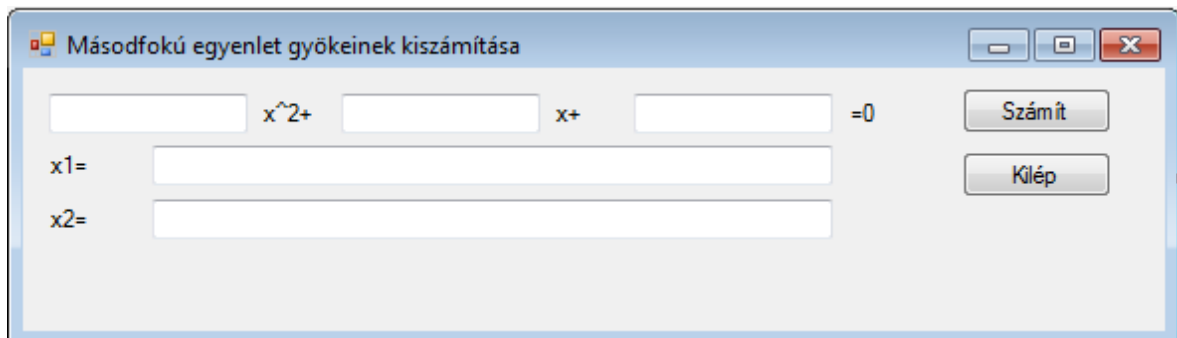


78. ábra.

### 2.1.7. Házi feladat

Készítsünk egy grafikus felületű alkalmazást a másodfokú egyenlet gyökeinek kiszámítására. Ha csak egy gyök van, akkor az  $x_2=$  és a mellette levő szövegmező ne legyen látható. Ha a gyökök nem valósak, akkor  $a+b*i$  formában jelenjenek meg.





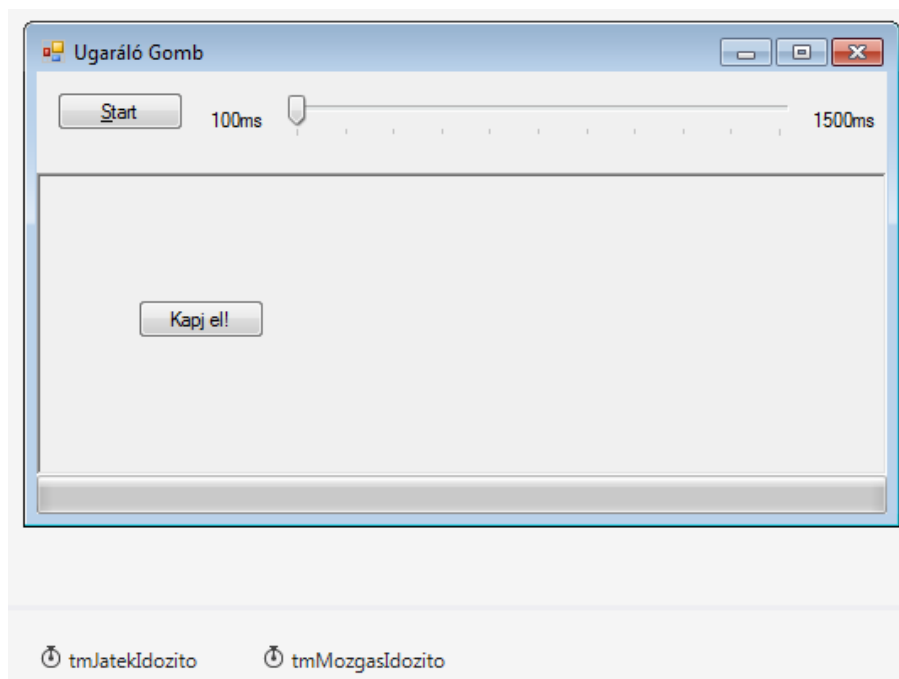
79. ábra.

## 2.2. Ugráló Gomb

Készítsünk egy egyszerű játékprogramot, ami egy mozgó nyomógombot tartalmaz. A nyomógomb beállított ideig marad egy helyben, majd az ablakon számára elhatárolt terület (panel) egy véletlenszerűen kiválasztott pozíciójában jelenik meg újra. A játék során az a feladat, hogy minél többször kattintsunk a gombon az egér segítségével. A játék előre beállított ideig tart, a form alján elhelyezett végrehajtásjelző tájékoztat az eltelt időről.

A feladat egy lehetséges megoldása a következő:

### 2.2.1 Felület kialakítása



80. ábra.

- A főablak neve legyen **Name=frmUgraloGomb**, az őt definiáló állomány nevét is változtassuk meg **frmUgraloGomb.cs**-re. Solution Explorerben jobb egérgomb a Form1.cs-n, majd a név átírása.
- Egy nyomógombot (Button) helyezünk a formra a bal felső sarokba. **Name=btnStart**, **Text=&Start**. Ezzel lehet majd elindítani a játékot.

- Egy csúszkát (TrackBar) helyezünk el a nyomógomb jobb oldalára. **Name=trbCsuszka**. Ezzel lehet majd beállítani, hogy mennyi ideig maradjon egy helyben a mozgó nyomógomb.
- Egy-egy címkét (Label) helyezünk el a csúszka bal és jobb oldalára. Feladatuk a csúszkával beállítható minimális és maximális időérték kijelzése. Nevük: **Name=lblMin** és **Name=lblMax**.
- Egy végrehajtásjelzőt (ProgressBar) helyezünk el a form aljába. Ez fog tájékoztatni az eltelt játékidőről. Tulajdonságai: **Name=prbVegrehajtasJelzo**, **Dock=Bottom**.
- Egy panelt helyezünk el a nyomógomb alatti ablakterületre, ez lesz a játéktér, ezen belül jelenhet meg a mozgó nyomógomb. Tulajdonságai: **Name=pLap**, **BorderStyle=Fixed3D**, **Dock=Bottom**.
- Egy nyomógombot (Button) helyezünk a panelre, ezen kell kattintani játék közben. Tulajdonságai: **Name=btnKapjEl**, **Text=Kapj el!**
- Két időzítő (Timer) komponenst helyezünk a panelre. Az első feladata az lesz, hogy másodpercenként Tick eseményt generálva lehetővé teszi az eltelt játékidő követését és a végrehajtásjelző léptetését. Tulajdonsága: **Name= tmJatekIdozito**. A második feladata az lesz, hogy jelezze, hogy mikor telt le az az idő, amíg egy helyben maradhat a „Kapj El!” nyomógomb. Tulajdonsága: **Name= tmMozgasIdozito**.

### 2.2.2. Adattagok definiálása

Hozzunk létre az ablak osztályában egy adattagot az elért pontszám tárolására az osztálydiagram segítségével.

```
/// <summary>
/// Az elért pontszám.
/// </summary>
private int eredmeny;
```

Hozzunk létre az ablak osztályában egy adattagot véletlenszámok előállítására szolgáló objektum referenciájának tárolására.

```
/// <summary>
/// Véletlenszámok előállítására szolgáló objektum.
/// </summary>
private Random veletlen;
```

### 2.2.3. Kezdőérték adás a konstruktorban.

Az alábbi utasításokkal beállítjuk az ablakon elhelyezett komponensek tulajdonságait.

```
public frmUgraloGomb()
{
    InitializeComponent();
    //A vezérlők kezdeti beállítása
    eredmeny = 0;
    //Az alsó és felső határérték arra, hogy mennyi ideig maradhat
    //egy helyben a mozgó nyomógomb.
    trbCsuszka.Minimum = 100;
    trbCsuszka.Maximum = 1500;
    //A csúszka jelzővonalainak távolsága
    trbCsuszka.TickFrequency = 200;
    //Mekkora elmozdulást jelent a csúszkán a le/fel nyíl billentyű
    // lenyomása
```

```

trbCsuszka.SmallChange = 100;
// Mekkora elmozdulást jelent a csúszkán a Page Up / Page Down
// nyíl billentyű lenyomása.
trbCsuszka.LargeChange = 500;
//A csúszka kezdeti pozíciója
trbCsuszka.Value = 500;
//A bal és jobb oldali címkék(feliratok) szövege.
lblMin.Text = trbCsuszka.Minimum.ToString() + " ms";
lblMax.Text = trbCsuszka.Maximum.ToString() + " ms";
//Mozgó gomb kezdetben letiltva
btnKapjEl.Enabled = false;
//Véletlenszámokat előállító objektum létrehozása.
veletlen = new Random();
//Mindkét időzítő kezdetben leállítva.
tmJatekIdozito.Enabled = false;
tmMozgasIdozito.Enabled = false;
//Végrehajtásjelző szélsőértékeihez társított számértékek.
prbVegrehajtasJelzo.Maximum = 10;//a játékidő s-ban
prbVegrehajtasJelzo.Minimum = 0;
//Végrehajtásjelző kezdőértéke
prbVegrehajtasJelzo.Value = 0;
//Mekkora egy lépés a végrehajtásjelzőn.
prbVegrehajtasJelzo.Step = 1;
}

```

#### 2.2.4. Az ablak fejlécében feliratot megjelenítő metódus definiálása

Az ablak fejlécében ki akarjuk jelezni, hogy mennyi az eddig elért találatok száma, milyen időközönként mozdul el a nyomógomb, és mennyi idő van még hátra a játékból.



81. ábra.

```

/// <summary>
/// Friss információt jelenít meg a játék állásáról az ablak
/// fejlécében.
/// </summary>
private void FeliratKiir()
{
    Text = string.Format("Találatok: {0} Időzítés: {1} ms Még hátravan: {2} s",
        eredmény, trbCsuszka.Value, prbVegrehajtasJelzo.Maximum -
        prbVegrehajtasJelzo.Value);
}

```

#### 2.2.5. A játékot elindító Start gomb eseménykezelőjének elkészítése.

A Start gombot kijelöljük, Properties ablak, Events gomb, Click esemény, dupla kattintás.

```

/// <summary>
/// A Start gombon történő kattintásra reagáló eseménykezelő.
/// Kinullázza az eredményt tároló változót és a végrehajtásjelzőt.
/// A mozgásidőzítőt a csúszka állapotához igazítja és indítja.
/// Engedélyezi a játékidő mérés időzítőjét.
/// Engedélyezi a KapjEl gombot, tiltja a Start gombot.
/// Kezdeti feliratot jelenít meg az ablak fejlécében.

```

```
/// </summary>
private void btnStart_Click(object sender, EventArgs e)
{
    eredmeny = 0;
    prbVegrehajtasJelzo.Value = 0;
    tmMozgasIdozito.Interval = trbCsuszka.Value;
    btnStart.Enabled = false;
    tmMozgasIdozito.Enabled = true;
    tmJatekIdozito.Enabled = true;
    FeliratKiir();
    btnKapjEl.Enabled = true;
}
}
```

### 2.2.6. Eseménykezelő készítése a csúszka mozgatásához

Kijelöljük a csúszkát, Properties ablak, Events gomb, Scroll esemény, dupla kattintás.

```
/// <summary>
/// Eseménykezelő: a felhasználó elmozdította a csúszkát.
/// Leállítjuk a mozgásidőzítőt. A csúszka értékének megfelelően
/// beállítjuk a mozgásidőzítés idejét. Ha mindez játékidőben
/// történt, akkor engedélyezzük a mozgásidőzítőt. Frissítjük a
/// feliratot az ablak fejlécében.
/// </summary>
private void trbCsuszka_Scroll(object sender, EventArgs e)
{
    bool vanJatek = tmMozgasIdozito.Enabled;
    tmMozgasIdozito.Enabled = false;
    tmMozgasIdozito.Interval = trbCsuszka.Value;
    if (vanJatek)
        tmMozgasIdozito.Enabled = true;
    FeliratKiir();
}
}
```

### 2.2.7. Eseménykezelő készítése a Kapj El! gombhoz.

Kijelöljük a gombot, Properties ablak, Events gomb, Click esemény, dupla kattintás.

```
/// <summary>
/// Eseménykezelő: a felhasználó kattintott a KapjEl gombon.
/// Növeli eggyel az eredményt és frissíti a feliratot az
/// ablak fejlécében.
/// </summary>
private void btnKapjEl_Click(object sender, EventArgs e)
{
    eredmeny++;
    FeliratKiir();
}
}
```

### 2.2.8. Eseménykezelő készítése a mozgásidő Tick eseményéhez

Kijelöljük a játékidő időzítőt, Properties ablak, Events gomb, Tick esemény, dupla kattintás.

```
/// <summary>
/// Eseménykezelő: lejárt a mozgásidőzítő ideje. A KapjEl gombot
/// egy véletlenszerűen kiválasztott új pozícióba helyezi át.
/// </summary>
private void tmMozgasIdozito_Tick(object sender, EventArgs e)
{
    btnKapjEl.Left = veletlen.Next(pLap.Width - btnKapjEl.Width);
}
```

```

    btnKapjEl.Top = veletlen.Next(pLap.Height - btnKapjEl.Height);
}

```

### 2.2.9. Eseménykezelő készítése a játékidő időzítőhöz

Kijelöljük a játékidő időzítőt, Properties ablak, Events gomb, Tick esemény, dupla kattintás.

```

/// <summary>
/// Eseménykezelő: lejárt a játékidőidőzítő időegysége. Lépteti a
/// végrehajtásjelzót.
/// Frissíti a feliratot az ablak fejlécében. Ha lejárt a játékidő,
/// akkor leállítja a két időzítőt, letiltja a KapjEl gombot és
/// engedélyezi a Start gombot.
private void tmJatekIdozito_Tick(object sender, EventArgs e)
{
    prbVegrehajtasJelzo.PerformStep();
    FeliratKiir();
    if(prbVegrehajtasJelzo.Value == prbVegrehajtasJelzo.Maximum){
        tmMozgasIdozito.Enabled = false;
        tmJatekIdozito.Enabled = false;
        btnStart.Enabled = true;
        btnKapjEl.Enabled = false;
    }
}

```

### 2.2.10. Érvénytelen találatok kiszűrése

Csak az egérrel elért találatokat tekintjük érvényesnek, ezért meg szeretnénk akadályozni, hogy a játékos a Return gomb megnyomásával is pontot szerezzen. Ehhez először létrehozunk egy bool típusú adattagot az ablak osztályában ervenyes néven.

```

/// <summary>
/// Meghatározza, hogy találatot jelent-e a Click esemény.
/// </summary>
private bool ervenyes;

```

Csak akkor érvényes a találat, ha a Kapj el gomb Click eseményénél az egér a nyomógomb felett tartózkodott. Ezért készítünk egy eseménykezelőt, ami igazra állítja az ervenyes adattagot, ha az egérkurzor belép a nyomógomb területére (Kapj el gomb MouseEnter eseménye).

```

/// <summary>
/// Igazra állítja az Érvényes adattagot, amikor az egérkurzor belép
/// a nyomógomb területére.
/// </summary>
private void btnKapjEl_MouseEnter(object sender, EventArgs e)
{
    ervenyes = true;
}

```

Ezután készítünk egy eseménykezelőt, ami hamisra állítja az ervenyes adattagot, ha az egérkurzor kilép a nyomógomb területéről (Kapj el gomb MouseLeave eseménye).

```

/// <summary>
/// Hamisra állítja az Érvényes adattagot, amikor az egérkurzor
/// kilép a nyomógomb területéről.
/// </summary>
private void btnKapjEl_MouseLeave(object sender, EventArgs e)
{
    ervenyes = false;
}

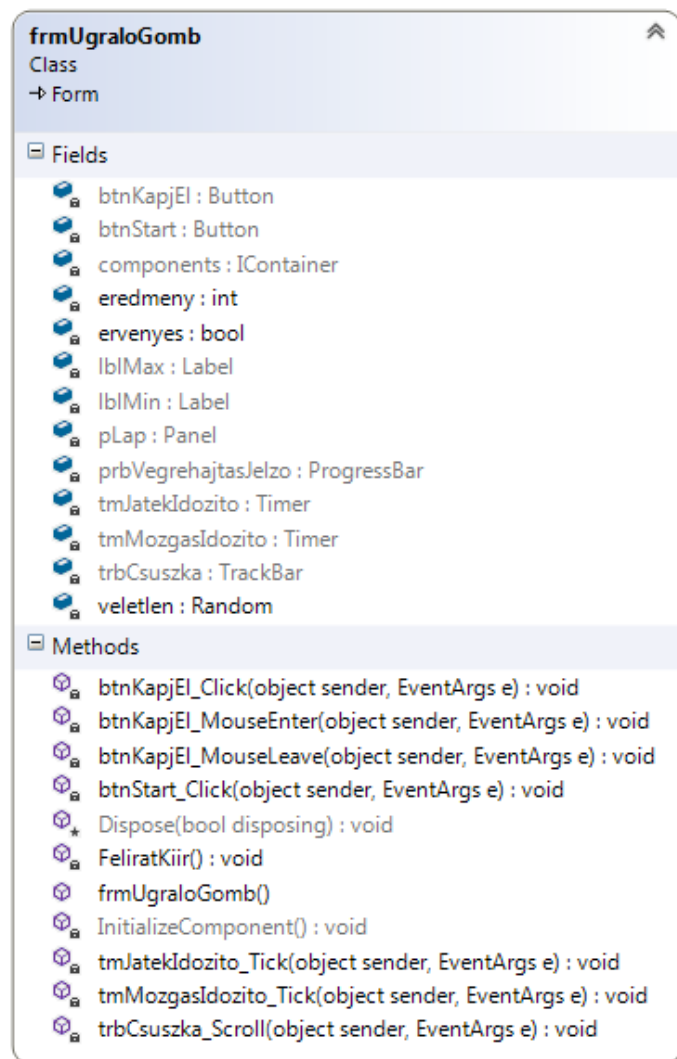
```

```
}
```

Végül úgy alakítjuk át a nyomógomb Click eseménykezelőjét, hogy csak akkor számoljon, ha érvényes a találat.

```
/// <summary>  
/// Eseménykezelő: a felhasználó kattintott a KapjEl gombon.  
/// Növeli eggyel az eredményt és frissíti a feliratot az  
/// ablak fejlécében. Csak akkor számol találatot, ha az  
/// ervenyes adattag értéke igaz.  
/// </summary>  
private void btnKapjEl_Click(object sender, EventArgs e)  
{  
    if (ervenyes)  
    {  
        eredmeny++;  
        FeliratKiir();  
    }  
}
```

A program osztálydiagramja a következőképp néz ki:

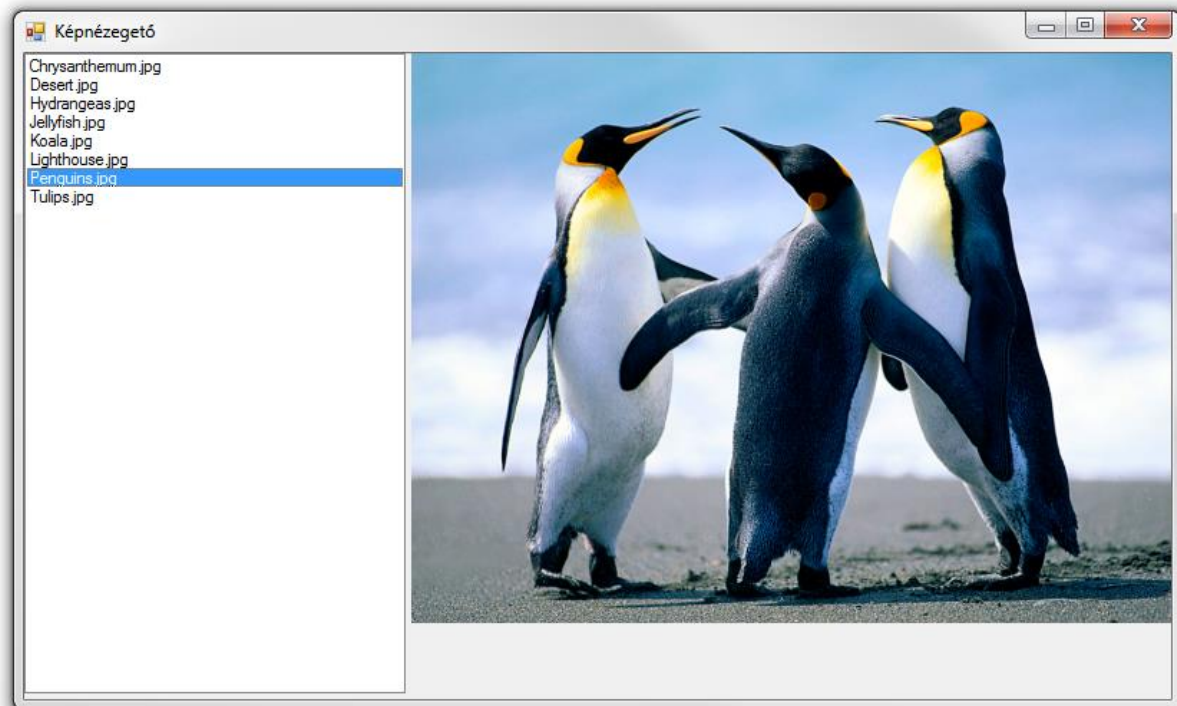


82. ábra.

### **2.2.11. Házi Feladat**

Tegyük lehetővé a felhasználó számára, hogy állítsa be a játékidő nagyságát. A játék közben ezen ne lehessen változtatni. Tehát a beállítás csak két játék közötti időben legyen lehetséges.

## 2.3. Képnézegető Program



83. ábra.

Készítsünk egy egyszerű képnézegető programot JPG képekhez. Az ablak függőlegesen két részre legyen osztva. Baloldalon egy listaablak jelenjen meg, amiben a fájlok nevei láthatók. A le-fel nyílbillentyűk segítségével ezek közül választhatjuk ki a jobb oldali részben megjeleníteni kívánt képet. Kezdetben a legelső kép legyen kiválasztva. A képek egy előre meghatározott könyvtárban kell legyenek: projekt könyvtára\bin\Debug\res. A jobb oldali részben úgy kell megjeleníteni a képeket, hogy a lehető legjobban kitöltsék a rendelkezésre álló helyet, de az eredeti szélesség/magasság arány ne torzuljon.

### 2.3.1. A feladat megoldása

A feladat megoldásának fontosabb lépései a következők:

- Grafikus felület létrehozása
- Képnevek beolvasása a program indulásakor
- Kiválasztott kép betöltése
- Kép megjelenítése
- Képmegjelenítés már a program indulásakor

### 2.3.2. Grafikus felület létrehozása

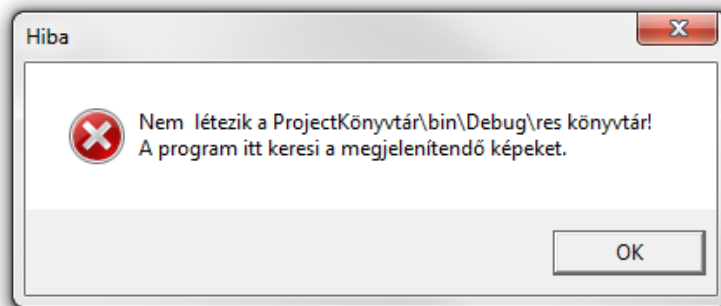
Hozunk létre egy Windows Forms Application projektet Kepnezegeto néven. Az ablak fejlécében helyezzük el a Képnézegető szöveget (**Text=Képnézegető**). Az ablakosztályt nevezzük át **frmKepnezegeto**-re, míg az őt tartalmazó állományt **frmKepnezegeto.cs**-re.



Helyezzünk el egy **SplitContainer** komponenst az ablakon, és nevezzük el **scElvalasztó**-nak. Ez vízszintes irányban két részre osztja az ablakot úgy, hogy futási időben az elválasztó vonal megfogható az egérrel, és a helyzete módosítható. A két részbe különböző komponensek helyezhetők.

Helyezzünk egy **ListBox** komponenst a bal oldali részbe úgy, hogy teljesen kitöltse azt. Tulajdonságok: **Name: lbFajlok**, **Dock=Fill**. Helyezzünk egy **PictureBox** komponenst a jobb oldali részbe úgy, hogy a bal felső sarka pontosan illeszkedjen a jobb oldali panel bal felső sarkához. Tulajdonságok: **Name: pbKep**, **Location=0;0**.

### 2.3.3. Képnevek beolvasása a program indulásakor



84. ábra.

A projekt könyvtárán belül a `Debug\bin` mappában hozzunk létre egy `res` nevű alkönyvtárat. Ebbe másoljuk le a szerveren található képeket.

Az ablakosztály konstruktorában hajtsuk végre a következőket. A képállományok neveivel töltsük fel a listaablakot. Ha nem létezik `res` könyvtár, idézzünk elő egy kivételt, aminek adjunk át egy hibaüzenetet. Ehhez alakítsuk ki az alábbiak szerint a konstruktort.

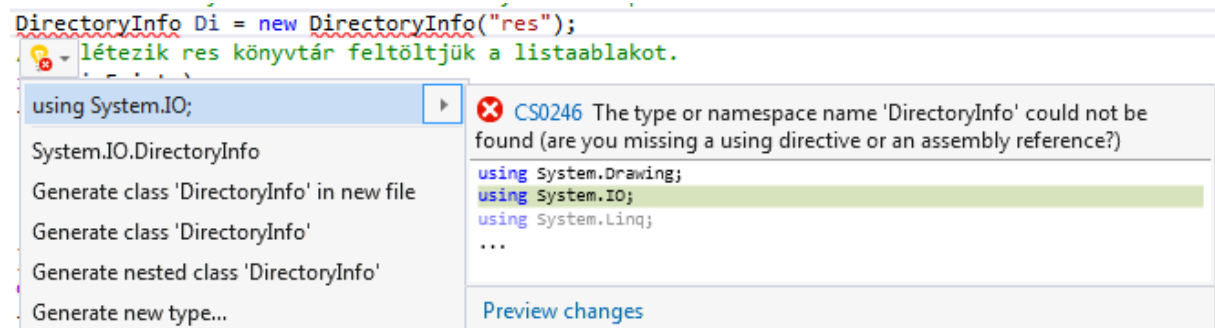
```

/// <summary>
/// Listaablak feltöltése.
/// Kép komponens eredeti méreteinek lekérdezése.
/// </summary>
public frmKepnezegeto()
{
    InitializeComponent();
    //Saját inicializálás
    //Az exe-t tartalmazó ../projekt/Debug/bin könyvtáron belül
    //van a res könyvtár. Ide lettek elhelyezve a képek.
    DirectoryInfo Di = new DirectoryInfo("res");
    //Ha létezik res könyvtár feltöltjük a listaablakot.
    if (Di.Exists){ //Lekérdezzük a jpg kiterjesztésű állományokat.
        FileInfo[] Fi = Di.GetFiles("*.jpg");
        //Minden állománynevet felvesszünk a listaablakba.
        foreach (FileInfo Fajl in Fi)
            lbFajlok.Items.Add(Fajl.Name);
    }
    else{ //Ha nem létezik res könyvtár hiabüzenet és kilépés
        throw new Exception("Nem létezik "+a ProjektKönyvtár\\bin\\Debug\\res
            könyvtár!\n" +
            "A program itt keresi a megjelenítendő képeket.");
    }
}
}

```

Nyissuk meg a Program.cs állományt a Solution Explorer segítségével. A Main metódusban helyezünk el kivételkezelő kódot az alábbiak szerint. Ez a kivételkezelő kód fog lefutni, ha a konstruktorban bekövetkezik a kivétel. Megjeleníti a hibaüzenetet, majd kilép a programból.

```
static void Main()
{
    try
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new frmKepnezegeto());
    }
    catch(Exception exc)
    {
        MessageBox.Show(exc.Message, "Hiba", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```



85. ábra.

Fordítsuk le a projektet.

Kapunk négy hibaüzenetet.

A System.IO névtérhivatkozás hiányzik. Kattintsunk az első hullámos piros vonallal aláhúzott osztálynévre bal egérgombbal, majd a megjelenő villanykörtére, végül a hiányzó névtérre.

Készítsük el a projekt osztálydiagramjait. Ehhez Solution Explorerben kijelöljük a projektet, majd kattintunk az osztálydiagram ikonon. Hozzunk létre az ablak osztályában egy adattagot, amit a lemeztől betöltött kép ideiglenes tárolására fogunk használni:

```
/// <summary>
/// Adattag a kép ideiglenes tárolásához.
/// </summary>
private Image Kep;
```

### 2.3.4. Kiválasztott kép betöltése

Készítsünk egy eseménykezelőt a listaablak elemkiválasztásához (SelectedIndexChanged), amelyben betöltjük a lemeztől a listaablakban kiválasztott kép állapotát, és tároljuk azt a Kép adattagban. Amennyiben a képmegjelenítő komponensben van kép korábbról, akkor felszabadítjuk az általa lefoglalt erőforrásokat, érvénytelenítjük a képmegjelenítő komponens aktív területét.

```
/// <summary>
/// Ez fut le, ha az állományneveket tartalmazó listaablakban
/// megváltozik a kijelölés. Betölti a kiválasztott képet.
/// </summary>
```

```
private void lbFajlok_SelectedIndexChanged(object sender, EventArgs e)
{
    //Kép betöltése és Image objektum létrehozása belőle.
    Kep = Image.FromFile(@"res\" + (string)lbFajlok.SelectedItem);
    //Ha a képmegjelenítő komponensben van eltárolt kép,
    //akkor ezt először törölni kell (fel kell szabadítani a
    //hozzá lefoglalt erőforrásokat).
    if (pbKep.Image != null)
    {
        pbKep.Image.Dispose();
        pbKep.Image = null;
    }
    pbKep.Refresh();
}
}
```

### 2.3.5. Kép megjelenítése

A kép megjelenítését a képmegjelenítő komponens Paint eseményéhez kívánjuk kapcsolni. Mikor keletkezik Paint esemény? Egy terület érvénytelenítése Paint eseményt idéz elő. Ez bekövetkezhet úgy, hogy a programozó idézi elő szándékosan egy utasítással, vagy ha a teljes komponens (ablak) vagy annak egy része érvénytelenné válik, pl. takarásból újra látható lesz, vagy átméretezi a felhasználó az ablakot, de ez az esemény az ablak első megjelenésekor is bekövetkezik.

Készítsünk egy eseménykezelőt a pbKép Paint eseményéhez. Ebben méretezzük át a képmegjelenítő komponensét úgy, hogy a Kép adattagban tárolt referenciájú kép eredeti arányainak megtartásával a lehető legjobban kihasználjuk a rendelkezésre álló helyet. Jelenítsük meg a képet.

```
/// <summary>
/// Ez fut le, ha újra kell festeni a képmegjelenítő komponens
/// tartalmát. Átméretezi a képmegjelenítő komponensét úgy,
/// hogy a kép arányainak elrontása nélkül a lehető legjobban
/// kihasználjuk a rendelkezésre álló helyet.
/// </summary>
private void pbKep_Paint(object sender, PaintEventArgs e)
{
    //ha van betöltött kép
    if (Kep != null){
        double nagyitas = Math.Min(
            (double)scElvalaszto.Panel2.Height / Kep.Height,
            (double)scElvalaszto.Panel2.Width / Kep.Width);
        pbKep.SizeMode = PictureBoxSizeMode.StretchImage;
        pbKep.Width = (int)(nagyitas * Kep.Width);
        pbKep.Height = (int)(nagyitas * Kep.Height);
        pbKep.Image = Kep;
    }
}
}
```

### 2.3.6. Képmegjelenítés már a program indulásakor

Készítsünk egy eseménykezelőt az ablak betöltődéséhez (Load esemény), amelyben kijelöljük a legelső listaelemet. Így már a program indulásakor látszik egy kép.

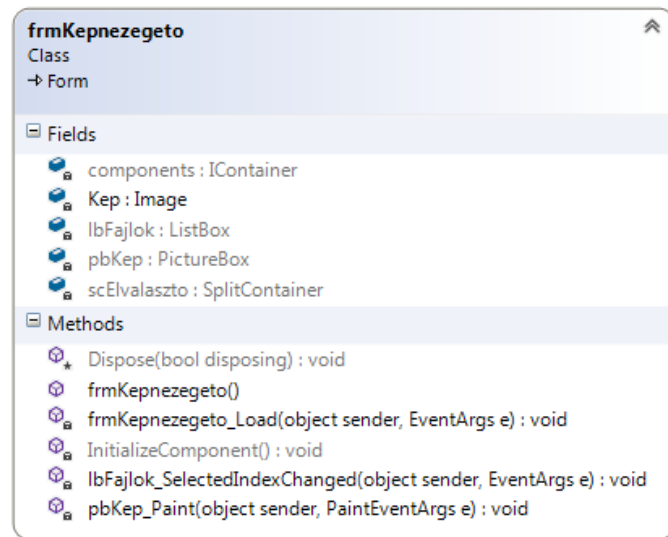
```
/// <summary>
/// A form betöltődésekor hajtódik végre még a rajzolási/kifestési
/// műveletek előtt. Kijelöli a listaablakban szereplő állományok /// közül
```

```

/// az elsőt, így a program indulásakor már látszik az első kép.
/// </summary>
private void frmKepnezegeto_Load(object sender, EventArgs e)
{
    // Ha vannak állománynevek a listaablakban, akkor kijelöljük az
    // elsőt, így a program indulásakor már látszik az első kép.
    if (lbFajlok.Items.Count > 0)
        lbFajlok.SelectedIndex = 0;
}

```

### 2.3.7. Osztálydiagram



86. ábra.

### 2.3.8. Házi feladat

Írjuk át a programot úgy, hogy egy fanézet (TreeView) komponenst is helyezzünk el az ablakban a mellékelt képnek megfelelően. A fanézetben lehessen kiválasztani a könyvtárat, majd a kiválasztott könyvtárban levő jpg állományok nevei jelenjenek meg a listaablakban. Ehhez felhasználható az ötödik előadáson bemutatott *KönyvtárFa* nevű mintaprogram.

A feladat megoldásának fontosabb lépései a következők:

- Kikapcsoljuk az lbFájlok panelkitöltését (Dock).
- Az lbFájlokat ideiglenesen áthelyezzük a jobb oldali panelre.
- Új SplitContainer komponenst helyezünk a bal oldali panelre, neve legyen scKiválasztó. Így az ablak összesen három részre osztott lesz.
- Az lbFájlokat áthelyezzük a középső panelre. Bekapcsoljuk a kitöltést (Dock).
- Egy fanézet (TreeView) komponenst helyezünk a bal oldali panelre, neve legyen tvKönyvtár.

### 2.4. Szöveg elhelyezése bittömbbe és kiolvasása program

Készítsünk grafikus felületű programot a korábban megismert SzBArray osztály (ld. 3. gyakorlat SzövegBittömbbe projekt) használatának kipróbálásához.

A program az alábbi funkcionalitást kell megvalósítsa. A program indítása után megadunk egy szöveget az „Eredeti szöveg” felirat melletti szövegmezőbe, majd kattintunk az Indít nyomógombon. Ennek hatására a „Kódolva” felirat melletti szövegmezőben megjelenik a szöveget jelképező 0-sok és 1-esek sorozata. Ezután kattintunk a Kiolvas nyomógombon, aminek hatására a „Kiolvas” felirat melletti szövegmezőben megjelenik az eredeti szöveg. A Kilépés gombra kattintva léphetünk ki a programból. Ha úgy kattintunk a Kódol vagy a Kiolvas gombra, hogy nem adtunk meg szöveget, akkor hibaüzenetet kapunk.

Típus	Tulajdonság=érték
Label	Text=Eredeti szöveg
Label	Text=Kódolva
Label	Text=Kiolvas
TextBox	Name=tbEredetiSzoveg
TextBox	Name=tbKodolva, ScrollBars=Vertical, MultiLine=True ReadOnly=True
TextBox	Name=tbKiolvas
Button	Text=Kódol, Name=btnKodol
Button	Text=Kiolvas, Name=btnKiolvas
Button	Text=Kilép, Name=btnKilep

### 2.4.1. A feladat megoldása

A feladat megoldásának fontosabb lépései a következők:

- Grafikus felület létrehozása
- Eseménykezelő a Kilép nyomógombhoz
- Az SzBArray osztály hozzáadása a projekthez és kisebb módosítása
- SzBArray típusú adatag létrehozása az ablak osztályában
- Kódolás megvalósítása
- Kiolvasás megvalósítása

### 2.4.2. Grafikus felület létrehozása

Első lépésként létrehozunk egy Windows Forms Application típusú új projektet SzBGraf néven. Ezután tervezési (Design) nézetben elhelyezzük az ablakon a szükséges vezérlőelemeket a táblázat szerint. Az ablak (Form1) fejlécébe (Text tulajdonság) helyezzük el a „Szöveg elhelyezése bittömbbe és kiolvasása” feliratot.

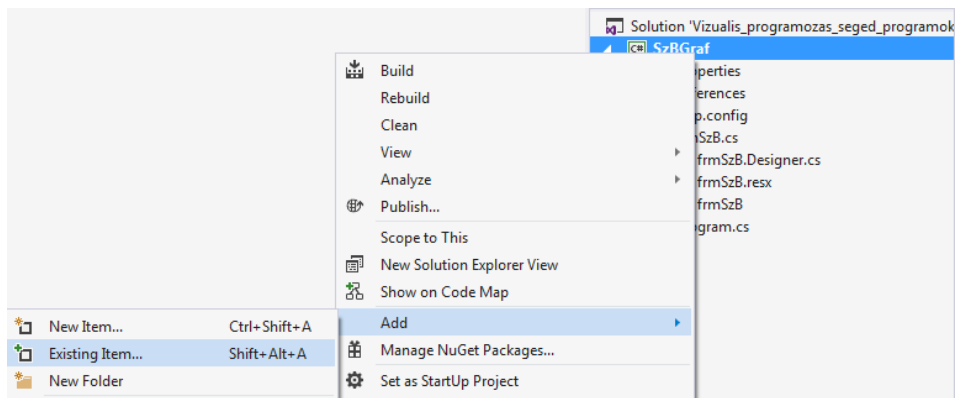
### 2.4.3. Eseménykezelő a Kilép nyomógombhoz

Hozunk létre egy eseménykezelőt a Kilép gombhoz úgy, hogy tervezési (Design) nézetben duplán kattintunk a nyomógombon. Az eseménykezelőben csak egyetlen utasítás szerepeljen:

```
private void btnKilep_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

### 2.4.4. Az SzBArray osztály hozzáadása a projekthez és kisebb módosítása

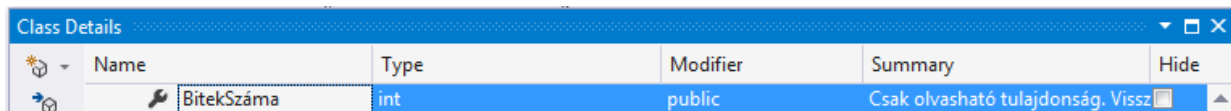
Másoljuk át a SzovegBittombbe.cs állományt a SzövegBittömbe órai projektből az aktuális projekt könyvtárába. Vegyük fel az aktuális projektbe az állományt. Ehhez a Solution Explorerben kattintsunk jobb egérgombbal a projekt nevére..., majd Add, Existing Item ...



87. ábra.

Nyissuk meg az állományt úgy, hogy a Solution Explorerben duplán kattintunk a SzovegBittombbe.cs állománynéven. Töröljük ki a 75-89 programsorokat (FO osztály).

Töröljük a Bitszám adattagot. Ehhez először a Solution Explorerben válasszuk ki a projektet, majd hozzunk létre egy osztálydiagramot. A diagramon kiválasztjuk az SzBArray osztály Bitszám adattagját, majd jobb egérgomb és a gyorsmenüben Delete Code.



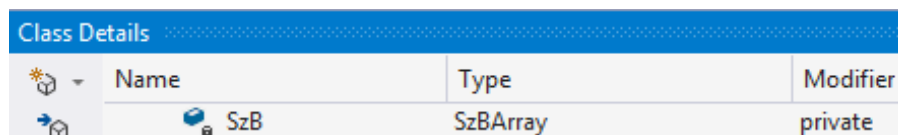
88. ábra.

A Class Details ablakban hozzunk létre egy új int típusú, nyilvános tulajdonságot BitekSzama néven. Kattintsunk duplán a Name oszlopban a nevet megelőző ikonra, majd a kódszerkesztőben töröljük a set elérőt. Ezáltal a tulajdonság csak olvashatóvá válik. A get elérőt az alábbi kódrészletnek megfelelően írjuk meg tudva, hogy a tulajdonság célja a tárolt bitek számának lekérdezése.

```

/// <summary>
/// Csak olvasható tulajdonság. Visszaadja a bitek számát.
/// </summary>
public int BitekSzama
{
    get
    {
        return BitTomb.Length;
    }
}
    
```

#### 2.4.5. SzBArray típusú adattag létrehozása az ablak osztályában



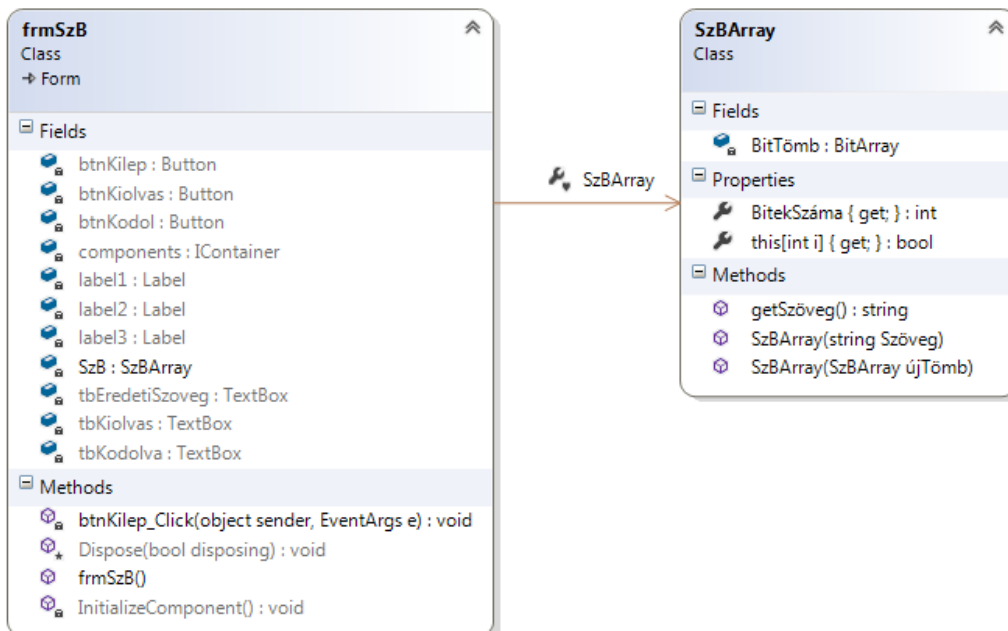
87. ábra.

Hozzunk létre egy **SzovegBittombbe.SzBArray** típusú és SzB nevű private adattagot az ablakot jelképező **frmSzB** osztályban vizuálisan a mellékelt ábra szerint vagy közvetlenül begépelve az alábbi kódot:

```
private SzövegBittömbbe.SzBArray SzB;
```

Hozunk létre egy SzövegBittömbbe.SzBArray típusú és SzB nevű tulajdonságot az ablakot jelképező frmSzB osztályban. Ehhez lépünk át az osztálydiagramba. A Toolbox palettán nyissuk ki a Class Designer csoportot, válasszuk ki a Association elemet, majd a frmSzB-ből kiindulva kössük össze a frmSzB-t és az SzBArray-t. A kódszerkesztőben az az alábbi mintakód alapján írjuk meg a szükséges módosításokat.

```
internal SzövegBittömbbe.SzBArray SzBArray
{
    get
    {
        if (SzB == null)
            throw new Exception("Még nem helyezett el szöveget "+ "a bittömbbe!");
        return SzB;
    }
}
```



89. ábra.

## 2.4.6. Kódolás megvalósítása

Hozunk létre egy eseménykezelőt a Kódol gombhoz úgy, hogy tervezési (Design) nézetben duplán kattintunk a nyomógombon. Az eseménykezelőt az alábbi mintakód alapján írjuk meg.

```
private void btnKodol_Click(object sender, EventArgs e)
{
    if (tbEredetiSzoveg.Text.Length == 0)
    {
        MessageBox.Show("Még nem adta meg az 'Eredeti szöveget'",
            "Hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    SzB = new SzövegBittömbbe.SzBArray(tbEredetiSzoveg.Text);
    string s = "";
    for(int i =0; i<SzB.BitekSzama; i++)
    {
        s = s + ((int)(SzB[i] ? 1 : 0)).ToString();
        tbKodolva.Text = s;
    }
}
```

```
    }  
}
```

### 2.4.7. Kiolvasás megvalósítása

Hozunk létre egy eseménykezelőt a Kiolvas gombhoz úgy, hogy tervezési (Design) nézetben duplán kattintunk a nyomógombon. Az eseménykezelőt az alábbi mintakód alapján írjuk meg.

```
private void btnKiolvas_Click(object sender, EventArgs e)  
{  
    try{ tbKiolvas.Text = SzBArray.getSzoveg(); }  
    catch(Exception exc)  
    {  
        MessageBox.Show(exc.Message, "Hiba", MessageBoxButtons.OK,  
        MessageBoxIcon.Error);  
    }  
}
```



## 2.5. Analóg óra

Készítsünk egy analóg órát megjelenítő alkalmazást.

A feladat egy lehetséges megoldása a következő:

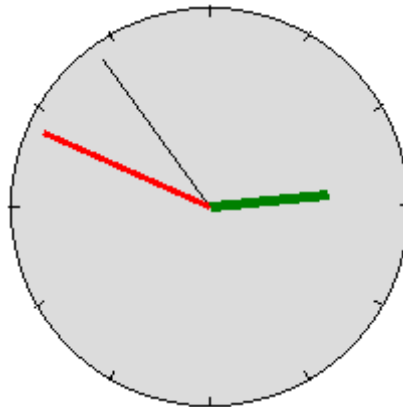
Az alkalmazás vázának automatikus generálása

Fájl menü, New, Project ...

Project Type: Visual C# - Windows

Templates: Windows Forms Application

Name: AnalogOra



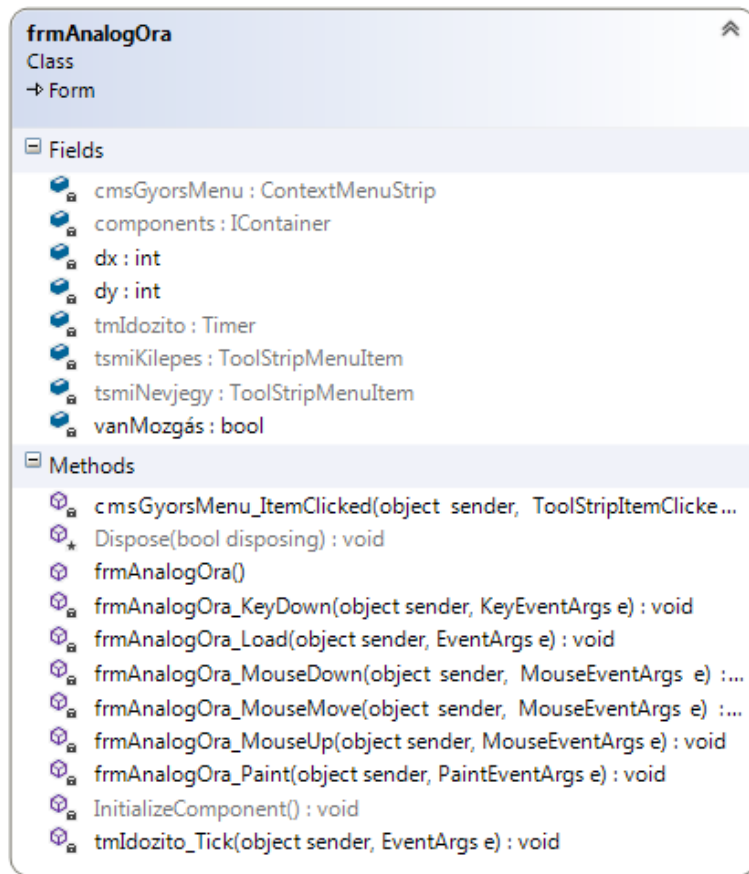
90. ábra.

### 2.5.1. Felület kialakítása

A Solution Explorerben nevezzük át a formot tartalmazó forrásállományt **frmAnalogOra.cs**-re. A formot nevezzük át (**Name=frmAnalógÓra**). Nincs szükségünk fejlécre és keretre, ezért **FormBorderStyle=None**, valamint az ablak legyen mindig látható: **TopMost=True**.

Az ablak mérete legyen 200x200-as. Ezt a konstruktorban állítjuk be:

```
public frmAnalogOra()
{
    InitializeComponent();
    Width = 200;
    Height = 200;
}
```



91. ábra.

## 2.5.2. Időkezelés

Az óra animációt egy Timer komponens segítségével oldjuk meg. Ez minden másodpercben egy Tick eseményt fog generálni. Ennek feldolgozása során frissítjük (újrarajzoljuk) a rajzterületet.

Tegyünk a formra egy Timer komponenst. Tulajdonságai: **Name=tmIdozito** és **Interval=1000** (ez 1 másodperc). Az ablak konstruktorában engedélyezzük a Timert. Ennek hatására a program indulásakor rögtön aktivizálódik az időzítő, azaz működik az óra.

```
tmIdozito.Enabled = true;
```

Az időzítő minden másodpercben generál egy Tick eseményt. Készítsünk ehhez egy eseménykezelőt, amiben előírjuk a rajzterület (form) frissítését, azaz egy Paint esemény előidézését. A formra a Paint esemény feldolgozása során fogunk rajzolni.

```
private void tmIdozito_Tick(object sender, EventArgs e)
{
    this.Refresh();
}
```

### 2.5.3. Rajzolás (mutatók)

Készítsünk az ablak Paint eseményéhez egy eseménykezelőt. Minden másodpercben keletkezik egy Paint esemény, és ennek feldolgozása során jelenítjük meg az órát. Vonalak rajzolásához minden egyes vonaltípushoz, még ha csak színben térnek is el egymástól, egy külön Pen objektumot kell definiálnunk. A kifestéshez Brush objektumot kell definiálnunk. A rajzolás egy Graphics típusú objektum segítségével lehetséges, ennek metódusai az egyes rajzoló függvények. Referenciáját az eseménykezelő metódus második paraméterének egy adattagjaként kapjuk meg.

```

/// <summary>
/// Megrajzolja az órát és a pontos időt.
/// </summary>
private void frmAnalogOra_Paint(object sender, PaintEventArgs e)
{
    //A háttér kifestéséhez szükség ecset objektum definiálása
    SolidBrush sbEcset = new SolidBrush(Color.Gainsboro);
    //Kifestett óra számlap megrajzolása
    e.Graphics.FillEllipse(sbEcset, 1, 1, Width - 2, Height - 2);
    //Toll definiálása a számlap keretvonalához
    Pen pToll = new Pen(Color.Black);
    //Keretvonal megrajzolása
    e.Graphics.DrawEllipse(pToll, 1, 1, Width - 2, Height - 2);
    //Középpont pozíciójának meghatározása
    int kozepPontX = Width / 2;
    int kozepPontY = Height / 2;
    //Jelzővonalak rajzolása 5 percenként
    for(int i = 0; i<60; i = i + 5)
    {
        e.Graphics.DrawLine(pToll,
            (int)(kozepPontX + (kozepPontX - 5) *
            Math.Sin(i * Math.PI / 30)),
            (int)(kozepPontY - (kozepPontX - 5) *
            Math.Cos(i * Math.PI / 30)),
            (int)(kozepPontX + kozepPontX *
            Math.Sin(i * Math.PI / 30)),
            (int)(kozepPontY - kozepPontY *
            Math.Cos(i * Math.PI / 30)));
    }
    //A lefoglalt erőforrások felszabadítása
    sbEcset.Dispose();
    pToll.Dispose();
    //Idő lekérdezése
    DateTime ido = DateTime.Now;
    /*Másodperc mutató megrajzolása*/
    int mutatoMp = (int)(0.9 * kozepPontX); //másodpercmutató hossza
    //Toll létrehozása a másodpercmutatóhoz
    Pen pMpToll = new Pen(Color.Black, 1);
    //Vonalhúzás a középponttól
    e.Graphics.DrawLine(pMpToll, kozepPontX, kozepPontY,
        (int)(kozepPontX + mutatoMp * Math.Sin(ido.Second * Math.PI / 30)),
        (int)(kozepPontY - mutatoMp * Math.Cos(ido.Second * Math.PI / 30)));
    //Erőforrás felszabadítása
    pMpToll.Dispose();
    /*Perc mutató megrajzolása*/
    int mutatoP = (int)(0.9 * kozepPontX); //A percmutató hossza
    //Toll létrehozása a percmutatóhoz
    Pen pPToll = new Pen(Color.Red, 3);

```

```
//Vonalhúzás középponttól
e.Graphics.DrawLine(pPToll, kozepPontX, kozepPontY,
(int)(kozepPontX + mutatoP *
Math.Sin(ido.Minute * Math.PI / 30)),
(int)(kozepPontY - mutatoP *
Math.Cos(ido.Minute * Math.PI / 30)));
//Erőforrás felszabadítása
pPToll.Dispose();
/*Óra mutató megrajzolása*/
int mutatoO = (int)(0.6 * kozepPontX); //Az óramutató hossza
//Toll létrehozása az óramutatóhoz
Pen pOToll = new Pen(Color.Green, 5);
//Vonalhúzás középponttól
e.Graphics.DrawLine(pOToll, kozepPontX, kozepPontY,
(int)(kozepPontX + mutatoO *
Math.Sin((ido.Hour + ido.Minute / 60.0) * Math.PI / 6)),
(int)(kozepPontY - mutatoO *
Math.Cos((ido.Hour + ido.Minute / 60.0) * Math.PI / 6)));
//Erőforrás felszabadítása
pOToll.Dispose();
}
```

#### 2.5.4. Esc billentyű lenyomására kilépés a programból

Készítünk egy eseménykezelőt az ablakon történő billentyűlenyomás (KeyDown) eseményhez

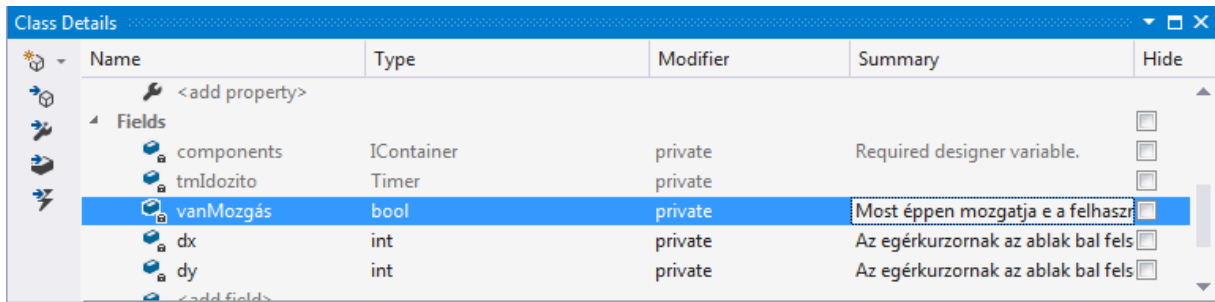
```
/// <summary>
/// ESC billentyű hatására kilépés a programból
/// </summary>
private void frmAnalogOra_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Escape)
    {
        tmIdozito.Enabled = false;
        Application.Exit();
    }
}
```

#### 2.5.5. Legyen az óra ablak kör alakú

Állítsuk be az ablak háttérszínét pl. kékre: **BackColor=Blue**. Legyen a **TransparencyKey=Blue**, így futásidőben csak a kéktől eltérő rész marad meg az ablakból.

#### 2.5.6. Legyen mozgatható az ablak az egér segítségével

Létrehozunk egy bool adattagot VanMozgás néven, feladata azon információ tárolása, hogy most éppen mozgatja-e a felhasználó az ablakot. Létrehozunk két int adattagot dx és dy néven, feladatuk az egérkurzornak az ablak bal felső sarkától mért vízszintes és függőleges távolságának tárolása.



92. ábra.

Készítünk egy eseménykezelőt az egérgomb lenyomásához (MouseDown). A bal egérgomb lenyomásakor VanMozgás-t igazra állítjuk, és tároljuk, hogy az egér hol van az ablak bal felső sarkához képest.

```

/// <summary>
/// A bal egérgomb lenyomásakor vanMozgas-t igazra állítjuk, és
/// tároljuk, hogy az egér hol van az ablak felső sarkához képest.
/// </summary>
private void frmAnalogOra_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        vanMozgás = true;
        dx = e.X;
        dy = e.Y;
    }
}

```

A bal egérgomb felengedésekor vége az ablakmozgatásnak. Készítünk egy eseménykezelőt az egérgomb felengedéséhez (MouseUp). Ebben a VanMozgás-t hamisra állítjuk.

```

/// <summary>
/// A bal egérgomb felengedésekor vége az ablakmozgásnak.
/// A vanMozgást hamisra állítjuk.
/// </summary>
private void frmAnalogOra_MouseUp(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        vanMozgás = false;
    }
}

```

Készítünk egy eseménykezelőt az egér mozgatás (MouseMove) eseményhez. Egér mozgatásakor ellenőrizzük, hogy a mozgatás be van-e kapcsolva. Ha igen, akkor az egér koordinátáit átszámoljuk ablak koordináta-rendszerből képernyő koordináta-rendszerbe. Az így kapott értékekből levonjuk az előzőekben meghatározott ablak bal felső sarok - egér kurzor távolságokat (dx és dy). A kapott eredmény adja az ablak bal felső sarkának új pozícióját.

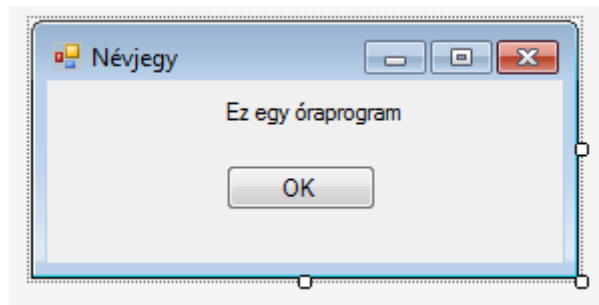
```

private void frmAnalogOra_MouseMove(object sender, MouseEventArgs e)
{
    if (vanMozgás)
    {
        Point p = new Point(e.X, e.Y);
        p = PointToScreen(p);
        Left = p.X - dx;
        Top = p.Y - dy;
    }
}

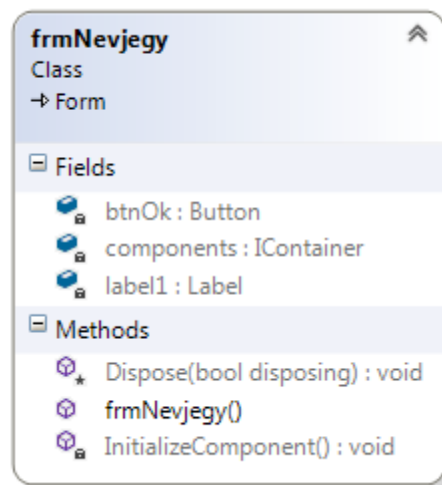
```

}

### 2.5.7. Névjegy panel készítése



93. ábra.



94. ábra.

Project menü, Add Windows Form...

Templates: Windows Form

Name: frmNevjegy.cs

Text=Névjegy és Name=frmNevjegy

Elhelyezünk egy címkét (Label) valamilyen felirattal (Text=...), és egy nyomógombot (Button) OK felirattal.

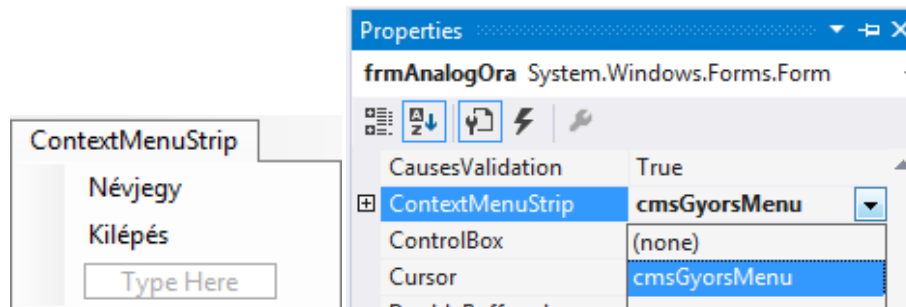
A nyomógomb tulajdonságai:

Text=OK és DialogResult=OK

Ez utóbbi azt eredményezi, hogy a gombon kattintva bezáródik a névjegyablak, és az őt eredetileg megjelenítő ShowDialog() metódus DialogResult.OK értékkel tér vissza.

### 2.5.8. Gyorsmenü készítése

Egy ContextMenuStrip típusú komponenst helyezünk a formra (frmAnalogOra), neve: **Name=cmsGyorsMenu**. Kijelöljük, majd a formon megjelenő menüszerkesztőben létrehozunk egy Névjegy (**Name=tsmiNevjegy**) és egy Kilépés (**Name=tsmiKilepes**) menüpontot. Köztük legyen egy elválasztó vonal. A form tulajdonságaiban beállítjuk a gyorsmenüt: ContextMenuStrip=cmsGyorsMenu



95. ábra.

### 2.5.9. Eseménykezelő készítése a gyorsmenühöz.

Tervezési nézetben kiválasztjuk a gyorsmenüt, majd a Properties ablakban duplán kattintunk az ItemClicked eseményen. A kiválasztott menüpont felirata alapján készítjük a feltételes elágazás szerkezetet az egyes menüpontokhoz tartozó funkcionalitás megvalósítására.

```
private void cmsGyorsMenu_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    switch (e.ClickedItem.Text)
    {
        case "Kilépés":
            Application.Exit();
            break;
        case "Névjegy":
            frmNevjegy frmNevjegy = new frmNevjegy();
            frmNevjegy.ShowDialog();
            break;
    }
}
```

### 2.5.10. Főablak kezdőpozíciója

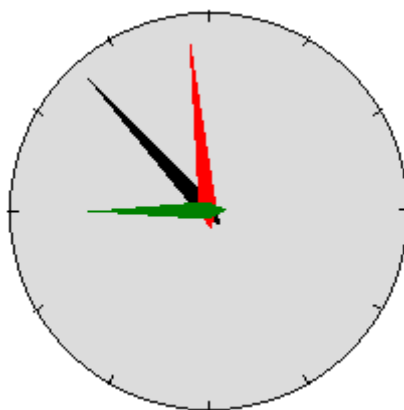
Beállítjuk a főablak kezdőpozícióját a 300,300-as koordinátájú pontba.

Ehhez eseménykezelőt készítünk a főablak Load eseményéhez.

```
/// <summary>
/// Beállítjuk a főablak kezdőpozícióját a
/// 300,300-as koordinátájú pontba
/// </summary>
private void frmAnalogOra_Load(object sender, EventArgs e)
{
    Left = 300;
    Top = 300;
}
```

### 2.5.11. Házi feladat

Alakítsa át úgy a programot, hogy a mutatókat ne egy vonal jelképezze, hanem a mellékelt ábrának megfelelő alakjuk legyen.



96. ábra.



## 3. Windows Presentation Foundation

### 3.1. Kétdimenziós rajzolás WPF-ben

A grafikus megjelenítés módjai WPF-ben:

**System.Windows.Shapes** névtér osztályaival

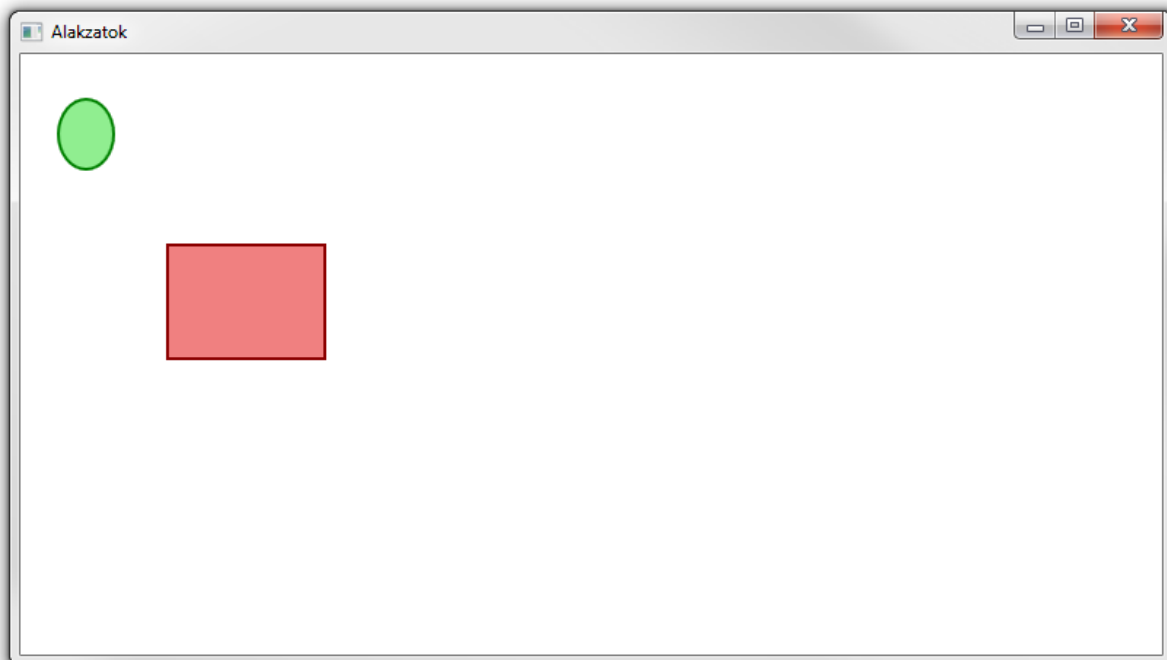
- Magas szintű, rengeteg metódus, tulajdonságok, eseménykezelés, input kezelés (egér, billentyűzet) → lassú; egy sor szabályos geometriai objektum (téglalap, ellipszis, stb.)
- Hozzunk létre egy Canvast XAML-ben **cvLap** néven, melyre a későbbiekben rajzolunk:  
`<Canvas Name="cvLap"></Canvas>`

- Egyszerűen leírható XAML-ben:

```
<Ellipse Width="40" Height="50" Stroke="Green" StrokeThickness="2"
Fill="LightGreen" Canvas.Left="25" Canvas.Top="30" Name="eLomb"/>
```

- A rajzolás C# kódból is megvalósítható:

```
double tetoMagassag = 30;
Rectangle rcHaz = new Rectangle();
rcHaz.Width = 110;
rcHaz.Height = 80;
rcHaz.Stroke = Brushes.DarkRed;
rcHaz.StrokeThickness = 2;
rcHaz.Fill = Brushes.LightCoral;
cvLap.Children.Add(rcHaz);
rcHaz.SetValue(Canvas.LeftProperty, (double)100);
rcHaz.SetValue(Canvas.TopProperty,
(double)(100 + tetoMagassag));
```



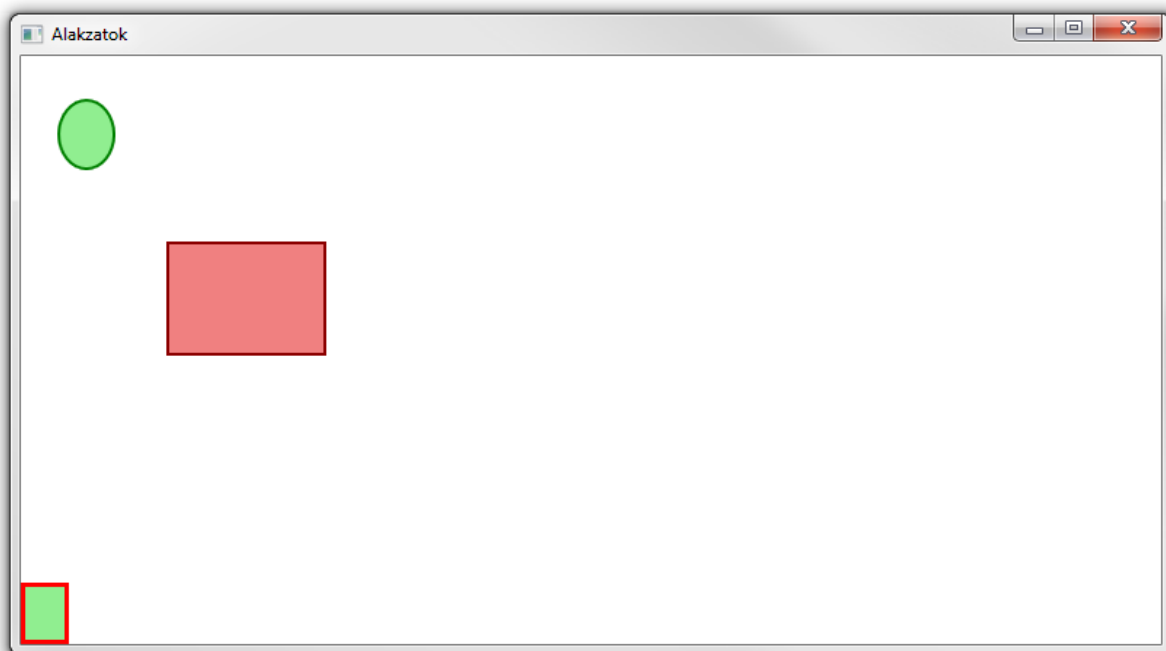
97. ábra.

- A tárolóra elhelyezett alakzatok között van egy Z-sorrend, ami azt jelenti, hogy a később feltett alakzatok elfedhetik a korábban feltett alakzatokat (pl. ha az elsőnek feltett alakzatot átmozgatjuk a másodiknak feltett alakzat pozíciójába, akkor az első a második alá kerül).

#### **System.Windows.Media.Drawing** absztrakt osztály leszármazottaival

- Vékonyabb réteg (ún. Pehelysúlyú szolgáltatások) → gyorsabb, kisebb erőforrásigény
- Nincs beépített input kezelés
- Valamilyen hoszt objektumban kell elhelyezni (pl. DrawingImage, DrawingBrush, DrawingVisual)
- Több kód szükséges
- Fontosabb osztályok: GeometryDrawing, ImageDrawing
- Leírható XAML-ben:

```
<Image Canvas.Left="0" Canvas.Bottom="0">  
  <Image.Source>  
    <DrawingImage>  
      <DrawingImage.Drawing>  
        <GeometryDrawing Brush="LightGreen">  
          <GeometryDrawing.Pen>  
            <Pen Brush="Red" Thickness="3"/>  
          </GeometryDrawing.Pen>  
          <GeometryDrawing.Geometry>  
            <RectangleGeometry Rect="0,0,30,40"/>  
          </GeometryDrawing.Geometry>  
        </GeometryDrawing>  
      </DrawingImage.Drawing>  
    </DrawingImage>  
  </Image.Source>  
</Image>
```



98. ábra.

**System.Windows.Media.Visual** absztrakt osztály leszármazottaival

- Legvékonyabb réteg → leggyorsabb; csak elemi szolgáltatások, mindenhez meg kell írni a kódot (legtöbb kódolás)
- nincs input esemény, felületmenedzser, adatkötés, alacsony szintű megközelítés
- Fontosabb osztályok: DrawingVisual, Viewport3DVisual, ContainerVisual
- Legkisebb erőforrásigény → Legjobb teljesítmény
- valamilyen hoszt objektumban kell elhelyezni (pl. DrawingImage, DrawingBrush, DrawingVisual)
- XAML-ből általában nem oldható meg
- Rajzolási kapcsolatot/eszközkapcsolatot kell létrehozni és megnyitni, majd a rajzolást követően lezárni (using szerkezet használható)
- Az új objektumot el kell helyezni a logikai és a vizuális fában.
- Át kell definiálni a VisualChildrenCount virtuális tulajdonságot.
- Át kell definiálni a GetVisualChild virtuális metódust.

**3.1.1. Feladat**

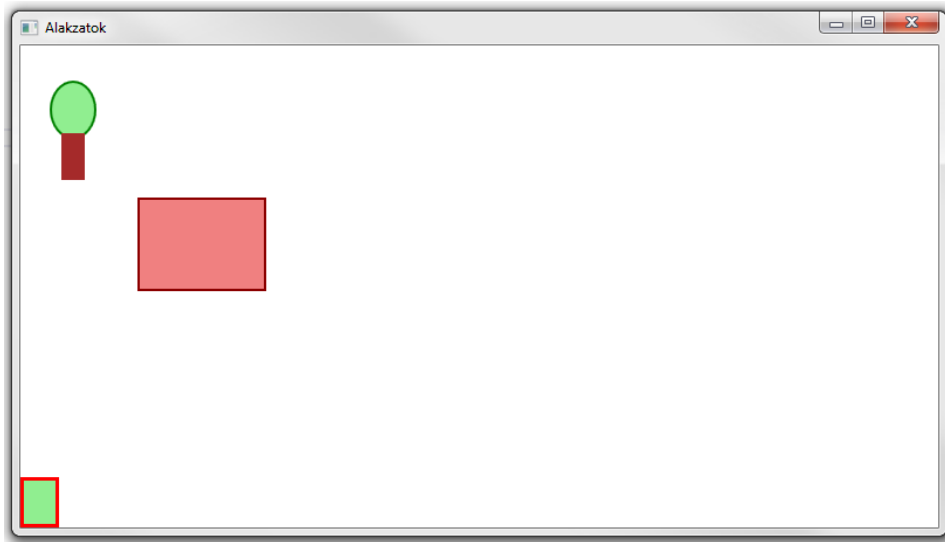
Készítsen egy WPF alkalmazást, ami

- Megrajzolja a képen látható fát (XAML-ből) és házat (programból) a „System.Windows.Shapes” megoldással.
- Az egér segítségével mozgathatóvá teszi a fát attól függetlenül, hogy a törzsön vagy a lombnál fogjuk-e meg.
- A fán (törzsön vagy lombon) kattintva jobb egérgombbal egy gyorsmenü jelenik meg (Töröl és Előre hoz menüpontokkal).
- Töröl: törli a fát.
- Előre hoz: a Z sorrend végére helyezi a fát, ami azt eredményezi, hogy amikor a ház területére húzzuk az egérrel, akkor elfedi a házat.

**3.1.2. Megoldás**

Az ablakot leíró XAML kód:

```
<Rectangle Width="20" Height="40" Stroke="Brown" StrokeThickness="2" Fill="Brown"
Canvas.Left="35" Canvas.Top="75" Name="rcTorzs"/>
```



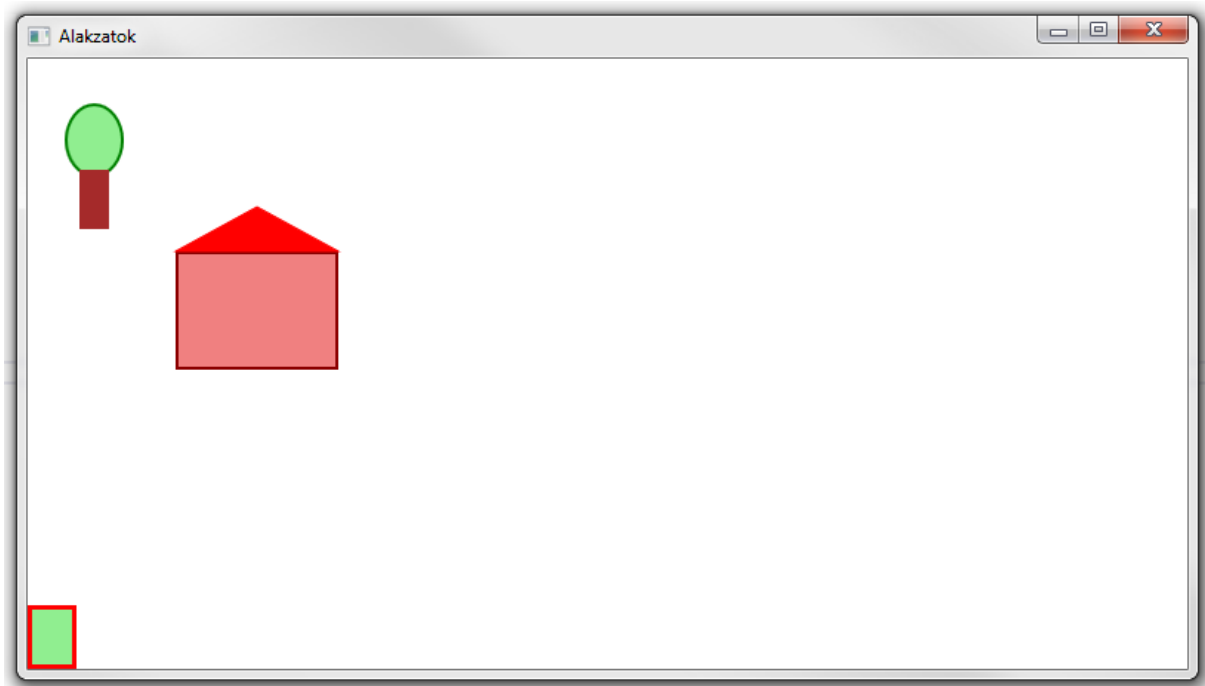
99. ábra.

A ház megrajzolását végző metódus:

```

/// <summary> Megrajzolja a házat.</ summary>
/// <param name="x">A házat befoglaló téglalap bal felső sarkának X
koordinátája</param>
/// <param name="y">A házat befoglaló téglalap bal felső sarkának Y
koordinátája</param>
public void Haz(double x, double y)
{
    //A tető magassága
    double tetoMagassag = 30;
    //A földszintet leíró téglalap definiálása.
    Rectangle rcHaz = new Rectangle();
    rcHaz.Width = 110;
    rcHaz.Height = 80;
    //Keretvonal színe
    rcHaz.Stroke = Brushes.DarkRed; //A fal színe
    rcHaz.StrokeThickness = 2; //A fal színe
    rcHaz.Fill = Brushes.LightCoral;
    //A ház helyzetének definiálása
    rcHaz.SetValue(Canvas.LeftProperty, (double)100);
    rcHaz.SetValue(Canvas.TopProperty,
    (double)(100 + tetoMagassag));
    //Elhelyezés a rajzlapon
    cvLap.Children.Add(rcHaz);
    //A tetőt leíró háromszög definiálása
    Polygon pgTeto = new Polygon();
    //Keretvonal színe
    pgTeto.Stroke = Brushes.Red;
    //Keretvonal vastagsága
    pgTeto.Fill = Brushes.Red;
    //A háromszög csúcsainak definiálása
    pgTeto.Points = new PointCollection();
    pgTeto.Points.Add(new Point(x, y + tetoMagassag));
    pgTeto.Points.Add(new Point(x + (rcHaz.Width / 2), y));
    pgTeto.Points.Add(new Point(x + rcHaz.Width,
    y + tetoMagassag));
    //Elhelyezés a rajzlapon
    cvLap.Children.Add(pgTeto);
}

```



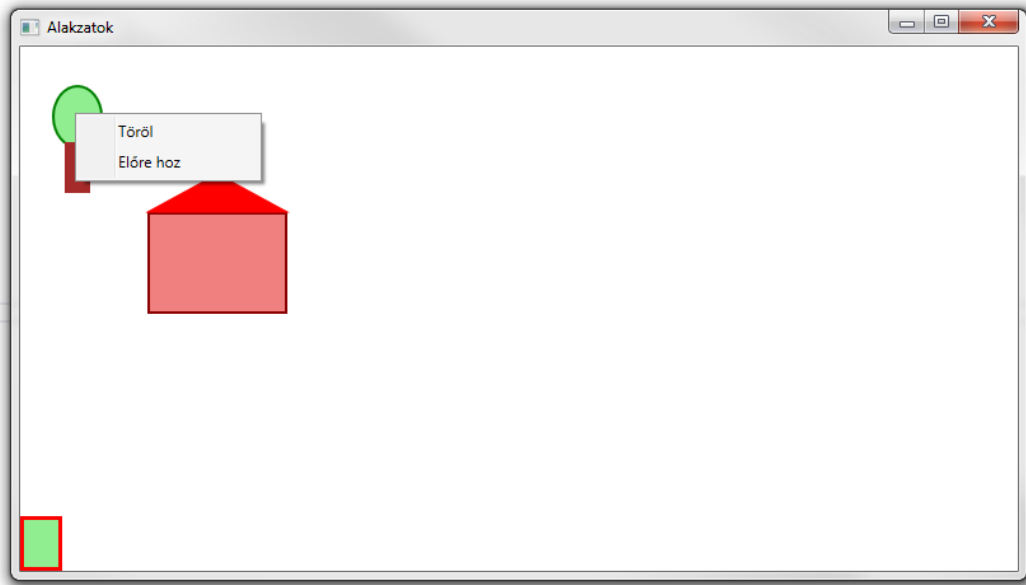
100. ábra.

A gyorsmenüt létrehozó metódus:

```

/// <summary>Létrehozza és a fához rendeli a gyorsmenüt</summary>
private void GyorsMenuLetrehoz()
{
    //Gyorsmenü definiálása
    ContextMenu cmGyorsMenu = new ContextMenu();
    //Töröl menüpont definiálása
    MenuItem miTorol = new MenuItem();
    //Megjelenő szöveg
    miTorol.Header = "Töröl";
    //Eseménykezelő hozzárendelése
    miTorol.Click += new RoutedEventHandler(miTorol_Click);
    //Hozzáadás a gyorsmenühöz
    cmGyorsMenu.Items.Add(miTorol);
    //Előre hoz menüpont definiálása
    MenuItem miEloreHoz = new MenuItem();
    //Megjelenő szöveg
    miEloreHoz.Header = "Előre hoz";
    //Eseménykezelő hozzárendelése
    miEloreHoz.Click += new RoutedEventHandler(miEloreHoz_Click);
    //Hozzáadás a gyorsmenühöz
    cmGyorsMenu.Items.Add(miEloreHoz);
    //Gyorsmenü hozzárendelése a lombot megvalósító objektumhoz
    elLomb.ContextMenu = cmGyorsMenu;
    //Gyorsmenü hozzárendelése a fatörzset megvalósító objektumhoz
    rcTorzs.ContextMenu = cmGyorsMenu;
}

```



101. ábra.

Az ablak konstruktora:

```

/// <summary>
/// Az ablakosztály konstruktora. Gondoskodik az XAML-ben
/// leírt felület megjelenítéséről, a ház megrajzolásáról és a
/// gyorsmenü létrehozásáról.
/// </summary>
public MainWindow()
{
    InitializeComponent();//Az XAML-ben leírt felület megjelenítése
    Haz(100,100); //A ház megrajzolása
    GyorsMenuLetrehoz();//A gyorsmenü létrehozása
}

```

A fa törlésének megvalósítása:

```

/// <summary>Törli a fát megjelenítő két alakzat objektumot a megjelenítendő ///
objektumok listájáról</summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void miToro1_Click(object sender, RoutedEventArgs e)
{
    cvLap.Children.Remove(e1Lomb);
    cvLap.Children.Remove(rcTorzs);
}
/// <summary>A Z-sorrend végére helyezi a fát</summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void miEloreHoz_Click(object sender, RoutedEventArgs e)
{
    //Töröljük a fát megjelenítő két alakzat objektumot a //megjelenítendő objektumok
    listájáról
    cvLap.Children.Remove(e1Lomb);
    cvLap.Children.Remove(rcTorzs);
    //Újra felvesszük őket a lista végére
    cvLap.Children.Add(e1Lomb);
    cvLap.Children.Add(rcTorzs);
}

```

A fa mozgatásának megvalósítása:

- Az egérgomb lenyomásakor (ha az a fa területén történik) tároljuk az egér helyzetét.
- Egér mozgatás eseménykor (ha az a fa területén történik) ha a bal oldali egérgomb le van nyomva
  - Lekérdezzük az egér helyzetét.
  - Kiszámoljuk, hogy mennyit mozdult el az előző pozícióhoz képest.
  - Lekérdezzük a lombot befoglaló téglalap bal felső sarkának helyzetét.
  - Elmozgatjuk a lombot.
  - Lekérdezzük a fatörzset befoglaló téglalap bal felső sarkának helyzetét.
  - Elmozgatjuk a fatörzset.
  - Tároljuk az egér aktuális helyzetét.

```

/// <summary>
/// Egérgomb mozgatása esemény (a fa területén) kezelője.
/// Gondoskodik a fa elmozdításáról.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void elLomb_MouseMove(object sender, MouseEventArgs e)
{
    //Ha a bal egérgomb le van nyomva
    if(e.LeftButton == MouseButtonState.Pressed)
    {
        //Lekérdezzük az egér helyzetét
        double ujx = e.GetPosition(cvLap).X;
        double ujy = e.GetPosition(cvLap).Y;
        //Kiszámoljuk, hogy mennyit mozdult el az előző pozícióhoz //képest
        double dx = ujx - x;
        double dy = ujy - y;
        //Lekérdezzük a lombot befoglaló téglalap bal felső //sarkának helyzetét.
        double lombx = (double)elLomb.GetValue(Canvas.LeftProperty);
        double lomby = (double)elLomb.GetValue(Canvas.TopProperty);
        //Elmozgatjuk a lombot
        elLomb.SetValue(Canvas.LeftProperty, lombx + dx);
        elLomb.SetValue(Canvas.TopProperty, lomby + dy);
        //Lekérdezzük a fatörzset befoglaló téglalap bal felső //sarkának helyzetét.
        double torzsx = (double)rcTorzs.GetValue(Canvas.LeftProperty);
        double torzsy = (double)rcTorzs.GetValue(Canvas.TopProperty);
        //Elmozgatjuk a torzset
        rcTorzs.SetValue(Canvas.LeftProperty, torzsx + dx);
        rcTorzs.SetValue(Canvas.TopProperty, torzsy + y);
        //Tároljuk az egér aktuális helyzetét
        x = ujx;
        y = ujy;
    }
}

```

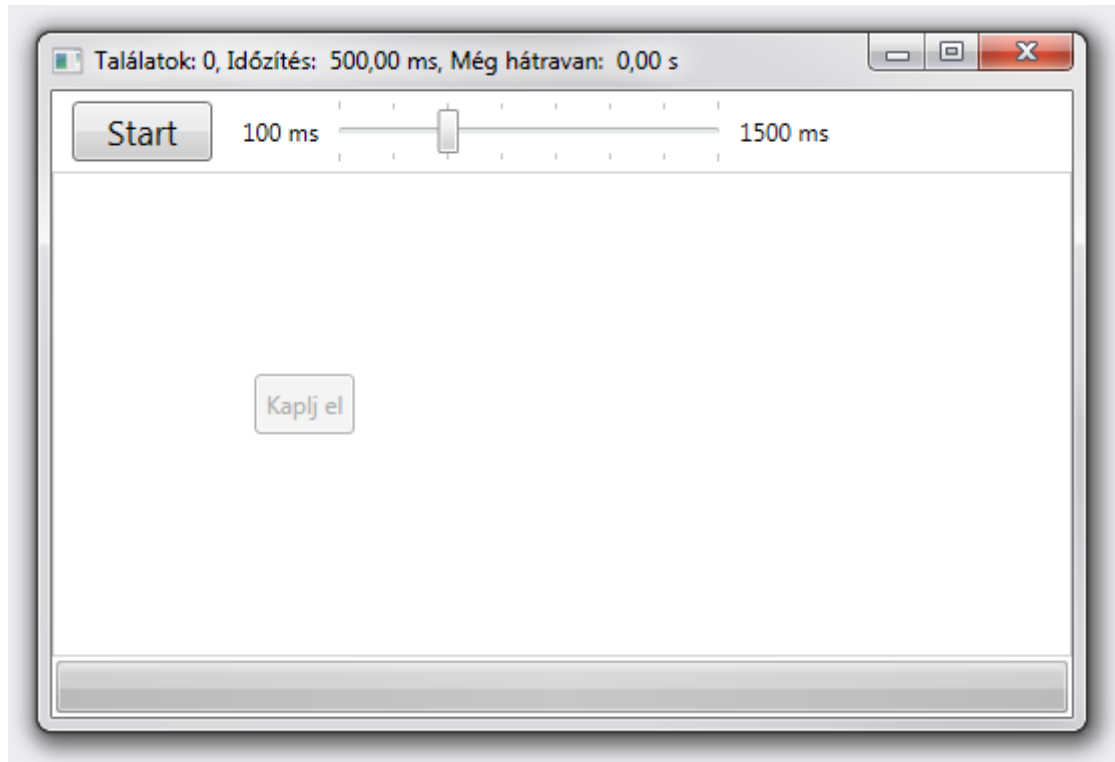
### 3.1.3. Egyénileg megoldandó feladat

- Készítsen a programból ablakot, ajtót és kéményt a házhoz.
- Egészítse ki a gyorsmenüt "Hátra viz" menüponttal, és valósítsa meg a hozzá tartozó funkcionalitást.

### 3.2. Ugráló gomb

Készítsünk egy egyszerű játékprogramot, ami egy mozgó nyomógombot tartalmaz. A nyomógomb beállított ideig marad egy helyben, majd az ablakon számára elhatárolt terület (panel) egy véletlenszerűen kiválasztott pozíciójában jelenik meg újra. A játék során az a feladat, hogy minél többször kattintsunk a gombon az egér segítségével. A játék előre beállított ideig tart, az ablak alján elhelyezett végrehajtásjelző tájékoztat az eltelt időről.

A feladat egy lehetséges megoldása a következő:



102. ábra.

#### 3.2.1. Az alkalmazás vázának automatikus generálása:

- Fájl menü, New, Project, Installed, Visual C# , Windows, WPF Application Name: UgraloGomb OK

#### 3.2.2. Felület kialakítása:

- Az ablak osztályának neve legyen wndUgrálóGomb, az őt definiáló állomány nevét is változtassuk meg wndUgraloGomb.xaml-re. Majd módosítsuk az App.xaml állományban a StartupUri attribútum értékét **StartupUri="wndUgraloGomb.xaml"**-re.
- A felületmenedzser neve legyen grRács. Hozzunk létre benne három sort. Az első sor magassága legyen 40, a harmadiké 30.
- Helyezzünk el egy StackPanel-t az első sorban spEszköztár névvel, úgy, hogy töltse ki a területet.
- Egy nyomógombot (Button) helyezzünk a StackPanelre . **x:Name=btStart, Text=&Start**. Ezzel lehet majd elindítani a játékot.



- Egy csúszkát (Slider) helyezünk el a nyomógomb jobb oldalára. **x>Name=slCsuszka**. Ezzel lehet majd beállítani, hogy mennyi ideig maradjon egy helyben a mozgó nyomógomb.
- Egy-egy címkét (Label) helyezünk el a csúszka bal és jobb oldalára. Feladatuk a csúszkával beállítható minimális és maximális időérték kijelzése. Nevük: **x>Name=llMin** és **x>Name=llMax**.
- Egy végrehajtásjelzőt (ProgressBar) helyezünk el az ablak aljába (a rács harmadik sora). Ez fog tájékoztatni az eltelt játékidőről. Tulajdonságai: **x>Name=pbVegrehajtasJelzo**.
- Egy keretet helyezünk el a rács második sorába, ebbe kerül a játéktér. Erre azért van szükség, hogy jól láthatóan elkülönítsük a játéktérrel az ablak többi részétől. Tulajdonságai: **x>Name=brKeret**
- A mozgó nyomógomb helyzetét a bal felső sarkainak koordinátaival szeretnénk szabályozni, ezért a játéktérrel egy Canvas komponenssel valósítjuk meg. Tulajdonságai: **Name=cvLap, HorizontalAlignment="Stretch"**
- Egy nyomógombot (Button) helyezünk a játéktérre, ezen kell kattintani játék közben. Fontosabb tulajdonságai: **Name=btKapjEl, Text=Kapj el!**

A felület teljes definciója az alábbi:

```
<Grid x>Name="grRacs">
  <Grid.RowDefinitions>
    <RowDefinition Height="40" />
    <RowDefinition />
    <RowDefinition Height="30" />
  </Grid.RowDefinitions>
  <StackPanel x>Name="spEszkoztar" Grid.Row="0" HorizontalAlignment="Stretch"
  Height="30" Orientation="Horizontal">
    <Button x>Name="btStart" Width="70" FontSize="14" Margin="10,0"
    Click="btStart_Click">_Start</Button>
    <Label x>Name="llMin" Content="ms" VerticalAlignment="Center" />
    <Slider x>Name="slCsuszka" VerticalAlignment="Center" Width="200"
    TickPlacement="Both" ValueChanged="slCsuszka_ValueChanged" />
    <Label x>Name="llMax" Content="ms" VerticalAlignment="Center" />
  </StackPanel>
  <Border x>Name="brKeret" Grid.Row="1" BorderThickness="1"
  BorderBrush="LightGray" >
    <Canvas x>Name="cvLap" >
      <Button x>Name="btKapjEl" Canvas.Left="100" Canvas.Top="100" Width="50"
      Height="30" Content="Kapj el!" Click="btKapjEl_Click"
      MouseEnter="btKapjEl_MouseEnter" MouseLeave="btKapjEl_MouseLeave" />
    </Canvas>
  </Border>
  <ProgressBar x>Name="pbVegrehajtasJelzo" Grid.Row="2" />
</Grid>
```

### 3.2.3. Adattagok definiálása

Hozunk létre az ablak osztályában egy adattagot az elért pontszám tárolására.

```
/// <summary>
/// Az elért pontszám.
/// </summary>
private int eredmény;
```

Hozzunk létre az ablak osztályában egy adattagot véletlenszámok előállítására szolgáló objektum referenciájának tárolására.

```
/// <summary>
/// Véletlenszámok előállítására szolgáló objektum.
/// </summary>
private Random veletlen;
```

Hozzunk létre egy adattagot a játék kezdő időpillanatának tárolására.

```
/// <summary>
/// A játék kezdete.
/// </summary>
private DateTime kezdodo;
```

Hozzunk létre egy adattagot a megengedett játékidő tárolására.

```
/// <summary>
/// Megengedett játékidő másodpercben.
/// </summary>
private int maxJatekIdo;
```

A játékidő méréséhez egy Timer objektumra lesz szükségünk. Mivel az időzítés lejártakor a felhasználói felületen kell végrehajtanunk változtatásokat, ezért a DispatcherTimer-t válasszuk a feladathoz.

```
/// <summary>
/// Időzítő a játékidő méréséhez és a gomb mozgatásához.
/// </summary>
private DispatcherTimer dtIdozito;
```

Hozzunk létre egy logikai adattagot, ami a későbbiekben a találatok érvényességének ellenőrzéséhez lesz szükséges.

```
/// <summary>
/// Meghatározza, hogy találatot jelent e a Click esemény.
/// </summary>
private bool ervenyes;
```

### 3.2.4. Kezdőérték adás a konstruktorban

Az alábbi utasításokkal beállítjuk az ablakon elhelyezett komponensek tulajdonságait. Az időzítő Tick eseményéhez kapcsolódó eseménykezelő (*dtIdozito\_Tick*) vázát a Visual Studioval generáltatjuk le automatikusan.

```
/// <summary>
/// Konstruktor. A kóponensek egyes tulajdonságainak beállítása.
/// </summary>
public wndUgraloGomb()
{
    InitializeComponent();
    // Timer objektum létrehozása a játékidő követésére és a gomb // mozgatásához.
    // Az időzítő alapbeállításaként 0,5
    // másodpercenként jelez. Időzítő kezdetben leállítva.
    dtIdozito = new DispatcherTimer
    {
        Interval = new TimeSpan(0, 0, 0, 0, 500), IsEnabled = false
    };
    //Eseménykezelő az időzítőhöz
    dtIdozito.Tick += dtIdozito_Tick;
```

```

// Alsó és felső határérték ezredmásodpercben arra, hogy mennyi // ideig maradhat
egy helyben a mozgó nyomógomb.
slCsuszka.Minimum = 100;
slCsuszka.Maximum = 1500;
//A csúszka jelzővonalainak távolsága.
slCsuszka.TickFrequency = 200;
// Mekkora elmozdulást jelent a csúszkán a le/fel nyíl //billentyű lenyomása?
slCsuszka.SmallChange = 100;
// Mekkora elmozdulást jelent a csúszkán a Page Up/Page Down
// billentyű lenyomása?
slCsuszka.LargeChange = 500;
// A csúszka kezdeti pozíciója.
slCsuszka.Value = 500;
// A csúszka bal és jobb oldali címkének (feliratok) szövege.
llMin.Content = slCsuszka.Minimum + " ms";
llMax.Content = slCsuszka.Maximum + " ms";
// A mozgó gomb kezdetben letiltva.
btKapjEl.IsEnabled = false;
// A megengedett játékidő másodpercben.
maxJatekIdo = 10;
// Végrehajtásjelző szélsőértékeihez társított számértékek.
pbVegrehajtasJelzo.Minimum = 0;
// A játékidő másodpercben.
pbVegrehajtasJelzo.Maximum = maxJatekIdo;
// Végrehajtásjelző kezdőértéke.
pbVegrehajtasJelzo.Value = 0;
// Véletlenszámokat előállító objektum létrehozása.
veletlen = new Random();
}

```

### 3.2.5. Eseménykezelő készítése a Tick eseményéhez.

Az eseménykezelő utasításblokkjában frissítjük a feliratot az ablak fejlécében, beállítjuk a végrehajtásjelzőt, és új pozícióba helyezzük a KapjEl gombot. A pozíciót meghatározó adattagok (*LeftProperty*, *TopProperty*) *DependencyProperty* típusúak, ezért értéküket nem tudjuk közvetlenül beállítani. Az értékadás a nyomógomb objektum *SetValue()* metódusa segítségével történik. Ha lejárt a játékidő, akkor le kell állítani az időzítőt. Az ablak fejlécébe történő feliratkiírás egy külön metódus feladata (*FeliratKiir()*), itt csak meghívjuk a metódust és generáltatjuk a vázát a Visual Studio segítségével.

```

/// <summary>
/// Eseménykezelő: lejárt az időzítő időegysége. Lépteti a
/// végrehajtásjelzőt. Frissíti a feliratot az ablak fejlécében.
/// Beállítja a végrehajtásjelzőt. Új pozícióba helyezi a KapjEl
/// gombot. Ha lejárt a játékidő, akkor leállítja az időzítőt,
/// letiltja a KapjEl gombot, és engedélyezi a Start gombot.
/// </summary>
private void dtIdozito_Tick(object sender, EventArgs e)
{
    // Állapot kiírása az ablak fejlécébe.
    FeliratKiir();
    // Beállítjuk a végrehajtásjelzőt.
    pbVegrehajtasJelzo.Value = ElteltIdo();
    // Ha még nem járt le a játékidő, gomb mozgatása új pozícióba.
    if (ElteltIdo() < maxJatekIdo)
    {

```

```

    // A gomb bal felső sarkának új x koordinátája.
    btKapjEl.SetValue(LeftProperty, veletlen.NextDouble() * (cvLap.ActualWidth -
    btKapjEl.ActualWidth));
    // A gomb bal felső sarkának új y koordinátája.
    btKapjEl.SetValue(TopProperty, veletlen.NextDouble() * (cvLap.ActualHeight -
    btKapjEl.ActualHeight));
}
else
{ // Ha lejárt a játékidő.
  // Időzítő leállítása.
  dtIdozito.IsEnabled = false;
  // Start gomb engedélyezése.
  btStart.IsEnabled = true;
  // Mozgó nyomógomb letiltása.
  btKapjEl.IsEnabled = false;
}
}
}

```

### 3.2.6. Az ablak fejlécében feliratot megjelenítő metódus definiálása.

Az ablak fejlécében ki akarjuk jelezni, hogy mennyi az eddig elért találatok száma, mennyi időközönként mozdul el a nyomógomb, és mennyi idő van még hátra a játékból. A feladat megoldásához szükségünk van a játék kezdetétől eltelt másodpercek számára. Ennek kiszámítását egy külön metódusban (*ElteltIdő()*) helyezzük el.

```

/// <summary>
/// Friss információkat jelenít meg a játék állásáról az ablak fejlécében.
/// </summary>
private void FeliratKiir()
{
    Title = string.Format("Találatok: {0}, Időzítés: {1,7:F2} ms, Még hátravan:
    {2,5:F2} s", eredmény, slCsuszka.Value, Math.Max(0, maxJatekIdo - ElteltIdo()));
}
/// <summary>Kiszámolja a játék kezdetétől eltelt időt másodpercben.</summary>
/// <returns>Eltelt idő.</returns>
double ElteltIdo()
{
    // Lekérdezzük az aktuális időt.
    DateTime most = DateTime.Now;
    // A játék kezdete óta eltelt idő.
    return most.Subtract(kezdoIdo).TotalSeconds;
}

```

### 3.2.7. Eseménykezelő készítése a Kapj El! Gombhoz

```

/// <summary>
/// Eseménykezelő: a felhasználó kattintott a KapjEl gombon.
/// Növeli egyel az eredményt és frissíti a feliratot az
/// ablak fejlécében. Csak akkor számol találatot, ha az
/// Érvényes adattag értéke igaz.
/// </summary>
private void btKapjEl_Click(object sender, RoutedEventArgs e)
{
    // Ha az érvényesség ellenőrzést nem építenénk be, akkor az
    // Enter gomb lenyomása is pontot eredményezne.
    if (!ervenyes) return;
    // Ha érvényes a kattintás, azaz a nyomógomb felett volt az
    // egér az esemény keletkezésekor.
    eredmény++;
}

```

```

    // Eredmény megjelenítése az ablak fejlécében.
    FeliratKiir();
}

```

Csak az egérrel elért találatokat tekintjük érvényesnek, ezért meg szeretnénk akadályozni, hogy a játékos az Enter gomb megnyomásával is pontot szerezzen. Az érvényesség ellenőrzést úgy oldjuk meg, hogy amikor az egér a KapjEl gomb felé kerül, akkor az Érvényes logikai adattagot igazra állítjuk, amikor elhagyja a nyomógomb területét, akkorpedig hamisra.

```

/// <summary>
/// Igazra állítja az Érvényes adattagot, amikor az egérkurzor belép
/// a nyomógomb területére.
/// </summary>
private void btKapjEl_MouseEnter(object sender, MouseEventArgs e)
{
    ervenyos = true;
}

```

```

/// <summary>
/// Hamisra állítja az Érvényes adattagot, amikor az egérkurzor
/// kilép a nyomógomb területéről.
/// </summary>
private void btKapjEl_MouseLeave(object sender, MouseEventArgs e)
{
    ervenyos = false;
}

```

### 3.2.8. A játékot elindító Start gomb eseménykezelőjének elkészítése.

```

/// <summary>
/// A Start gombon történő kattintásra reagáló eseménykezelő.
/// Kinullázza az eredményt tároló változót és a végrehajtásjelzőt
/// alapállapotba állítja. Az időzítőt a csúszka állapotához
/// igazítja és indítja. Engedélyezi a btKapjEl gombot, tiltja
/// a btStart gombot. Kezdeti feliratot jelenít meg az ablak
/// fejlécében.
/// </summary>
private void btStart_Click(object sender, RoutedEventArgs e)
{
    // Lenullázzuk az eredményt.
    eredmeny = 0;
    // Játékidő előlről kezdődik.
    kezdoIdo = DateTime.Now;
    // Végrehajtásjelző a kezdő pozícióba.
    pbVegrehajtasJelzo.Value = 0;
    // Az időzítő beállítása a csúszka állása alapján.
    dtIdozito.Interval = new TimeSpan(0, 0, 0, 0, (int)slCsuszka.Value);
    // Start gomb letiltása.
    btStart.IsEnabled = false;
    // Időzítő indítása.
    dtIdozito.IsEnabled = true;
    // Eredmény megjelenítése az ablak fejlécében.
    FeliratKiir();
    // Kapj el nyomógomb engedély
    btKapjEl.IsEnabled = true;
}

```

### 3.2.9. Eseménykezelő készítése a csúszka mozgatásához.

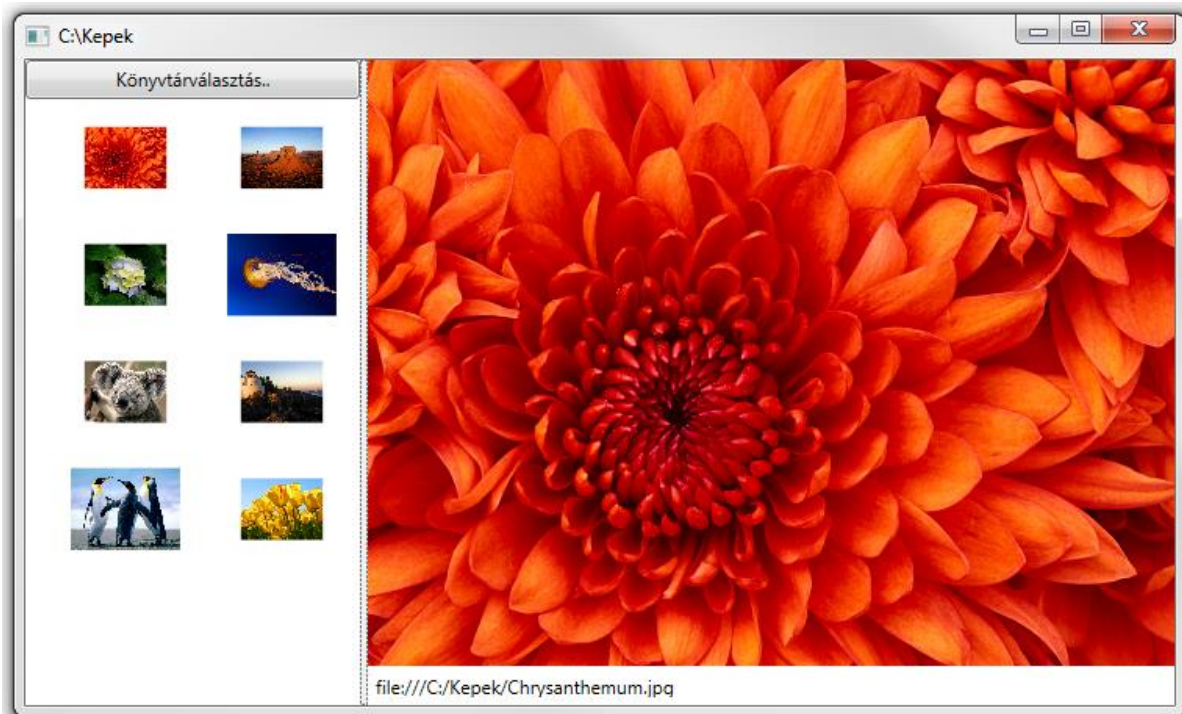
```
/// <summary>
/// Eseménykezelő: a felhasználó elmozdította a csúszkát.
/// Leállítjuk az időzítőt. A csúszka értékének megfelelően
/// beállítjuk az időzítés idejét. Ha mindez játékidőben történt,
/// akkor engedélyezzük az időzítőt. Frissítjük a feliratot az
/// ablak fejlécében.
/// </summary>
private void slCsuszka_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
{
    // Tároljuk, hogy most folyik-e játék.
    bool vanJatek = dtIdozito.IsEnabled;
    // Időzítő letiltása az intervallum módosítás miatt.
    dtIdozito.IsEnabled = false;
    // Időzítő intervallumának beállítása.
    dtIdozito.Interval = new TimeSpan(0, 0, 0, 0, (int)slCsuszka.Value);
    // A játék közben mozgatja a felhasználó a csúszkát, akkor
    if (vanJatek)
        dtIdozito.IsEnabled = true;
    // Eredmény megjelenítése az ablak fejlécében.
    FeliratKiir();
}
```

### 3.2.10. Házi feladat

Tegyük lehetővé a felhasználó számára, hogy állítsa be a játékidő nagyságát. A játék közben ezen ne lehessen változtatni. Tehát a beállítás csak két játék közötti időben legyen lehetséges.

### 3.3. Képnézegető alkalmazás WPF alapú felülettel

Készítsen egy WPF képnézegető alkalmazást, ami a mellékelt ábrának megfelelően a bal oldali oszlopban (Grid) egy könyvtárban található jpg képek bélyegképeit jeleníti meg, a jobb oldali oszlopban az éppen kiválasztott kép nagyban látható.



103. ábra.

A két oszlop között egy elválasztó sáv legyen (GridSplitter), ami lehetővé teszi, hogy a felhasználó változtathassa az oszlopok szélességét.

#### 3.3.1. Megoldás

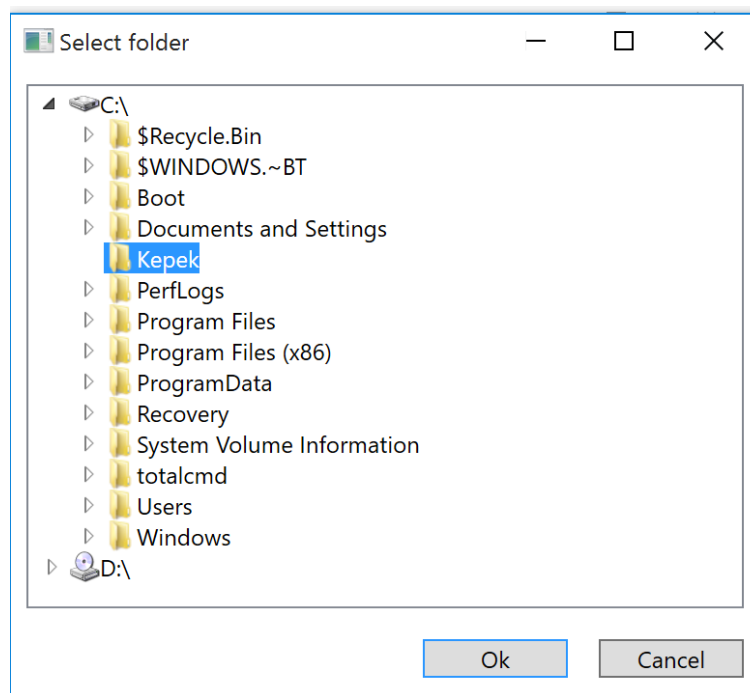
Készítsük egy WPF projektet Kepek néven. Az ablak állományának neve legyen wndFoablak.xaml és osztályának neve legyen wndFoablak.

#### 3.3.2. A felület elkészítése

Az ablakban két oszlopot alakítsunk ki: első a bélyegképek és a mappaválasztó nyomógomb, valamint az elválasztó vonal számára, míg a második a nagyméretű kép számára. Az első szélessége „150” legyen, míg a másodiké Auto.

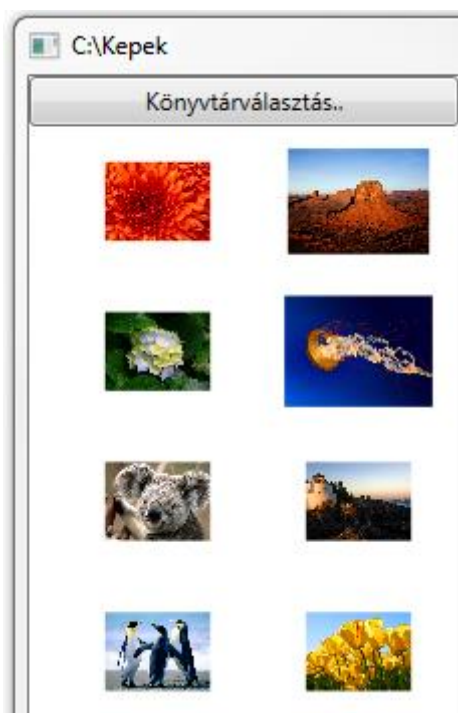
A baloldali oszlopban két sort hozunk létre egy Grid beépítésével. Az első sorban egy nyomógomb található, amin kattintva egy könyvtárválasztó párbeszédablak jelenik majd meg. A mappakiválasztó párbeszédablak nem beépített, az őt tartalmazó kódot (FolderPickerLib.dll) a `t:\info\Johanyák Csaba\Kepek\FolderPickerLib.dll` útvonalon érhetjük el, és be kell másolni a projekt könyvtárba, majd fel kell venni a projekt referenciái közé.





104. ábra.

A bélyegképeket egy ScrollViewer vezérlőre helyezett WrapPanel vezérlőn helyezük el, ez biztosítja, hogy szükség esetén jelenjen meg a görgető sáv, és annyi oszlopban jelenjenek meg, amennyinek a vízszintes megjelenítésére lehetőség van.



105. ábra.

Ha a bélyegképek felé visszük az egeret, akkor a képnek meg kell növekednie kissé egy animáció segítségével, majd az egér eltávolítását követően vissza kell zsugorodnia az eredeti méretére. Ehhez a megfelelő méretű helyet előre le kell foglalni, így a kép nem közvetlenül a WrapPanelre kerül, hanem egy keretre (Border), és a keretet tesszük a WrapPanelre. A keret



mérete nagyobb a bélyegkép méreténél, és az animáció során a bélyegkép ki fogja tölteni a keretet.

A jobb oldalon a nagyméretű kép és a kép teljes elérési útvonala+állományneve jelenik meg egymás alatt. Ehhez a jobb oldalon is egy Gridet építünk be, ami kétsoros lesz. Az első sor magassága 25 lesz.

### A felületet leíró XAML kód

```
<Window x:Class="Kepek.wndFoablak"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:Kepek"
  mc:Ignorable="d"
  Title="Válasszon könyvtárat!" Height="350" Width="525">
  <Grid Name="grRacs">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="150"/>
      <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <Grid Grid.Column="0" Margin="0,0,5,0">
      <Grid.RowDefinitions>
        <RowDefinition Height="25"/>
        <RowDefinition/>
      </Grid.RowDefinitions>
      <Button Name="btKonyvtarValaszto" Content="Könyvtárválasztás.."
        Click="btKonyvtarValaszto_Click"/>
      <ScrollViewer Name="swKepek" VerticalScrollBarVisibility="Auto"
        Grid.Row="1">
        <WrapPanel Name="wpKepek" HorizontalAlignment="Right"/>
      </ScrollViewer>
    </Grid>
  <GridSplitter ResizeDirection="Columns" Grid.Column="0" Width="5" />
  <Grid Grid.Column="1">
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition Height="25" />
    </Grid.RowDefinitions>
    <Image Name="imNagyKep" MinWidth="100" MinHeight="100" Grid.Row="0"
      Stretch="Uniform"/>
    <TextBlock Name="tbKepNev" Height="15" VerticalAlignment="Center"
      Grid.Row="1" Margin="5,0,0,0"/>
  </Grid>
</Grid>
</Window>
```

A feladathoz felhasználható mintaképek a *t:\info\Johanyák Csaba\Kepek\* könyvtárban találhatóak meg.

### 3.3.3. A feladatot megvalósító kód

#### Adattagok

```
/// <summary>
/// A bélyegkép vezérlő alap szélessége.
/// </summary>
private double kepSzelesseg;
/// <summary>
/// A bélyegkép vezérlő alap magassága.
```

```
/// </summary>
private double kepMagassag;
/// <summary>
/// Ennyi ideig tart az animáció.
/// </summary>
private TimeSpan tsAnimacioIdo;
/// <summary>
/// Az animáció során ennyivel nő a kép szélessége.
/// </summary>
private double dSz;
/// <summary>
/// Az animáció során ennyivel nő a kép magassága.
/// </summary>
private double dM;
```

## Konstruktor

A konstruktorban megadjuk a bélyegképet megjelenítő vezérlő méreteit. A feladat egyszerűsítése érdekében itt úgy dolgozunk, hogy előre tudjuk, hogy a képek 1024x766-os felbontásúak lesznek. Az animációhoz fél másodperces időtartamot adunk meg. Az aktuális könyvtár (amiből a képeket meg fogja jeleníteni a program) elérési útvonalát az ablak fejlécében tároljuk, illetve jelenítjük meg. Ennek kezdőértékét (C:\Kepek\)) is a konstruktorban adjuk meg. Végül beolvassuk a memóriába az adott könyvtárban levő képeket. Ez utóbbi feladatot egy külön metódussal oldjuk meg (KépeketBetölt()). A metódus vázát automatikusan generáltatjuk a Visual Studioval.

```
/// <summary>
/// A főablak konstruktora. Adattagok inicializálása és kezdőképek /// betöltése.
/// </summary>
public wndFoablak()
{
    InitializeComponent();
    //A kép vezérlő eredeti szélessége.
    kepSzelesseg = 70;
    //A kép vezérlő eredeti magassága. A felhasznált mintaképek mind //1024x766-os méretűek.
    kepMagassag = kepSzelesseg * 766 / 1024;
    //Az animáció során ennyivel nő a kép vezérlő szélessége.
    dSz = 30;
    //Az animáció során ennyivel nő a kép vezérlő magassága.
    dM = dSz * 766 / 1024;
    //Az animáció időigényének megadása.
    tsAnimacioIdo = TimeSpan.FromMilliseconds(500);
    //A kezdőkönyvtár(képeket tároló könyvtár) megadása.
    Title = @"C:\";
    //Beolvassuk a képeket a könyvtárból.
    KepeketBetolt();
}
```

## Képek betöltése

```
/// <summary>
/// Betölti a képeket a kiválasztott mappából, és kép vezérlők
/// formájában elhelyezi őket a WrapPanel-en.
/// </summary>
private void KepeketBetolt()
{
    //A képeket tartalmazó könyvtár objektum létrehozása.
    DirectoryInfo dI = new DirectoryInfo(Title);
```

```

//Töröljük a WrapPanel-en lévő vezérlők listáját.
wpKepek.Children.Clear();
try
{
    //Lekérdezzük a .jpg kiterjesztésű állományokat a
    //könyvtárból.
    FileInfo[] fI = dI.GetFiles("*.jpg");
    //Minden képet beolvasunk.
    foreach(FileInfo fajl in fI)
    {
        //A helyőrző létrehozása. Ez nagyobb kell legyen, //mint a kép vezérlő.
        //Amikor növeljük a kép vezérlő //méretét, a helyőrzőt fogja kitölteni.
        Border bdHelyorzo = new Border();
        bdHelyorzo.Width = kepSzelesseg + dSz;
        bdHelyorzo.Height = kepMagassag + dM;
        //Felvesszük a helyőrző a panelre.
        wpKepek.Children.Add(bdHelyorzo);
        //Létrehozunk egy kép objektumot, és betöltjük a //fájlból a képet.
        var imKep = new Image
        { //Kép forrás megadása.
            Source = new BitmapImage(new Uri(fajl.FullName, UriKind.Absolute)),
            Width = kepSzelesseg,
            Height = kepMagassag
        };
        //A kép a vezérlőn töltsse ki a rendelkezésre álló //helyet az eredeti
        //képarány megtartásával.
        imKep.Stretch = Stretch.Uniform;
        //A kép vezérlő a helyőrző közepére kerüljön.
        imKep.VerticalAlignment = VerticalAlignment.Center;
        imKep.HorizontalAlignment = HorizontalAlignment.Center;
        //Eseménykezelő rendelése az egérgomb lenyomásához.
        imKep.MouseDown += imKep_MouseDown;
        //Eseménykezelő rendelése az egér vezérlő fölé //érkezéséhez.
        imKep.MouseEnter += imKep_MouseEnter;
        //Eseménykezelő rendelése az egér vezérlő fölül távozásához.
        imKep.MouseLeave += imKep_MouseLeave;
        //Kép elhelyezése a helyőrzőben.
        bdHelyorzo.Child = imKep;
    }
}
catch(Exception e)
{
    MessageBox.Show(e.Message); //Hibaüzenet, ha nem sikerült valamelyik művelet.
}
if(wpKepek.Children.Count > 0)
{
    //Beállítjuk a legelső képet nagynak.
    KepBeallit((Image)((Border)wpKepek.Children[0]).Child);
}
}

```

### Az animációt megvalósító metódusok

```

/// <summary>
/// A felhasználó az egeret elmozgatta a képről.
/// </summary>
/// <param name="sender">A kép vezérlő objektum.</param>
/// <param name="e"></param>
private void imKep_MouseLeave(object sender, MouseEventArgs e)
{

```

```

var imKep = (Image)sender;
//A vízszintes méretváltoztatást leíró animáció objektum.
DoubleAnimation dA = new DoubleAnimation();
//Kezdőméret.
dA.From = kepSzelesseg + dSz;
//Végső méret.
dA.To = kepSzelesseg;
//Az animáció időtartama.
dA.Duration = new Duration(tsAnimacioIdo);
//A függőleges méretváltoztatást leíró animáció objektum.
DoubleAnimation dB = new DoubleAnimation();
//Kezdőméret.
dA.From = kepMagassag + dM;
//Végső méret.
dA.To = kepMagassag;
//Az animáció időtartama.
dB.Duration = new Duration(tsAnimacioIdo);
//A két animáció elindítása.
imKep.BeginAnimation(WidthProperty, dA);
imKep.BeginAnimation(HeightProperty, dB);
}
/// <summary>
/// A felhasználó az egeret a kép vezérlő fele mozgatja.
/// </summary>
/// <param name="sender">A kép vezérlő objektum.</param>
/// <param name="e"></param>
private void imKep_MouseEnter(object sender, MouseEventArgs e)
{
    //Az aktuális kép objektum.
    var imKep = (Image)sender;
    //A vízszintes méretváltoztatást leíró objektum.
    DoubleAnimation dA = new DoubleAnimation();
    //Kezdőméret.
    dA.From = kepSzelesseg;
    //Végső méret.
    dA.To = kepSzelesseg + dSz;
    //Az animáció időtartama.
    dA.Duration = new Duration(tsAnimacioIdo);
    //A függőleges méretváltoztatást leíró objektum.
    DoubleAnimation dB = new DoubleAnimation();
    //Kezdőméret.
    dB.From = kepMagassag;
    //Végső méret.
    dB.To = kepMagassag + dM;
    //Az animáció időtartama.
    dB.Duration = new Duration(tsAnimacioIdo);
    //A két animáció elindítása.
    imKep.BeginAnimation(WidthProperty, dA);
    imKep.BeginAnimation(HeightProperty, dB);
}

```

### A nagyméretű kép megjelenítése

```

/// <summary>
/// Beállítjuk a képet a nagy vezérlőben láthatónak.
/// </summary>
/// <param name="imKep"></param>
private void KepBeallit(Image imKep)
{
    //A kép forrása.

```

```

imNagyKep.Source = imKep.Source;
//A kép alatt megjelenítjük az állomány nevét.
tbKepNev.Text = imNagyKep.Source.ToString();
}

```

### A bélyegképen történő egérgomb lenyomás eseménykezelője

```

/// <summary>
/// Egérgomb lenyomása egy kép vezérlőn.
/// </summary>
/// <param name="sender">A kép vezérlő.</param>
/// <param name="e"></param>
private void imKep_MouseDown(object sender, MouseButtonEventArgs e)
{
    //Beállítjuk a képet a nagy vezérlőn láthatónak.
    KepBeallit((Image)sender);
}

```

### 3.3.4. Eseménykezelő a mappaválasztáshoz.

A mappaválasztó gombon történt kattintást követően létrehozunk egy példányt a könyvtárvalasztó párbeszédablakból (FolderPickerDialog), beállítjuk a kezdő könyvtárat az ablak fejlécében tárolt útvonalnak megfelelően, majd megjelenítjük a párbeszédablakot.

Amennyiben a felhasználó az OK gombbal zárja be az ablakot, akkor a kiválasztott útvonalat átmásoljuk az ablak fejlécébe, és beolvassuk a memóriába az adott mappában található képeket.

```

/// <summary>
/// A képeket tartalmazó könyvtár kiválasztása.
/// </summary>
/// <param name="sender">Az eseményt előidéző nyomógomb.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void btKonyvtarValaszto_Click(object sender, RoutedEventArgs e)
{
    //Mappaválasztó párbeszédablak objektum létrehozása.
    var dlg = new FolderPickerDialog();
    //Kezdkönyvtár beállítása
    dlg.InitialPath = Title;
    //Párbeszédablak megjelenítése.
    if (dlg.ShowDialog() == true)
    {
        //A mappa elérési útvonalának átmásolása az ablak //fejlécbe.
        Title = dlg.SelectedPath;
        //A képek betöltése a kiválasztott mappából, és //elhelyezésük kép vezérlők
        formájában a WrapPanel-en.
        KepekBetolt();
    }
}
}

```

### 3.3.5. Házi feladat

Alakítsa át úgy a programot, hogy a nagyméretű kép alatt csak a képállomány neve jelenjen meg az elérési útvonal és a file:/// felirat nélkül.

Helyezzen el egy nyomógombot a nagyméretű kép alatt „Vágólapra másol” felirattal, és készítse el hozzá az eseménykezelőt.

Ellenőrizze le, hogy az eseménykezelő által a vágólapra másolt kép beilleszthető-e egy Word dokumentumba.

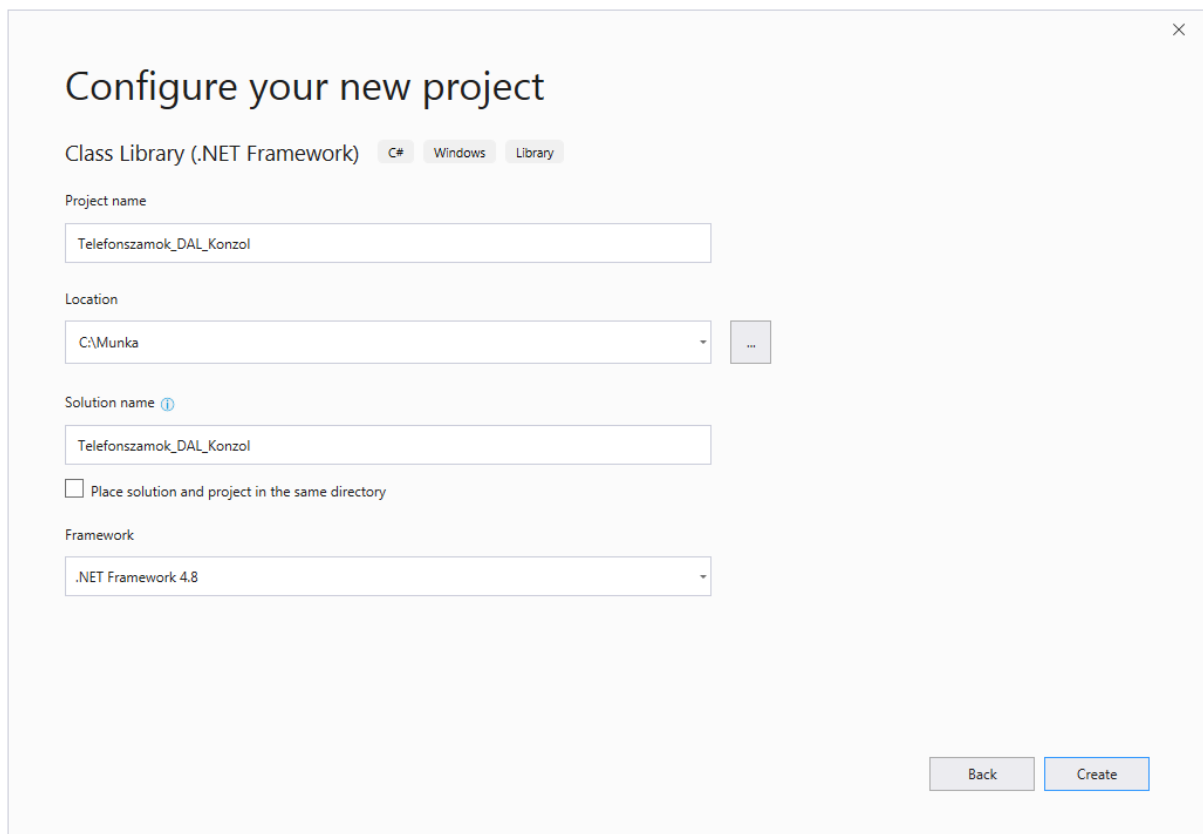
## 4. Adatbáziskezelés – Model First Entity Framework

### 4.1. Telefonszámok konzol alkalmazás

A gyakorlat célja Model-first megközelítéssel Entity-Framework modell létrehozása, majd ebből adatbázis generálása a Visual Studio-val együtt települő LocalDB adatbáziskezelő segítségével. Adatok felvitele közvetlenül, illetve programból, lekérdezések gyakorlása.

#### 4.1.1. Az Entity Framework modell és az adatbázis létrehozása

Az EF modellt egy külön osztálykönyvtár típusú projektben hozzuk létre. Ez képezi az alkalmazásunk adathozzáférési rétegét. Így könnyen megoldható hogy különböző típusú (konzol és WPF) alkalmazásokból is hozzáférhessünk az adatokhoz.



Configure your new project

Class Library (.NET Framework) C# Windows Library

Project name  
Telefonszamok\_DAL\_Konzol

Location  
C:\Munka

Solution name ⓘ  
Telefonszamok\_DAL\_Konzol

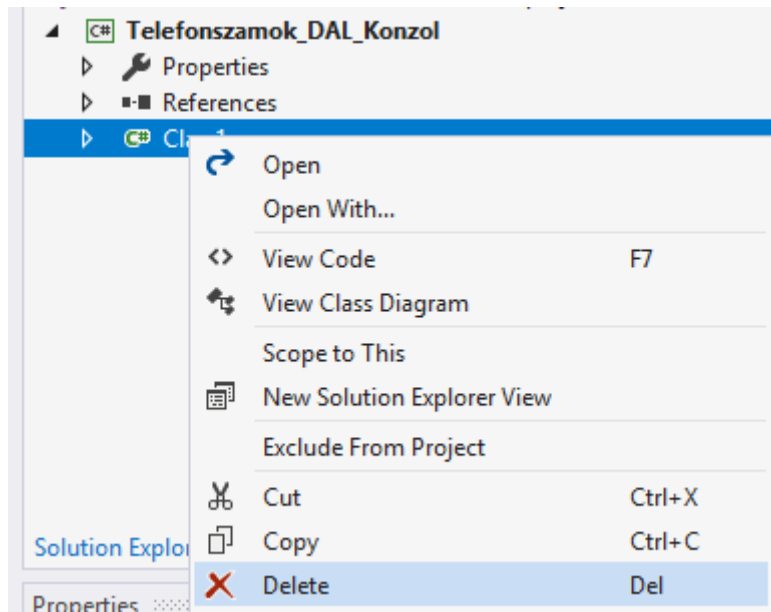
Place solution and project in the same directory

Framework  
.NET Framework 4.8

Back Create

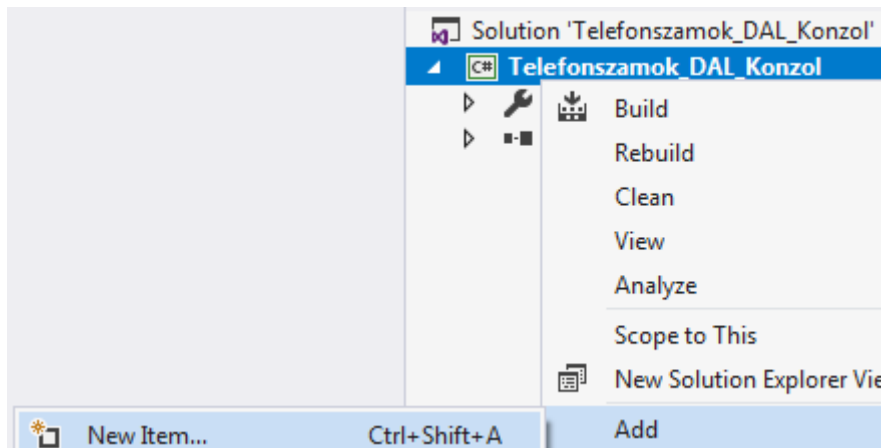
106. ábra.

Töröljük a Class1.cs állományt, mivel nem szükséges az EF modellhez.

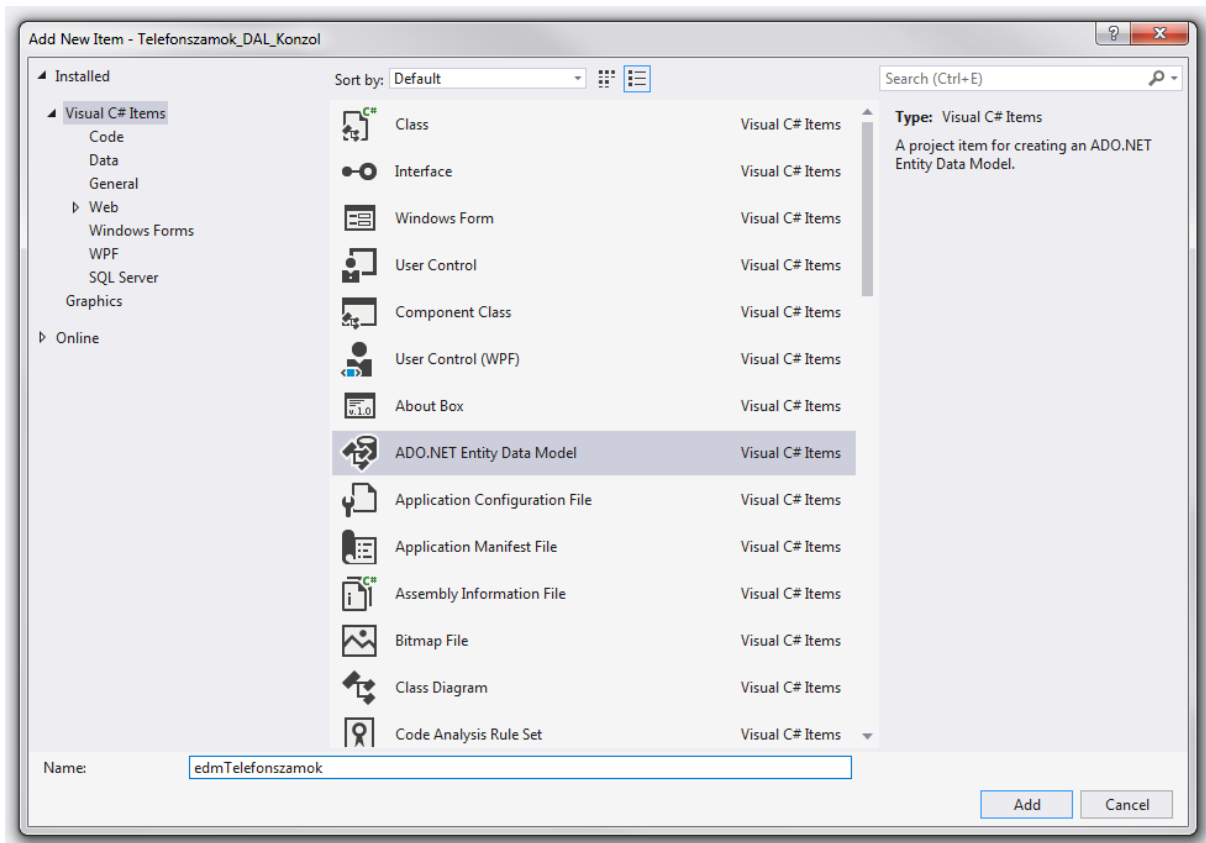


107. ábra.

A projekthez adunk egy Entity Data Model-t, ebben fogjuk létrehozni az entitás modellünket.

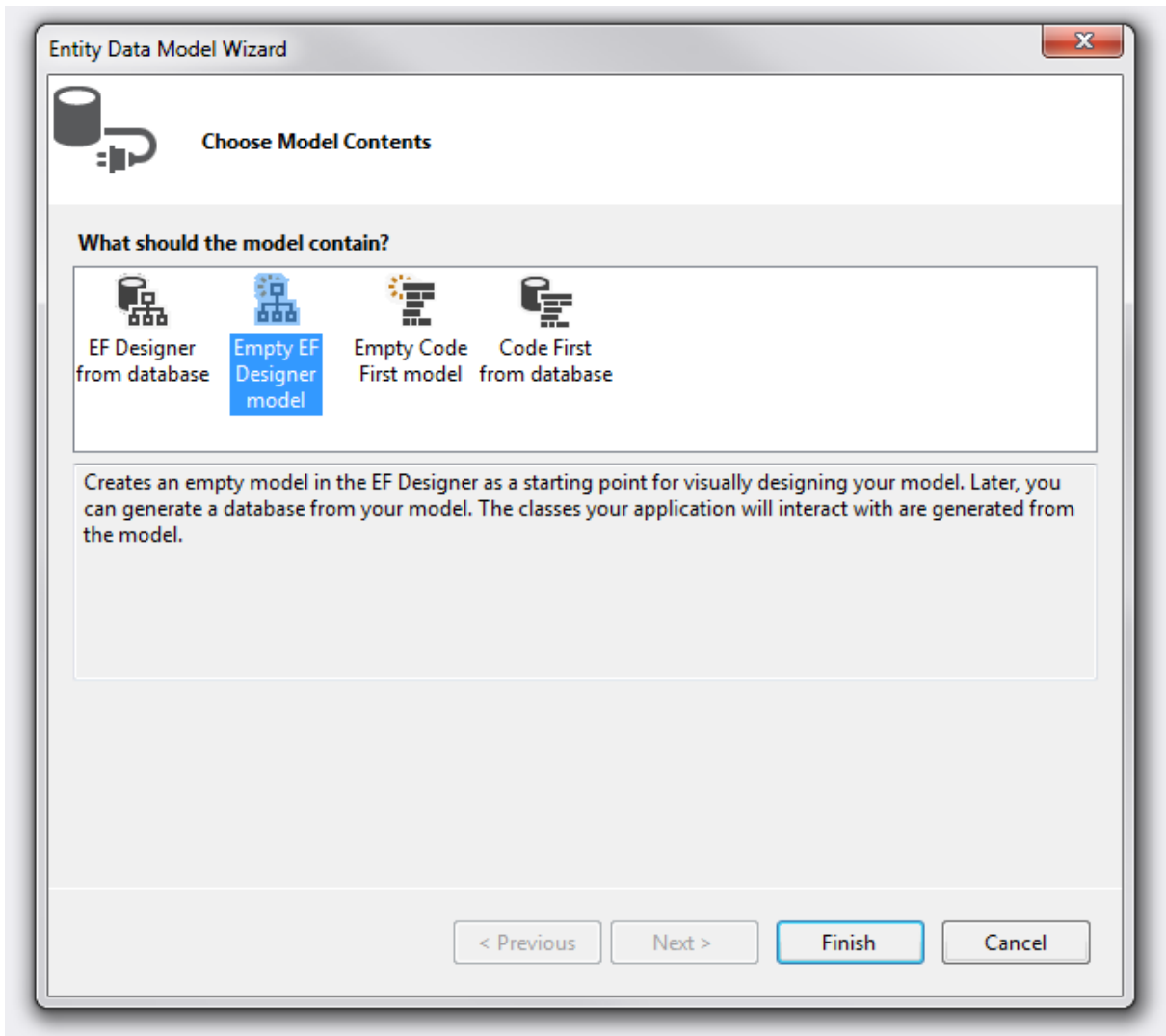


108. ábra.



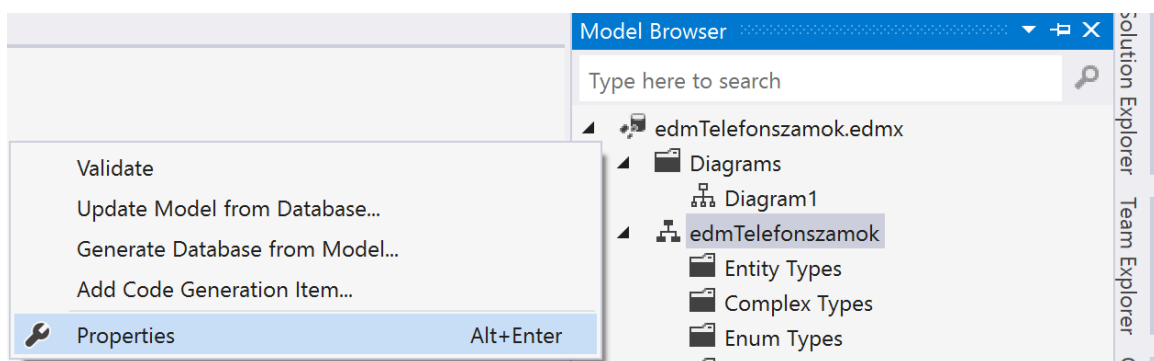
109. ábra.





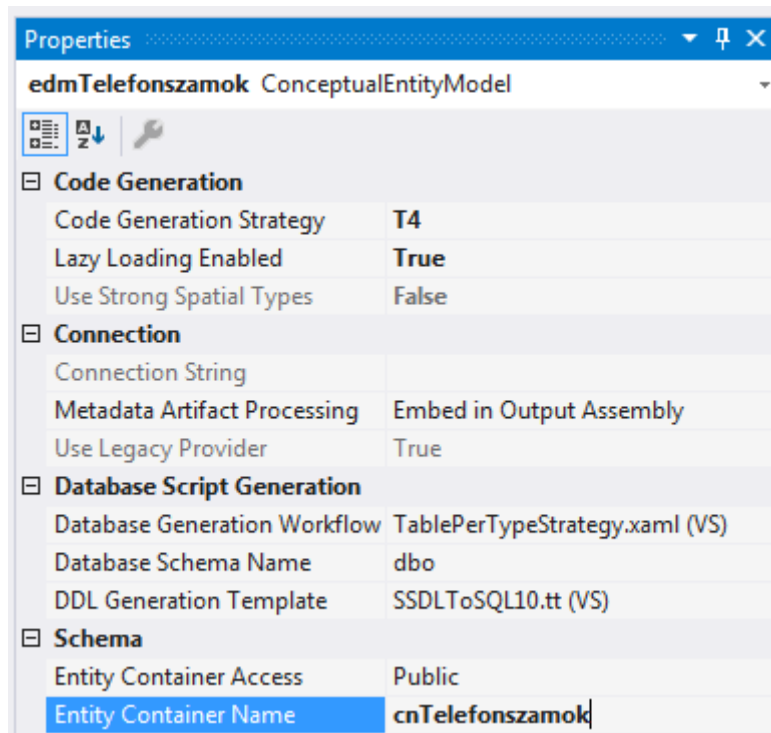
110. ábra.

Model Browser-ben megnyitjuk az edmTelefonszamok tulajdonságait.



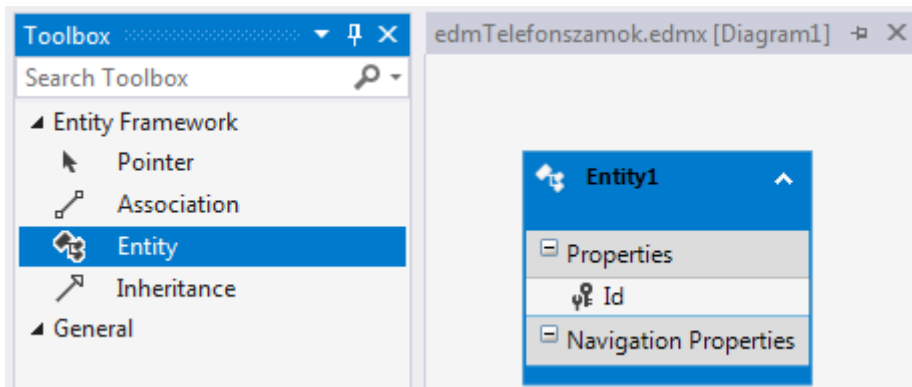
111. ábra.

Megadjuk az entitás konténer osztály nevét (cnTelefonszamok).



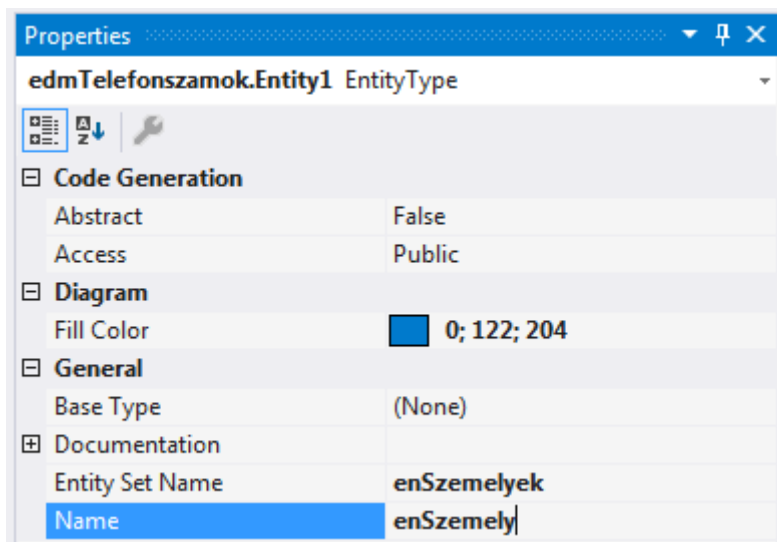
112. ábra.

Létrehozunk egy enSzemely nevű entitást.



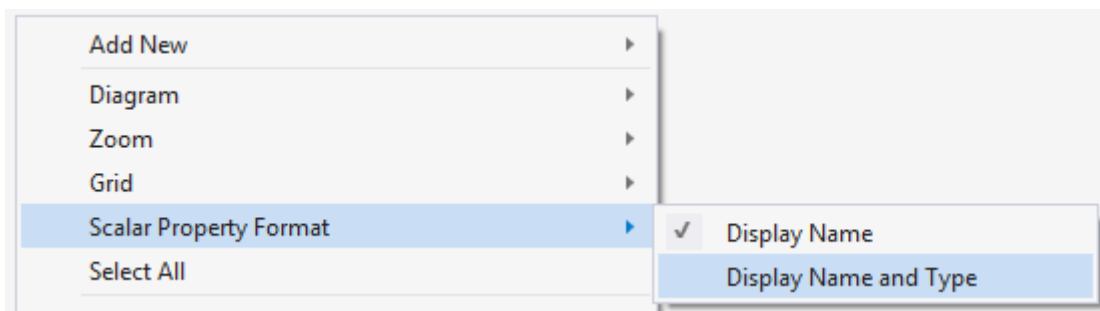
113. ábra.

Beállítjuk a nevét és az entitáshalmaz nevét.



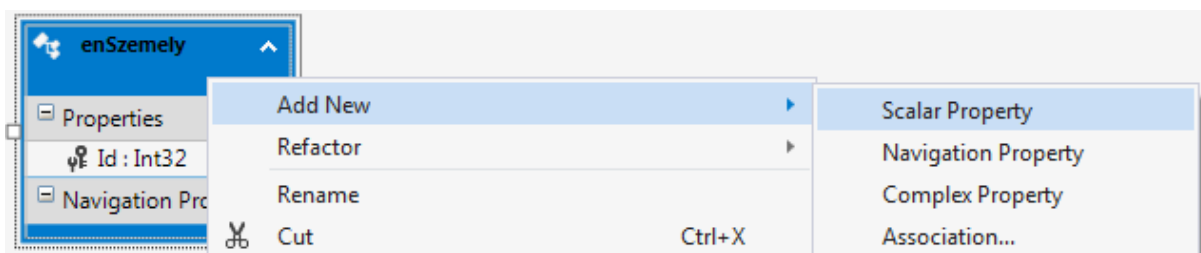
114. ábra.

Beállítjuk, hogy a tulajdonságok nevei is megjelenjenek.

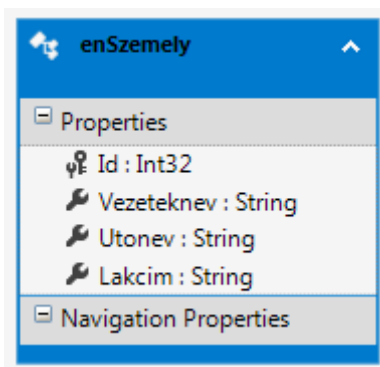


115. ábra.

Hozzáadunk újabb tulajdonságokat.

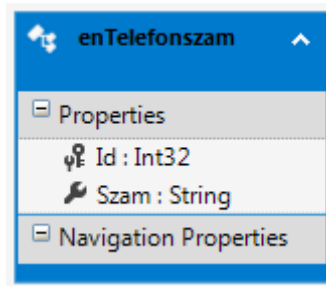


116. ábra.



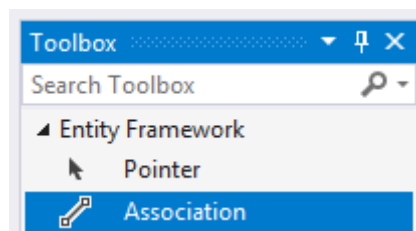
117. ábra.

Egy személyhez több telefonszám is tartozhat, ezért a számhoz létrehozunk egy külön entitást, aminek lesz egy Szam tulajdonsága a konkrét telefonszám tárolására.



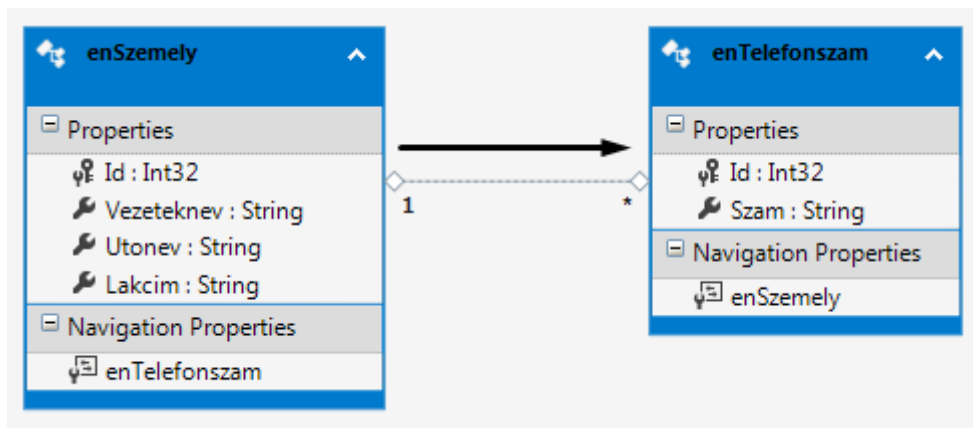
118. ábra.

Összekötjük őket egy Association kapcsolattal az enSzemely-től kiindulva.



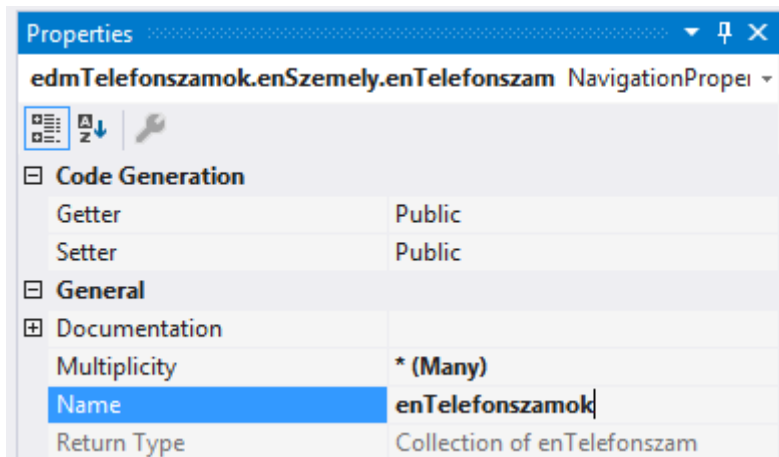
119. ábra.

Navigációs tulajdonságok jelennek meg mindkét entitásnál.



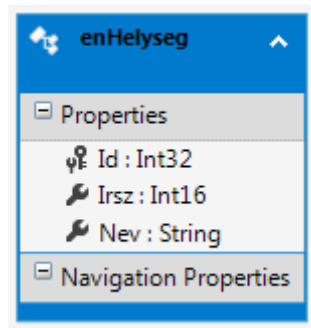
120. ábra.

Az enSzemely oldalon a tulajdonság egy gyűjtemény, ezért a nevét többes számba tesszük a Properties ablakban.



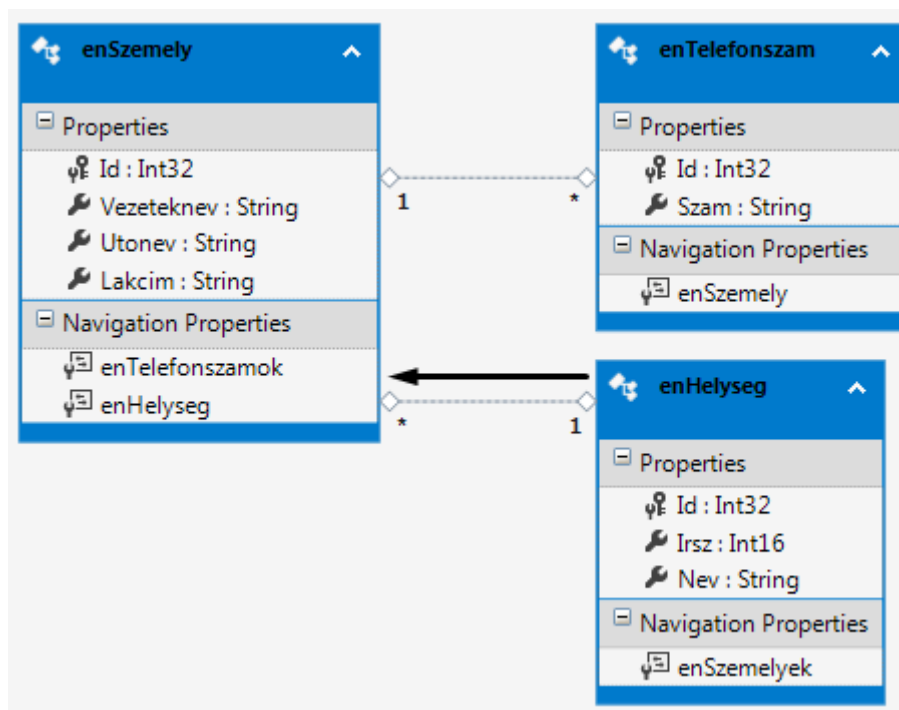
121. ábra.

Létrehozunk egy entitást enHelyseg néven.



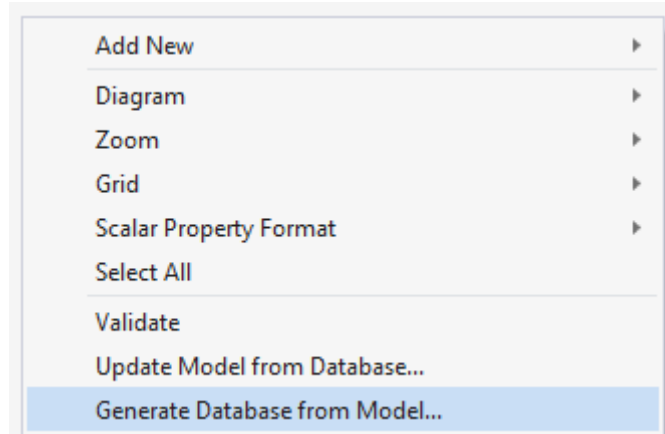
122. ábra.

Egy helységben több személy is lakhat. Ennek megfelelően az enHelyseg felől indítjuk a kapcsolatot, majd itt is többes számba tesszük a személyeket.

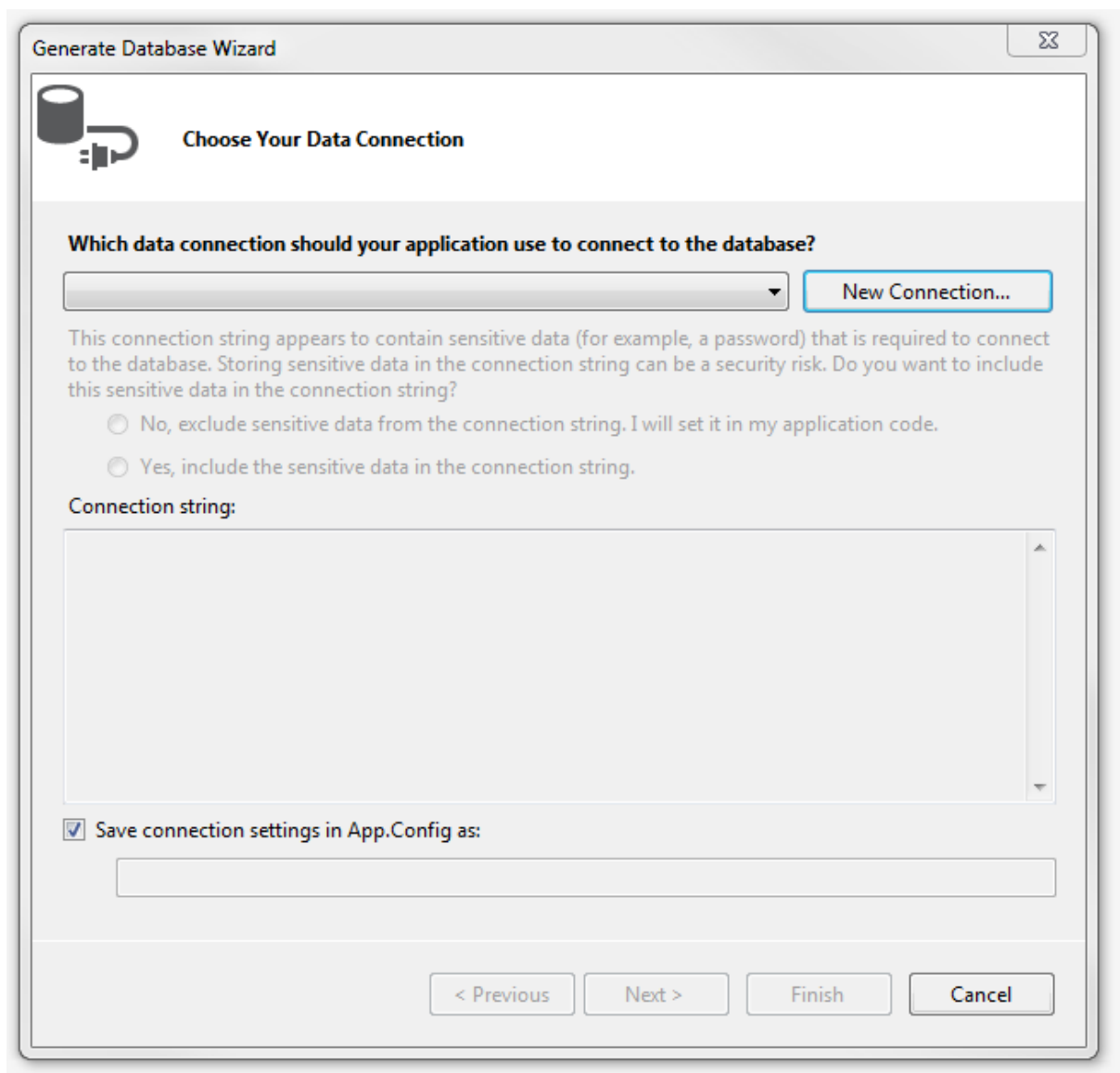


123. ábra.

Legeneráljuk az adatbázist.

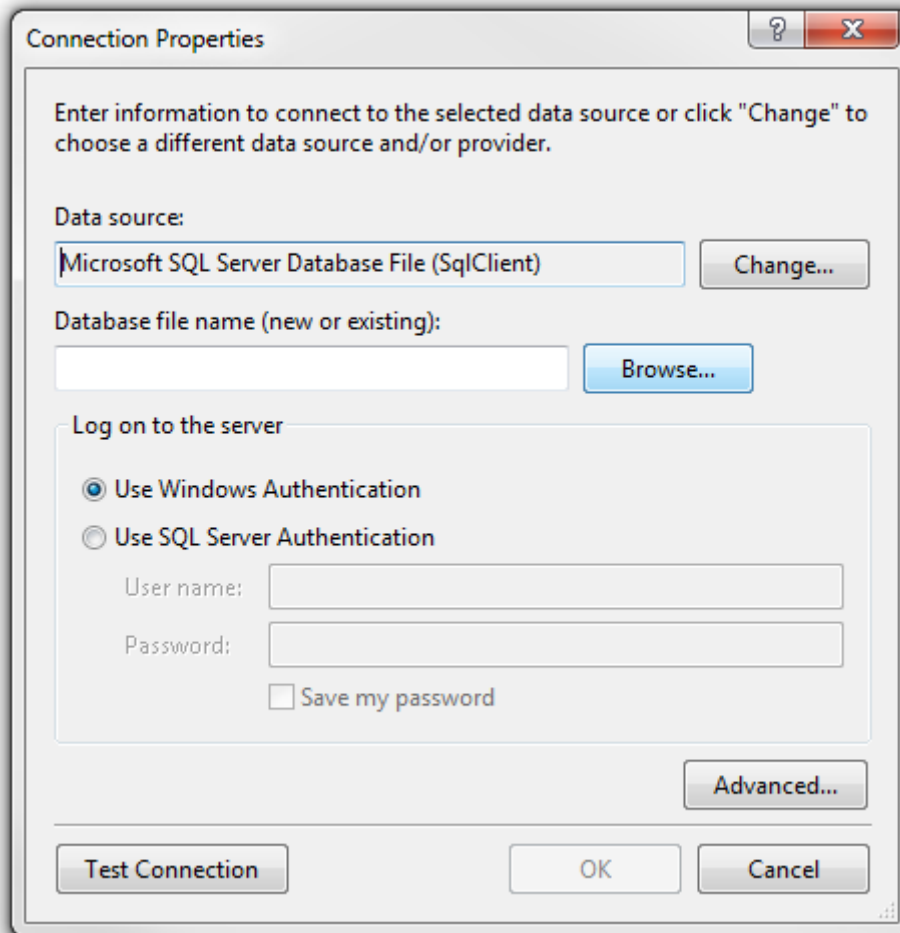


124. ábra.



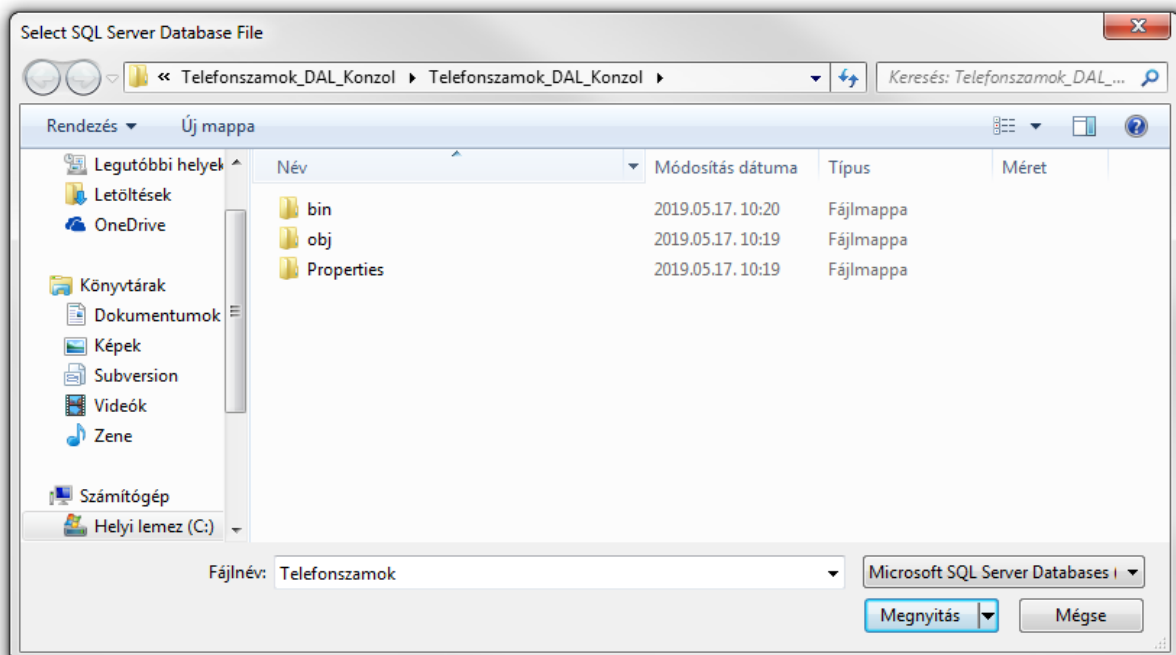
125. ábra.

Az adatokat egy adatbázis állományban fogjuk tárolni a projekt könyvtárban.

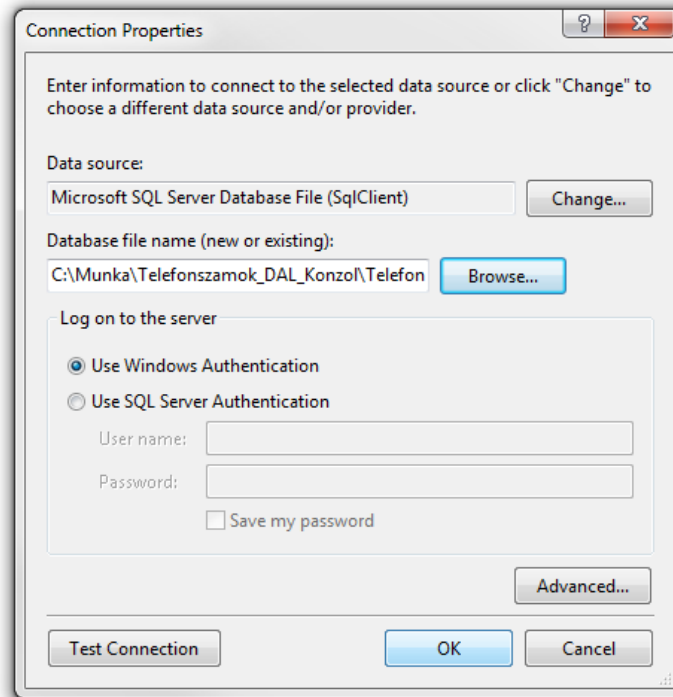


126. ábra.

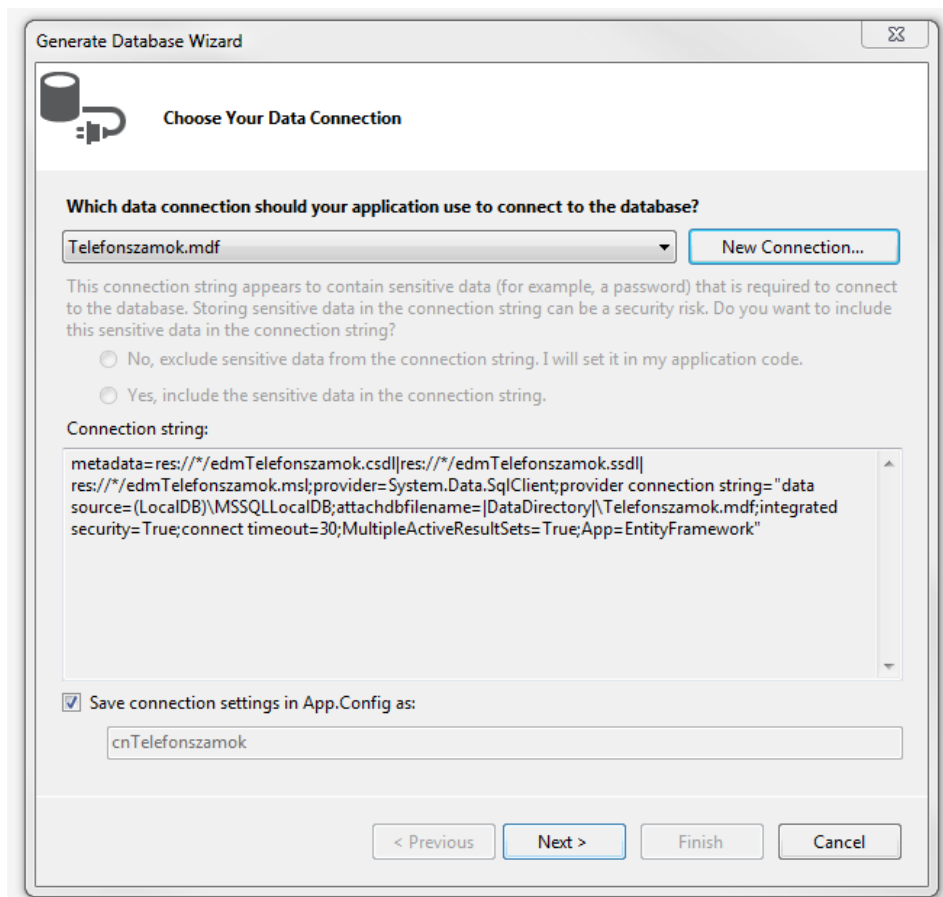
Kiválasztjuk a projekt könyvtárát, majd megadjuk az állománynevet.



127. ábra.

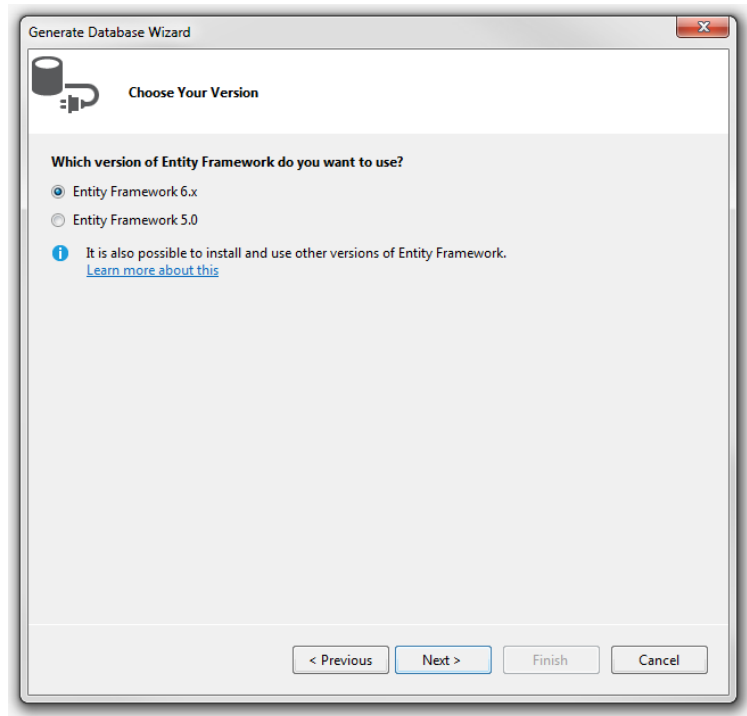


128. ábra.

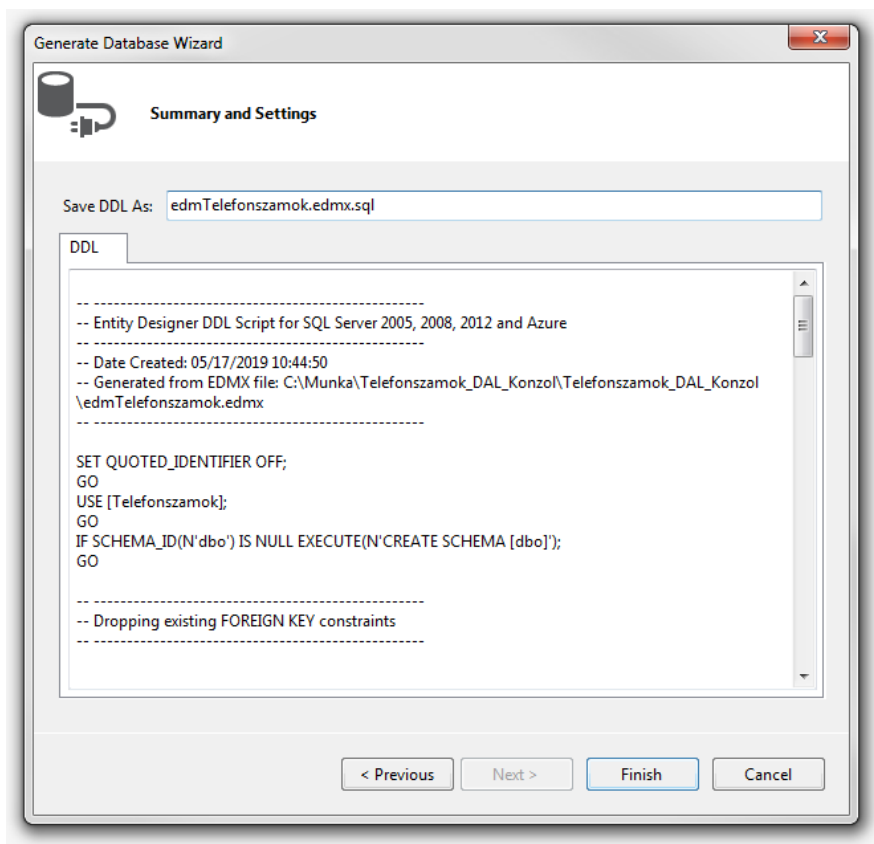


129. ábra.



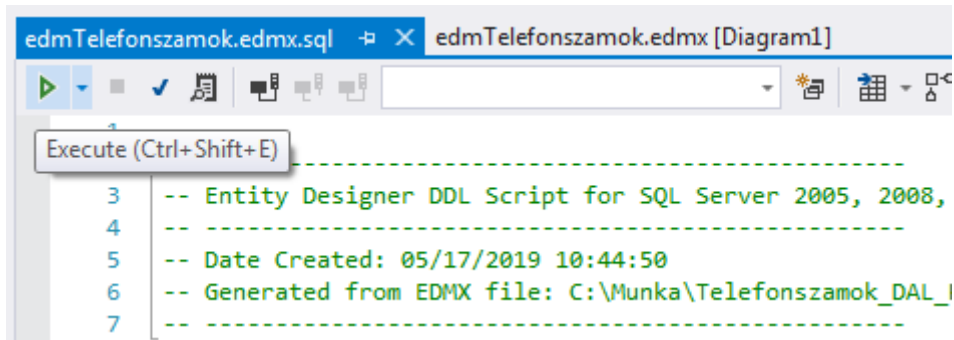


130. ábra.

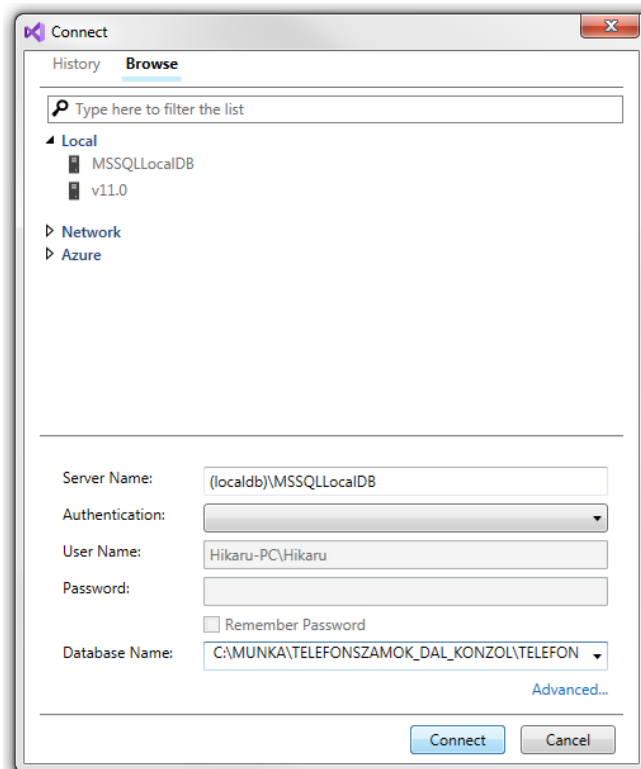


131. ábra.

Futtatjuk a szkriptet.

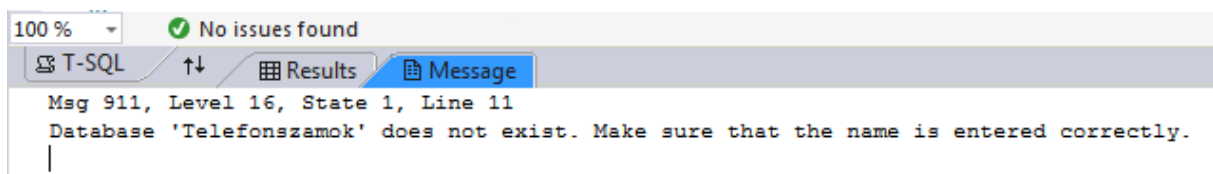


132. ábra.



133. ábra.

Hibaüzenet:



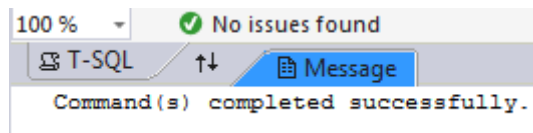
134. ábra.

Megoldásként megadjuk az adatállomány elérési útvonalát az edmTelefonSzamok.edmx.sql állományban a USE kulcsszó után (a nagybetű nem kötelező).

```
SET QUOTED_IDENTIFIER OFF;  
GO
```

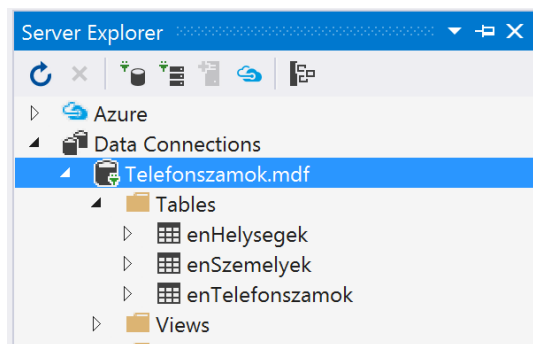
```
USE [C:\MUNKA\TELEFONSZAMOK_DAL_KONZOL\TELEFONSZAMOK_DAL_KONZOL\TELEFONSZAMOK.MDF];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');
GO
```

Majd újból indítjuk a szkriptet. Ekkor már sikeres a végrehajtás.



135. ábra.

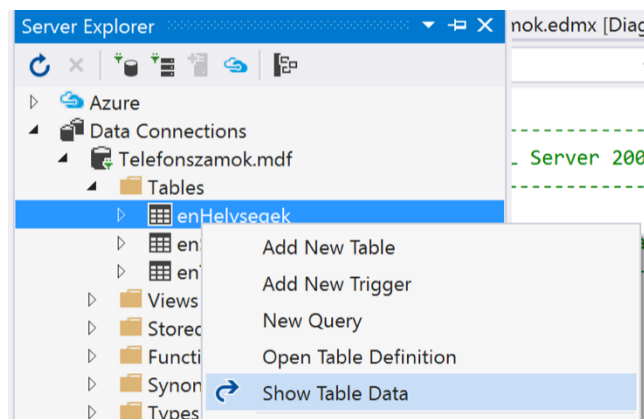
A Server Explorerben leellenőrizhetjük az adatbázis meglétét.



136. ábra.

#### 4.1.2. Adatfelvitel közvetlenül az adattáblákba

Vigyünk fel adatokat az adatbázisba a Visual Studio segítségével.



137. ábra.

Helységadatok: Csak a 2. és 3. oszlopba kell adatokat írni, az 1. oszlop automatikusan töltődik ki.

Id	Irsz	Nev
1	6000	Kecskemét
2	2038	Sóskút
3	2039	Pusztazámor
4	2040	Budaörs
5	2045	Törökbálint
▶*	NULL	NULL

138. ábra.

Személyek:

Id	Vezeteknev	Utonev	Lakcim	enHelyseg_Id
1	Senki	Alfonz	Kis út 1.	1
2	Gipsz	Jakab	Malom köz 1.	1
3	Erős	Áron	Alma rét 3.	3
4	Olvasó	Jolán	Fűzes út 2.	4
▶*	NULL	NULL	NULL	NULL

139. ábra.

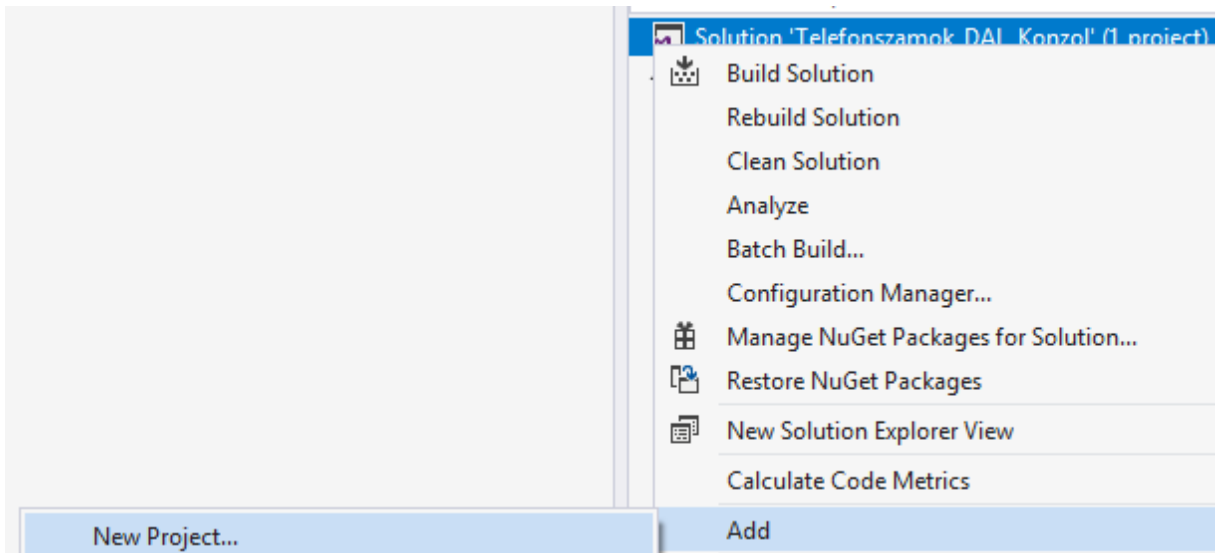
Telefonszámok

Id	Szam	enSzemely_Id
1	+36-111-111	4
2	+36-222-222	4
3	+36-333-333	3
4	+36-444-444	1
▶*	NULL	NULL

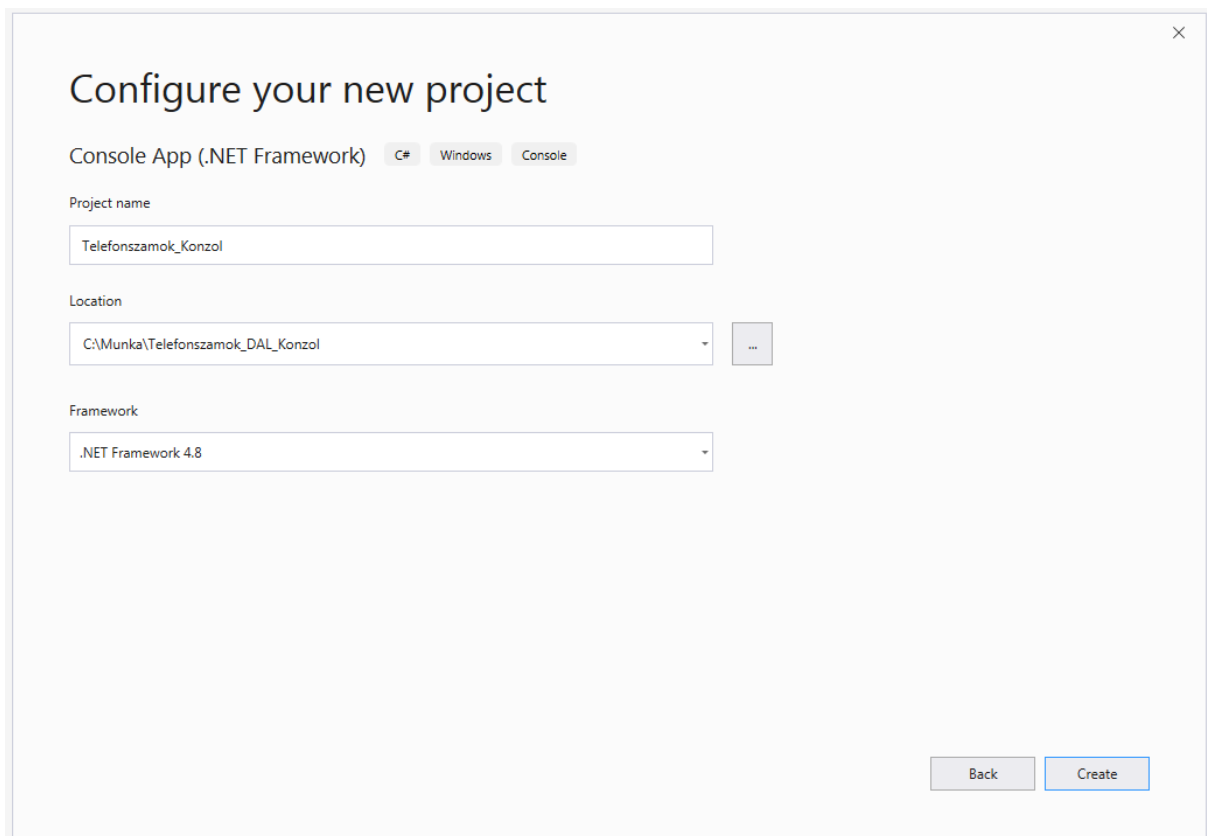
140. ábra.

### 4.1.3. Adatfelvitel programból

Hozzunk létre egy konzolalkalmazás projektet a megoldáson belül.

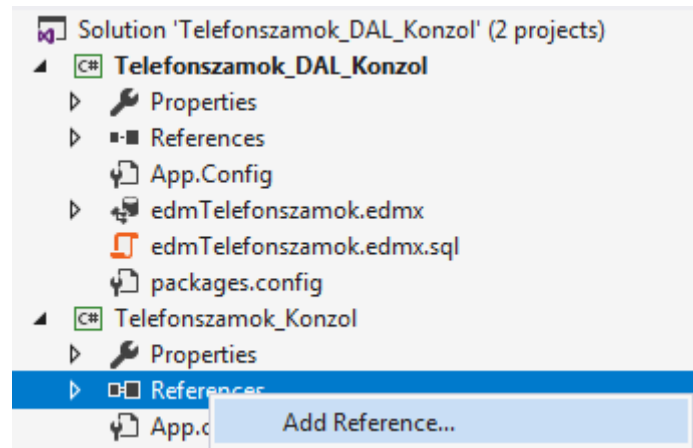


141. ábra.

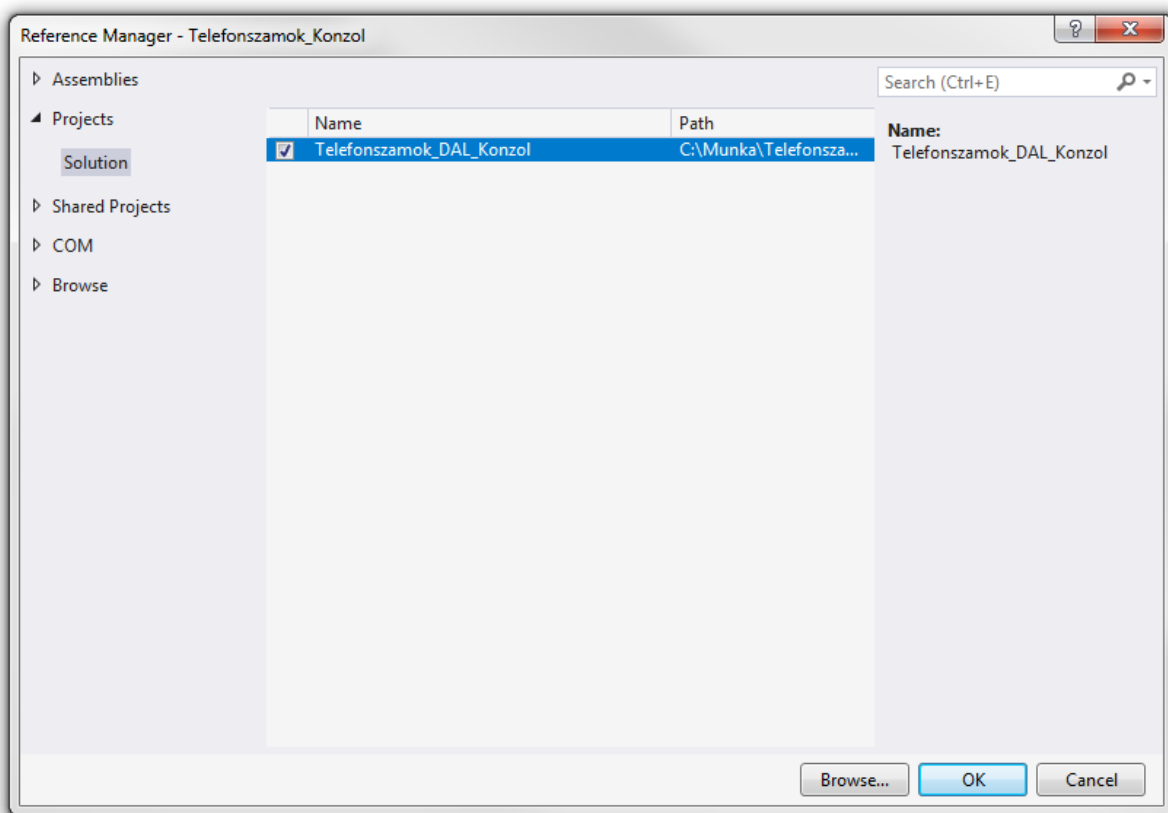


142. ábra.

Vegyük fel a referenciák közé az osztálykönyvtár projektünket.

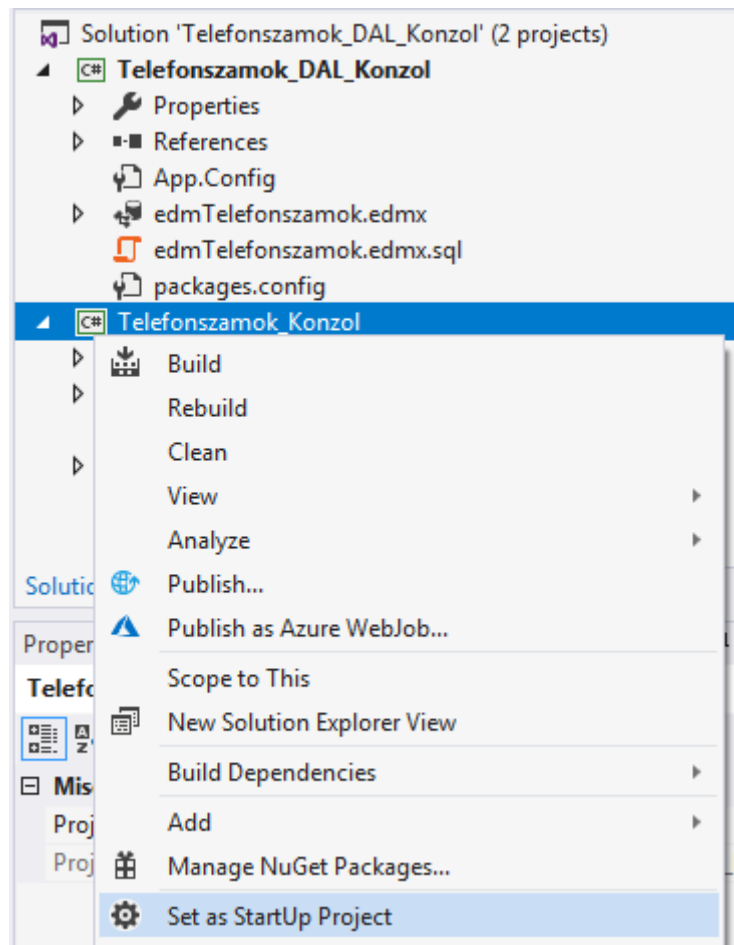


143. ábra.



144. ábra.

Állítsuk be indító projektnek a konzol projektet.



145. ábra.

Állítsuk be a DAL projekt névtérét felhasznált névtérként a konzol alkalmazásban.

```
using Telefonaszamok_DAL_Konzol;
```

Létrehozunk egy adattagot az entitáskonténer számára, majd felvesszük a referenciák közé az Entity Framework-öt. Tegyük az adattagot statikussá.

```
static cnTelefonaszamok cnTelefonaszamok;
```

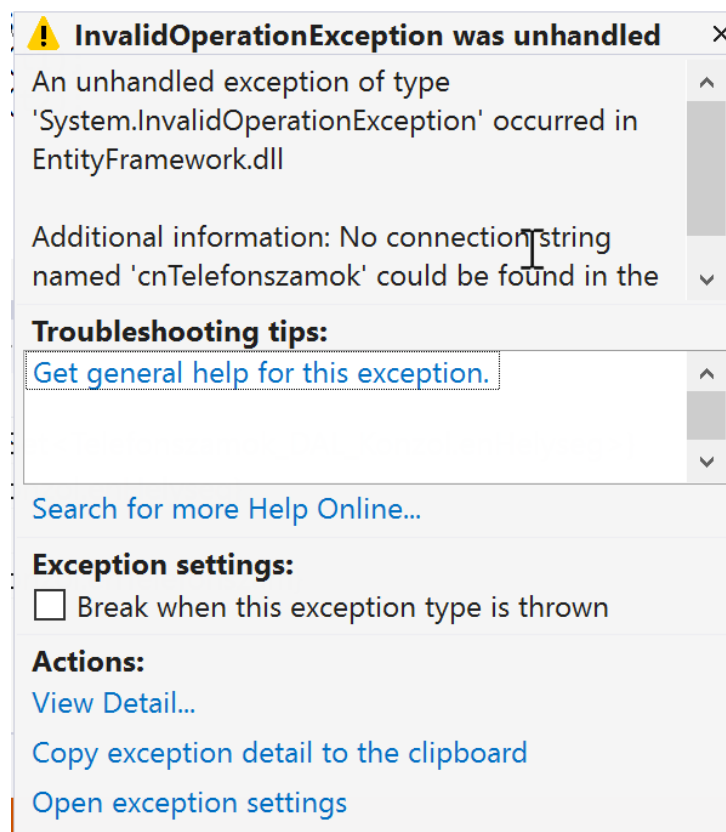
Hozzunk létre egy konténer objektumot. Készítsünk egy statikus metódust adatfelvitel céljára.

```
static void Main(string[] args)
{
    cnTelefonaszamok = new cnTelefonaszamok();
    Adatfelvitel();
}

private static void Adatfelvitel()
{
    var h = new enHelyseg{ Irsz = 2090, Nev = "Remeteszőlős"};
    var sz = new enSzemely {
        Vezeteknev = "Argon",
        Utonev = "Géze",
        Lakcim = "Ordas Köz 6.",
        enHelyseg = h
    };
    h.enSzemelyek.Add(sz);
}
```

```
var t1 = new enTelefonszam
{
    Szam = "+36-555-555", enSzemely = sz
};
var t2 = new enTelefonszam
{
    Szam = "+36-666-666", enSzemely = sz
};
sz.enTelefonszamok.Add(t1);
sz.enTelefonszamok.Add(t2);
cnTelefonszamok.enHelysegek.Add(h);
cnTelefonszamok.enSzemelyek.Add(sz);
cnTelefonszamok.enTelefonszamok.Add(t1);
cnTelefonszamok.enTelefonszamok.Add(t2);
cnTelefonszamok.SaveChanges();
}
```

Futtassuk le a programot, majd ellenőrizzük le, hogy az adatbázisba kerültek-e az adatok.



146. ábra.

Hiba: nincs kapcsolatunk az adatbázishoz. Megoldás: másoljuk át a Telefonszamok\_DAL\_Konzol osztálykönyvtár projekt App.Config állományából a ConnectionStrings meghatározását a konzolalkalmazás projektjébe.

Indítsuk el a konzolalkalmazást. Amennyiben hibaüzenet nélkül lefut, akkor tegyük megjegyzésbe az Adatfelvitel(); utasítást, mert erre már nem lesz szükségünk a továbbiakban, az adatok bekerültek az adatbázisba. Ellenőrizzük le a Server Explorer segítségével, hogy sikerült-e az adatfelvitel.



Id	Irsz	Nev
1	6000	Kecskemét
3	2038	Sóskút
4	2039	Pusztazámor
5	2040	Budaörs
6	2045	Törökbálint
7	2090	Remeteszőlős
NULL	NULL	NULL

147. ábra.

Id	Vezeteknev	Utonev	Lakcim	enHelyseg_Id
1	Senki	Alfonz	Kis út 1.	1
2	Gipsz	Jakab	Malom köz 1.	1
3	Erős	Áron	Alma rét 3.	3
4	Olvasó	Jolán	Füzes út 2.	4
5	Argon	Géza	Ordas Köz 6.	7
NULL	NULL	NULL	NULL	NULL

148. ábra.

Id	Szam	enSzemely_Id
1	+36-111-111	4
2	+36-222-222	4
4	+36-333-333	3
5	+36-444-444	1
7	+36-555-555	5
8	+36-666-666	5
NULL	NULL	NULL

149. ábra.

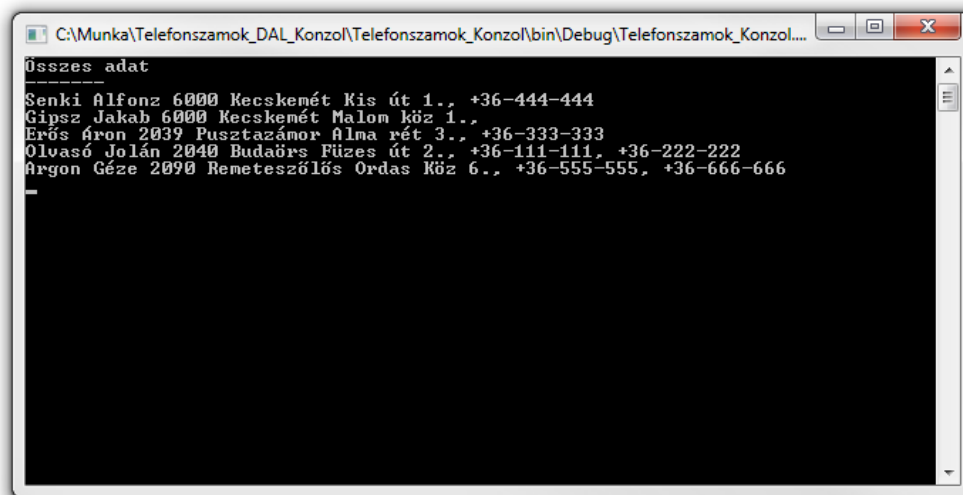
#### 4.1.4. Lekérdezések programból

Készítsünk egy lekérdező metódust, ami a lekérdezés eredményét kiírja a konzolra. A metódusban vegyük sorra a személyeket, és írassuk ki az adataikat. A telefonszámok egymástól vesszővel elválasztva jelenjenek meg. Hívjuk meg a Lekerdez() metódust a Main() metódusból.

```
private static void Lekerdez()
```

```
{
    Console.WriteLine("Összes adat\r\n-----");
    foreach(var x in cnTelefonszamok.enSzemelyek)
    {
        var s = x.Vezeteknev + " " + x.Utonev + " " + x.enHelyseg.Irsz + " " +
        x.enHelyseg.Nev + " " + x.Lakcim + ", ";
        foreach(var y in x.enTelefonszamok)
        {
            s += y.Szam;
            if (y != x.enTelefonszamok.Last())
                s += ", ";
        }
        Console.WriteLine(s);
    }
}
```

Futtassuk le a programot.



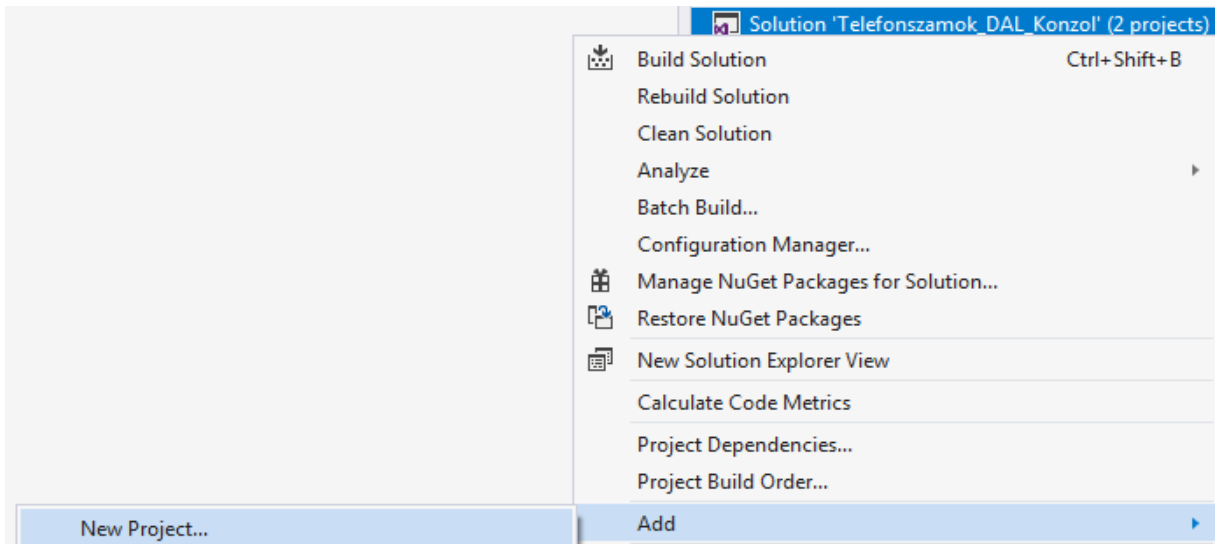
150. ábra.

## 4.2. Telefonszámok WPF alkalmazás

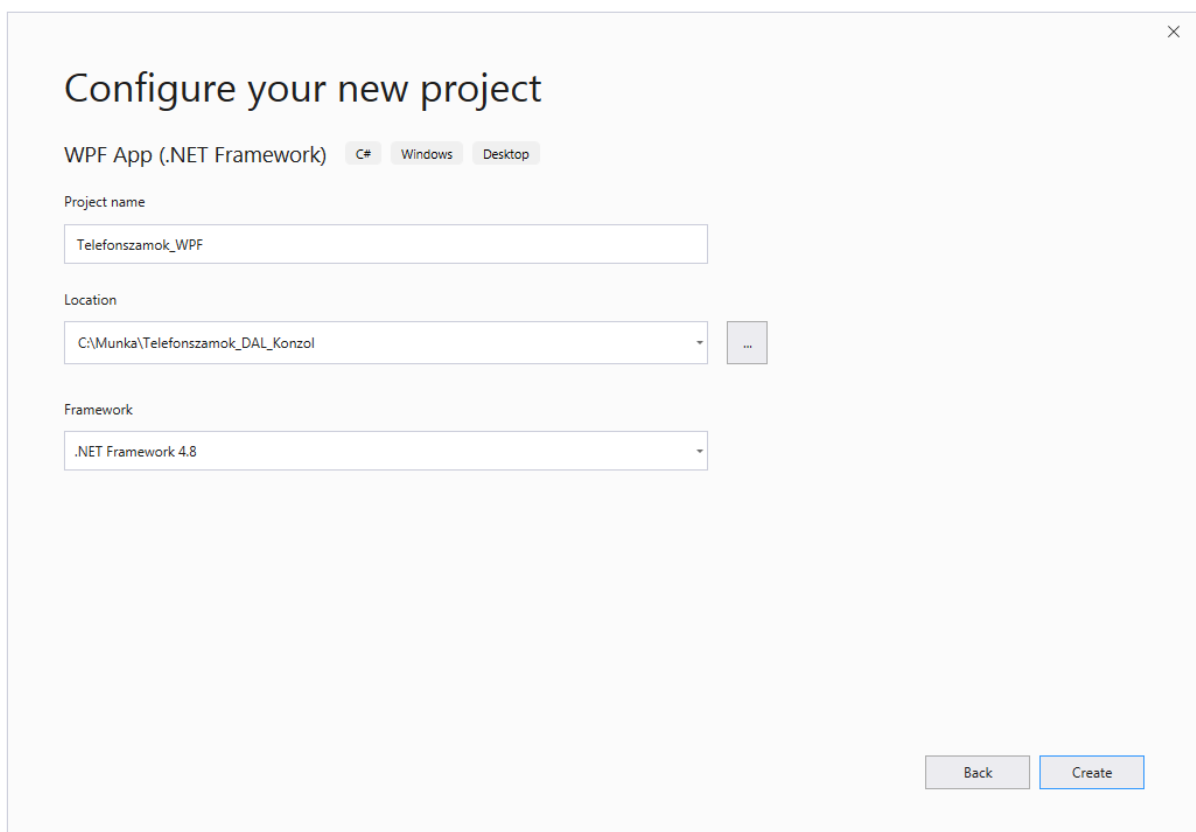
A gyakorlat célja az, hogy a korábban létrehozott Telefonszám kezelő alkalmazást kiegészítsük egy WPF típusú felülettel.

### 4.2.1. Projekt és alapbeállítások

Töltsük le és nyissuk meg a kiinduló megoldást (Solution-t), ami a korábban létrehozott adathozzáférési réteg (Telefonszamok\_DAL\_Konzol) és konzol alkalmazás (Telefonszamok\_Konzol) projekteket tartalmazza. Hozzunk létre egy új WPF projektet (Telefonszamok\_WPF) a megoldáson belül.

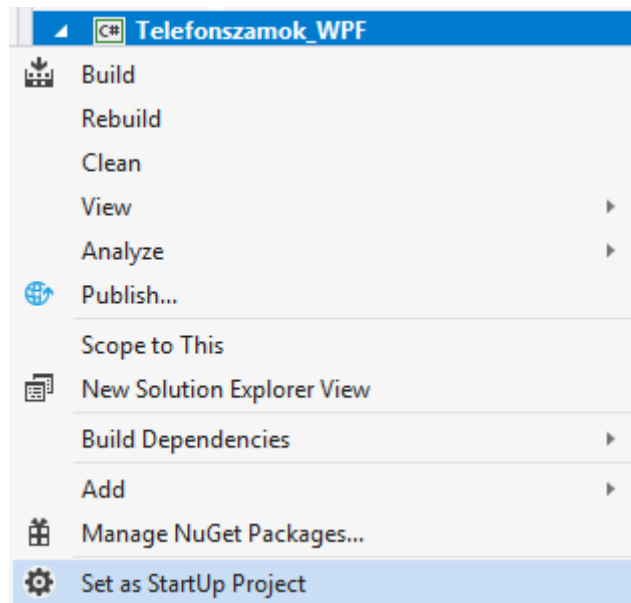


151. ábra.



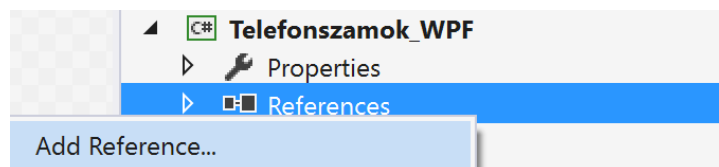
152. ábra.

Állítsuk be indító projektként az újonnan létrehozott projektet.

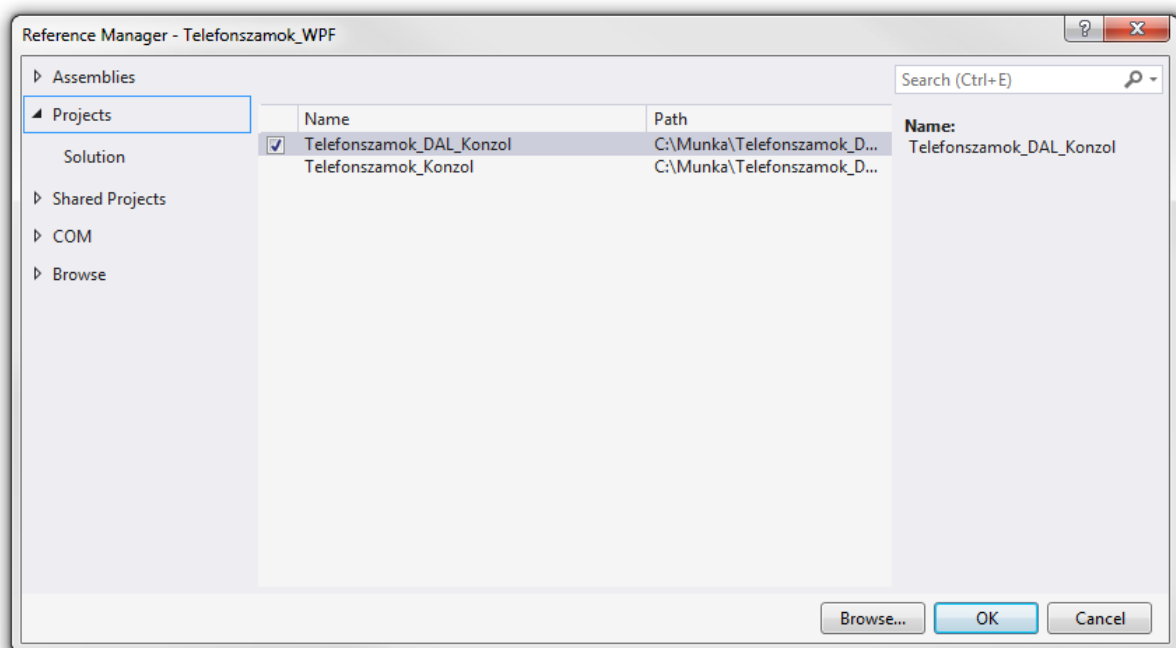


153. ábra.

A referenciák között állítsuk be a Telefonsszamok\_DAL\_Konzol projektet annak érdekében, hogy felhasználhassuk a későbbiekben az ott létrehozott entitás modellt.



154. ábra.



155. ábra.

Állítsuk be a DAL\_Konzol projekt névterét felhasznált névtérként az ablak C# kódjában (MainWindow.xaml.cs).

```
using Telefonszamok_DAL_Konzol;
```

Átmásoljuk a DAL\_Konzol project App.config állományából a ConnectionString-et az aktuális projektbe.

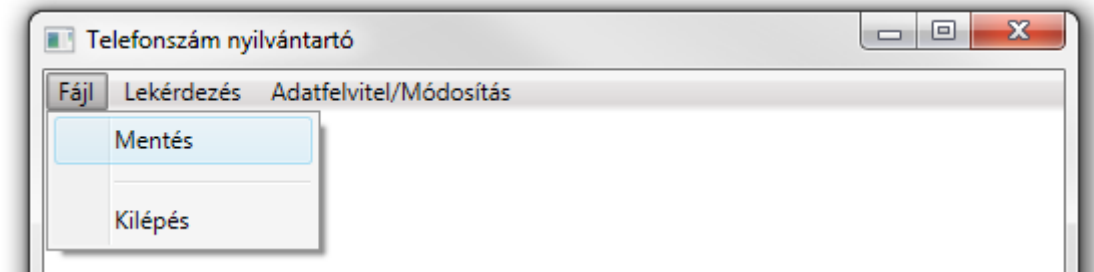
Létrehozunk egy adattagot az entitáskonténer számára, majd felvesszük a referenciák közé az Entity Framework-öt és létrehozzuk a konténer objektumát a főablak konstruktorában.

```
namespace Telefonszamok_WPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private cnTelefonszamok cnTelefonszamok;
        public MainWindow()
        {
            InitializeComponent();
            cnTelefonszamok = new cnTelefonszamok();
        }
    }
}
```

#### 4.2.2. A felület elkészítése

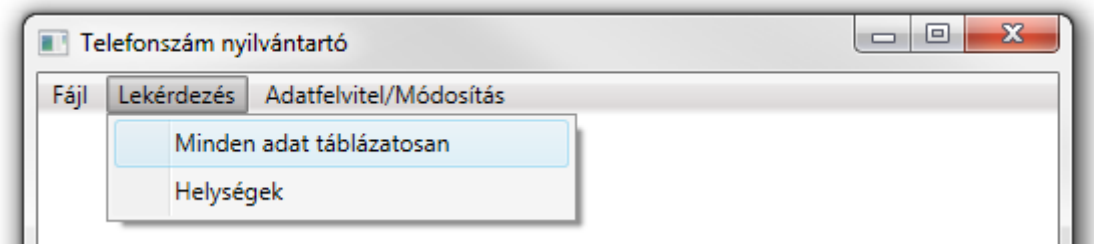
Az alkalmazás felületét a megkövetelt funkcionalitás határozza meg. A következőket várjuk el.:

1. Legyen képes a módosított adatok mentésére



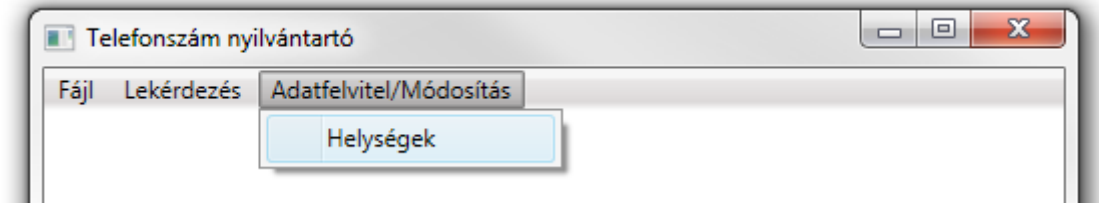
156. ábra.

2. A tárolt adatok legyenek lekérdezhetőek egy minden információt tartalmazó táblázatban.



157. ábra.

3. A helységekre vonatkozó adatok legyenek lekérdezhetőek egy táblázatban
4. A helységadatok legyenek módosíthatóak.



158. ábra.

A fenti igények megvalósításához StackPanel rétegmenedzsert használunk, amire egy menüt egy DataGrid és egy Grid komponenst fogunk elhelyezni. A Grid segítségével alakítjuk ki a helységadatok megjelenítéséhez szükséges űrlapot. Kezdetben sem a DataGrid, sem a Grid nem látható. Az ablak XAML kódja az alábbi:

```
<Window x:Class="Telefonszamok_WPF.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Telefonszamok_WPF"
mc:Ignorable="d"
Title="Telefonszám nyilvántartó" Height="350" Width="525">
<StackPanel>
  <!-- Menü -->
  <Menu>
    <MenuItem x:Name="miFajl" Header="Fájl" >
    <MenuItem x:Name="miMentes" Header="Mentés" Click="miMentes_Click"/>
    <Separator/>
    <MenuItem x:Name="miKilepes" Header="Kilépés" Click="miKilepes_Click"/>
  </Menu>
  <MenuItem x:Name="miLekerdezes" Header="Lekérdezés">
    <MenuItem x:Name="miMindenAdat" Header="Minden adat táblázatosan"
Click="miMindenAdat_Click" />
    <MenuItem x:Name="miHelysegek" Header="Helységek" Click="miHelysegek_Click"
/>
  </MenuItem>
  <MenuItem x:Name="miAdatFelvitelModositas" Header="Adatfelvitel/Módosítás">
    <MenuItem x:Name="miHelysegekAM" Header="Helységek"
Click="miHelysegekAM_Click"/>
  </MenuItem>
</Menu>
  <!-- Adatrács -->
  <DataGrid x:Name="dgAdatracs" ItemsSource="{Binding}" Visibility="Hidden"/>
  <!-- Helységadatok megjelenítése és módosítása -->
  <Grid x:Name="grHelyseg" Visibility="Hidden" Margin="0,10,0,0"
DataContext="{Binding}">
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Label Content="Keresés irányítószám alapján" Grid.Row="0" Grid.Column="0"/>

```

```

<ComboBox x:Name="cbIrsz" ItemsSource="{Binding}" DisplayMemberPath="Irsz"
Grid.Row="0" Grid.Column="1" Margin="5"
SelectionChanged="cbIrsz_SelectionChanged"
IsSynchronizedWithCurrentItem="True"/>
<Label Content="Keresés helységnev alapján" Grid.Row="1" Grid.Column="0" />
<ComboBox x:Name="cbHelysegnev" ItemsSource="{Binding}"
DisplayMemberPath="Nev" Grid.Row="1" Grid.Column="1" Margin="5"
IsSynchronizedWithCurrentItem="True"/>
<Label Content="Írányítószám" Grid.Row="2" Grid.Column="0"/>
<TextBox x:Name="tbIrsz" Grid.Row="2" Grid.Column="1" Margin="5"/>
<Label Content="Helységnev" Grid.Row="3" Grid.Column="0"/>
<TextBox x:Name="tbHelysegnev" Grid.Row="3" Grid.Column="1" Margin="5"/>
<StackPanel Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="2"
Orientation="Horizontal" HorizontalAlignment="Center">
<Button x:Name="btRogzit" Content="Módosított adatpár rögzítése"
Click="btRogzit_Click" Margin="50,50,10,10"/>
<Button x:Name="btUjHelyseg" Content="Új helység" Margin="50,50,10,10"
Click="btUjHelyseg_Click"/>
<Button x:Name="btVissza" Content="Vissza" Click="btVissza_Click"
Margin="50,50,10,10" />
</StackPanel>
</Grid>
</StackPanel>
</Window>

```

Minden olyan menüponthoz, amelyből nem nyílik almenü, egy eseménykezelőt készítünk a Properties ablak események (Events) funkciójával.

A Mentés menüpont eseménykezelőjében elmentjük az adatbázisba az eddig végrehajtott módosításokat.

```

private void miMentes_Click(object sender, RoutedEventArgs e)
{
    cnTelefonszamok.SaveChanges();
}

```

A Kilépés menüpont eseménykezelőjében kilépünk a programból.

```

private void miKilepes_Click(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown();
}

```

### 4.2.3. Egyszerű lekérdezés

A helységadatok megjelenítésénél egy táblázatot szeretnénk látni, amelyben a helységek nevei és irányítószámaik láthatóak. Az eseménykezelőben először láthatóvá tesszük az adatrácsot, majd végrehajtjuk a lekérdezést és végül a lekérdezés eredményét adatkötéssel kapcsoljuk a rácshoz. A lekérdezés eredményét egy listává alakítjuk, mert csak ekkor fog ténylegesen végrehajtódni a LINQ lekérdezés.

```

private void miHelysegek_Click(object sender, RoutedEventArgs e)
{
    grHelyseg.Visibility = Visibility.Hidden;
    dgAdatracs.Visibility = Visibility.Visible;
    var er = (from x in cnTelefonszamok.enHelysegek
select new { x.Nev, x.Irsz }).ToList();
    dgAdatracs.ItemsSource = er;
}

```

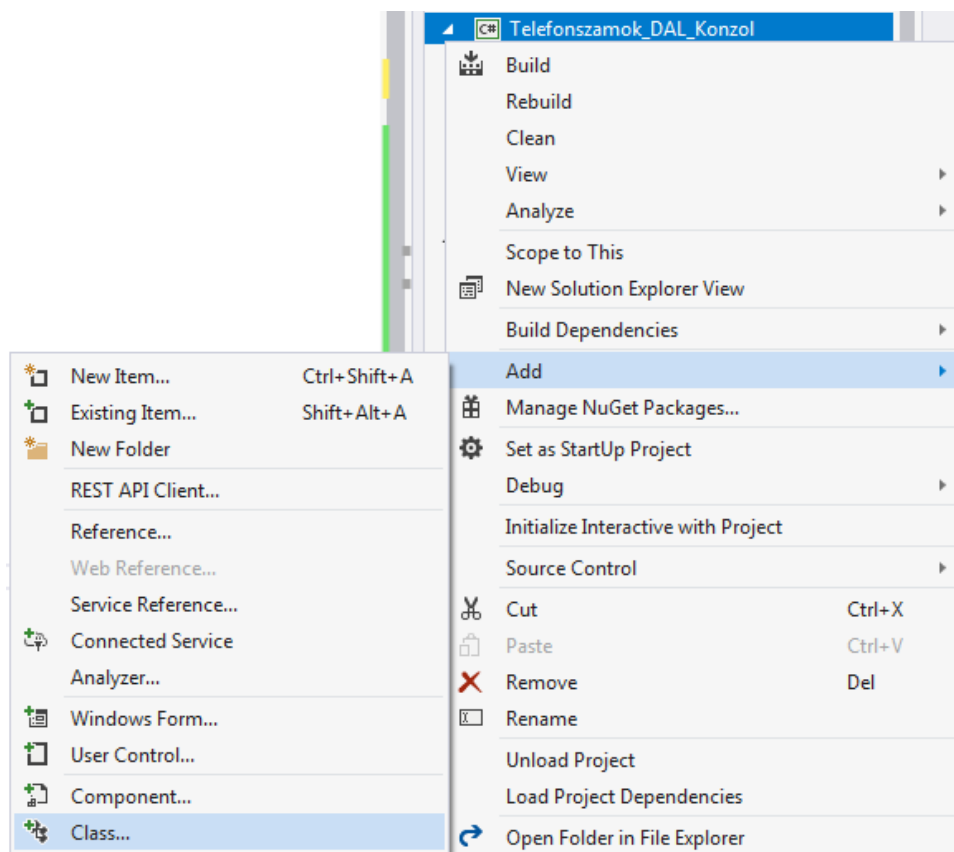
Nev	Irsz
Kecskemét	6000
Sóskút	2038
Pusztazámor	2039
Budaörs	2040
Törökbálint	2045
Remeteszőlős	2090
Remeteszőlős	2090
Remeteszőlős	2090

159. ábra.

Próbáljuk ki a kódot úgy is, hogy elhagyjuk a ToList() metódushívást.

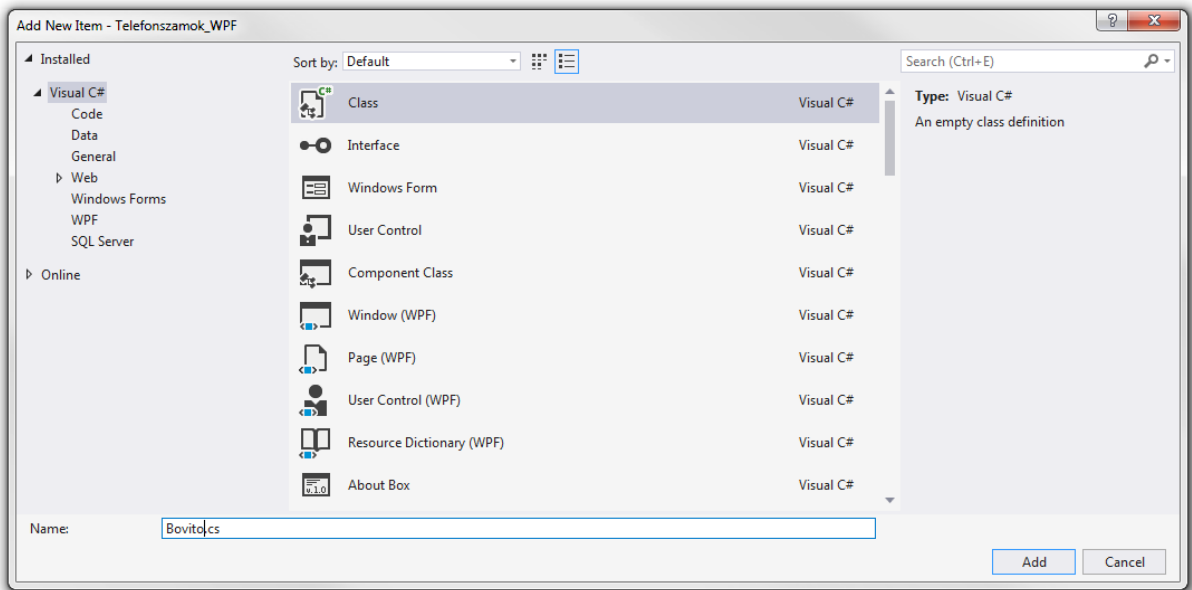
#### 4.2.4. Komplex lekérdezés

Az összes adat megjelenítésénél táblázatos formában szeretnénk látni minden tárolt adatot. Ennek érdekében egy olyan gyűjteményt kell előállítanunk, aminek egy eleme egy személy összes adatát tartalmazza. Mivel egy személyhez több telefonszám is tartozik, ezért azt szeretnénk elérni, hogy ezek a számok egyetlen sztringben, egymástól vesszővel elválasztva jelenjenek meg. Ezt a részfeladatot úgy oldjuk meg, hogy készítünk egy bővítő metódust az enSzemély entitás osztályhoz a CustomExtensions névtérben. A bővítő metódust egy Bővítő nevű statikus osztályban helyezük el.



160. ábra.



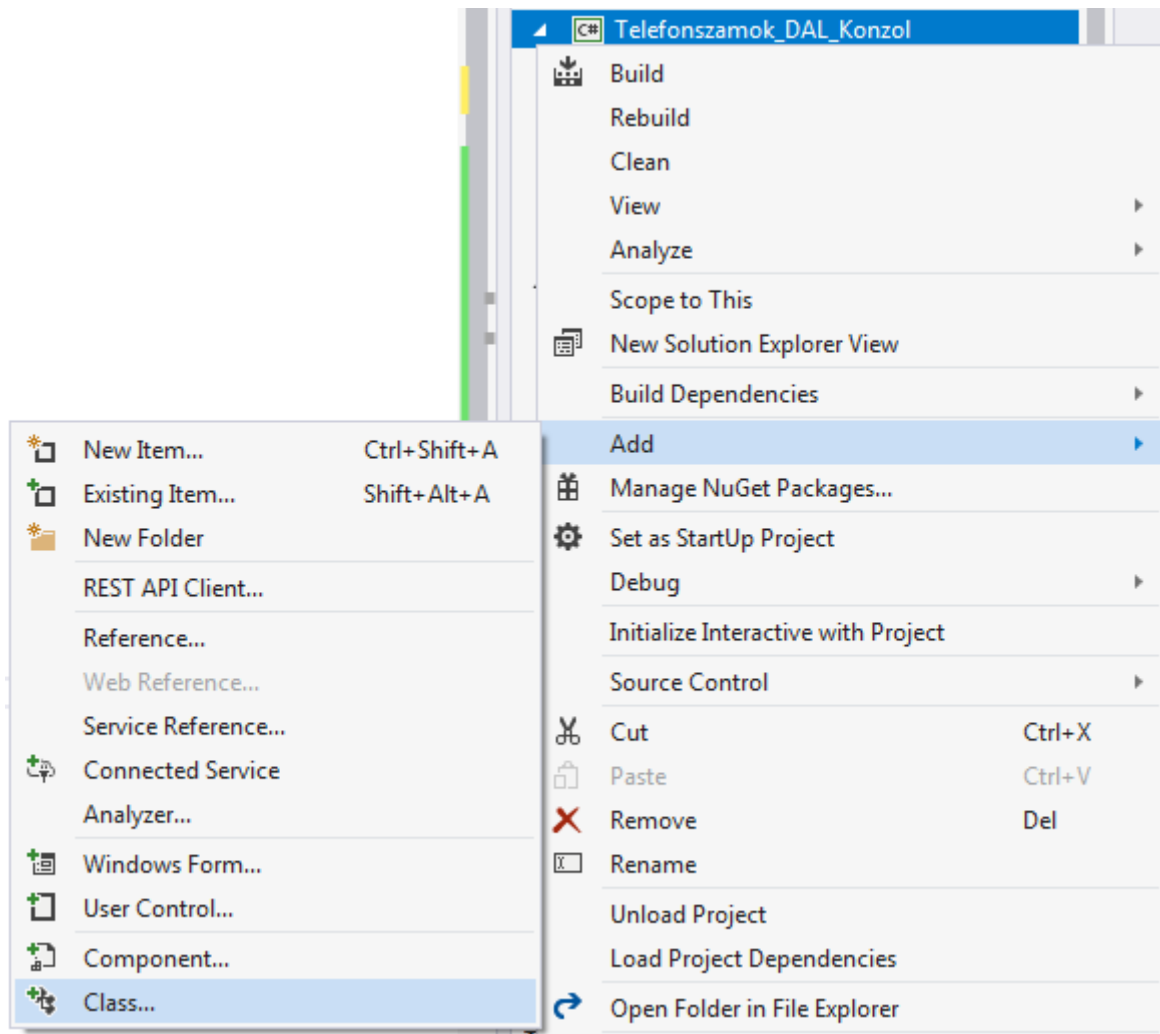


161. ábra.

A Telefonszámok bővítő metódusban megkapjuk egy enSzemely entitás referenciáját, és végigiterálunk a kapcsolódó telefonszám információkat tartalmazó entításokon. Az utolsó kivételével mindegyik után teszünk egy vesszőt. A metódus visszaadja a telefonszámok listáját tartalmazó sztringet.

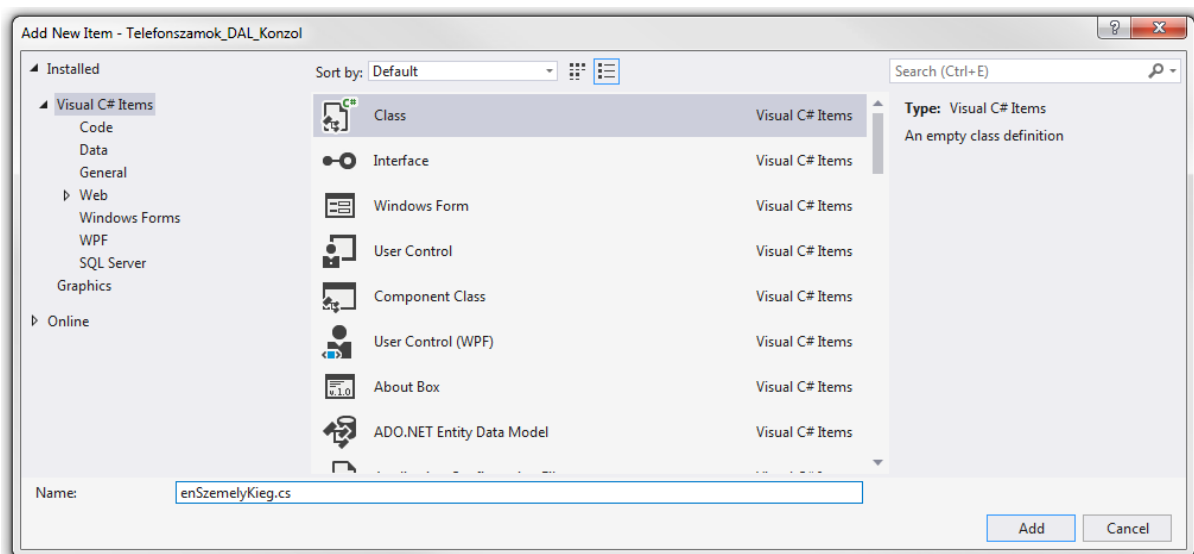
```
using Telefonszamok_DAL_Konzol;
namespace Telefonszamok_WPF
{
    public static class Bovito
    {
        public static string Telefonszamok(this enSzemely enSzemely)
        {
            var s = "";
            foreach(var x in enSzemely.enTelefonszamok)
            {
                s = s + x.Szam;
                if (x != enSzemely.enTelefonszamok.Last())
                    s = s + ", ";
            }
            return s;
        }
    }
}
```

A telefonszámlista előállításának egy másik útja az is lehetne, hogy a Telefonszamok\_DAL\_Konzol projektben létrehozunk a személyeket leíró enSzemely entitás osztályhoz egy kiegészítést (partial class), amiben egy új tulajdonságot definiálunk Telefonszámok néven. Ehhez a Solution Explorerben egy új osztály adunk a Telefonszamok\_DAL\_Konzol projekthez.



162. ábra.

A létrehozáskor az enSzemelyKieg nevet adjuk az új osztálynak, azonban a C# kódban ezt átírjuk.



163. ábra.

A tulajdonság csak get elérővel rendelkezik, és a lekérdezéshez tartozó kód gyakorlatilag azonos az előzőekben ismertetettel.

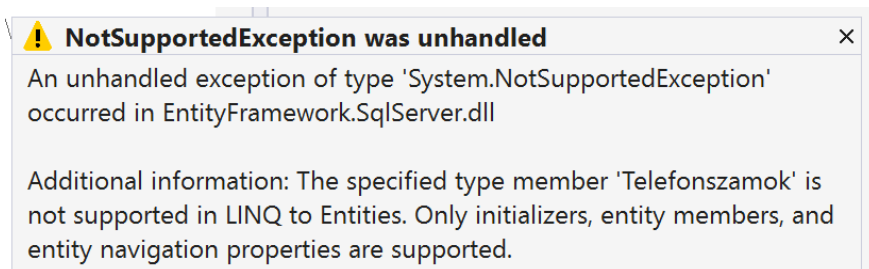
```
namespace Telefonszamok_DAL_Konzol
{
    public partial class enSzemely
    {
        public string Telefonszamok
        {
            get
            {
                var s = "";
                foreach(var x in enTelefonszamok)
                {
                    s = s + x.Szam;
                    if (x != enTelefonszamok.Last())
                        s = s + ", ";
                }
                return s;
            }
        }
    }
}
```

Ennek a megoldásnak az az előnye, hogy a későbbiekben a Telefonszámok tulajdonságra ugyanúgy tudunk hivatkozni, mintha az az entitás egy tárolt tulajdonsága lenne.

Visszatérve a MainWindow osztály miMindenAdat\_Click metódusához a következő feladatunk az, hogy előállítsuk azt a gyűjteményt, amit adatkötéssel az adatrácshoz kívánunk rendelni. Első ötletként az alábbi kódrészlet kínálkozik elegáns megoldásként. Egy LINQ lekérdezés, ami kihasználja az előzőekben alternatív megoldásként bemutatott Telefonszámok tulajdonságot.

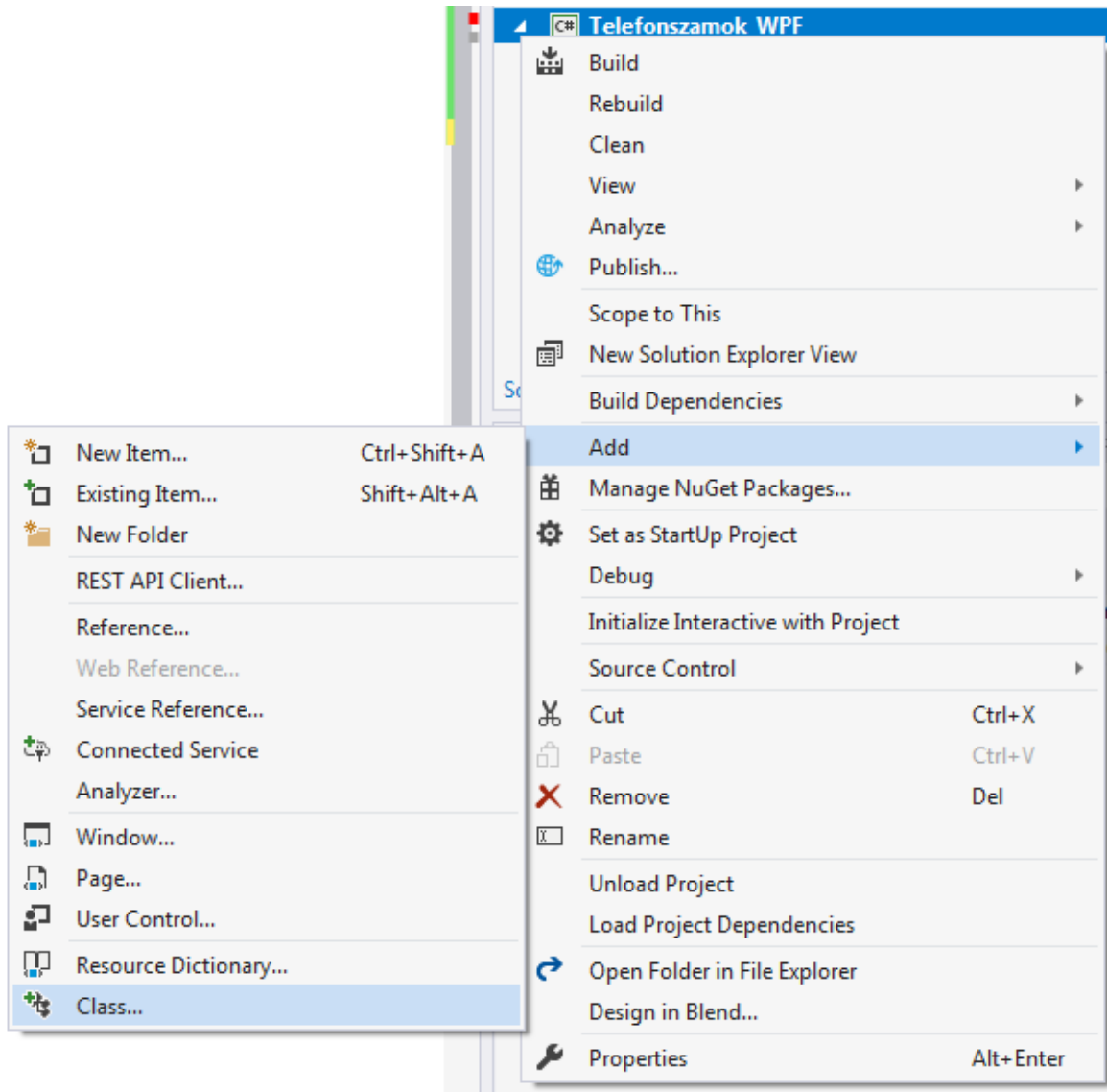
```
var er = (from x in cnTelefonszamok.enSzemelyek orderby x.Vezeteknev
select new
{
    x.Vezeteknev,    x.Utonev,    x.enHelyseg.Irsz,    x.enHelyseg.Nev,    x.Lakcim,
    x.Telefonszamok }).ToList();
```

Futtatáskor azonban azt kell tapasztalnunk, hogy az Entity Framework 6.1 változatához kapcsolódó LINQ to Entities a fentiekben ismertetett két megoldás egyikét se támogatja LINQ lekérdezésben.

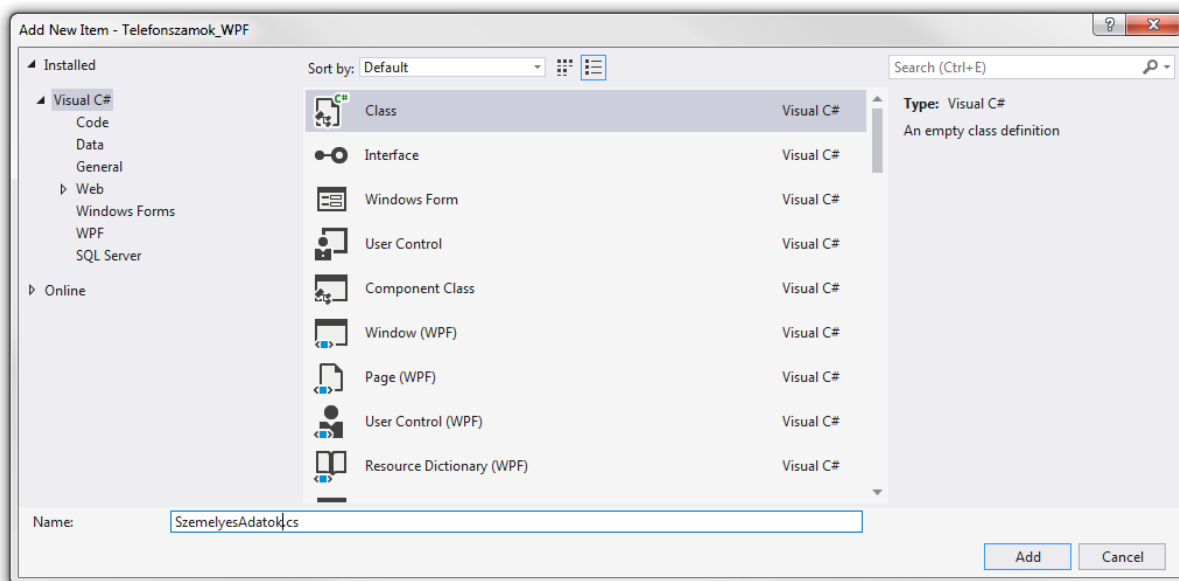


164. ábra.

Mivel LINQ-val nem tudjuk célunkat elérni, ezért egy foreach ciklussal haladunk végig a személy entitásokat tartalmazó gyűjteményen. LINQ nélkül az eredmény gyűjtemény előállításához először definiálnunk kell egy segédosztályt (SzemelyAdatok).



165. ábra.



166. ábra.

```
public class SzemelyesAdatok
{
    public string Vezeteknev { get; set; }
    public string Utonev { get; set; }
    public Int16 Irsz { get; set; }
    public string Helysegnev { get; set; }
    public string Lakcim { get; set; }
    public string Telefonszamok { get; set; }
}
```

A MainWindow osztály miMindenAdat\_Click metódusában létrehozunk egy típusos lista objektumot, amelynek elemtípusa SzemélyAdatok, majd az entitás halmazon végighaladva, minden személyhez létrehozunk egy SzemélyAdatok típusú névtelen objektumot, amit hozzáadunk a listához.

```
private void miMindenAdat_Click(object sender, RoutedEventArgs e)
{
    grHelyseg.Visibility = Visibility.Hidden;
    dgAdatracs.Visibility = Visibility.Visible;
    var er = new List<SzemelyesAdatok>();
    foreach (var x in cnTelefonszamok.enSzemelyek)
    {
        er.Add(new SzemelyesAdatok()
        {
            Vezeteknev = x.Vezeteknev,
            Utonev = x.Utonev,
            Helysegnev = x.enHelyseg.Nev,
            Irsz = x.enHelyseg.Irsz,
            Lakcim = x.Lakcim,
            Telefonszamok = x.Telefonszamok
        });
    }
    dgAdatracs.ItemsSource = er;
}
```

A programot futtatva a Lekérdezések menü Minden adat táblázatosan menüpontját választva az alábbi ábrán látható táblázatot kapjuk.

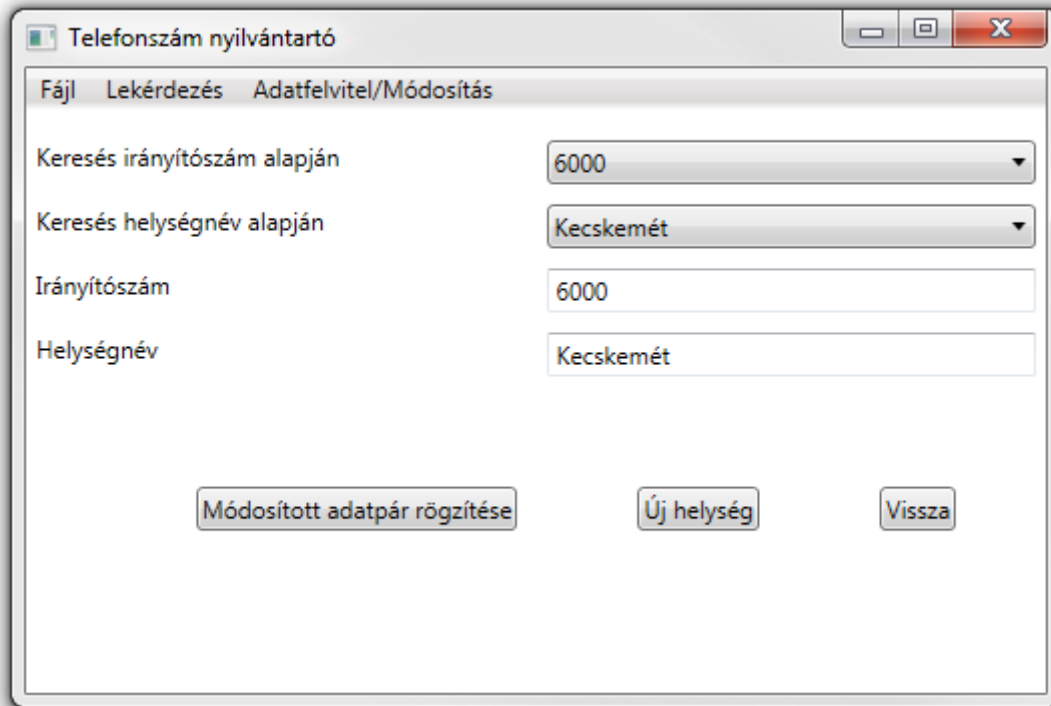
Vezeteknev	Utonev	Irsz	Helysegnev	Lakcim	Telefonszamok
Senki	Alfonz	6000	Kecskemét	Kis út 1.	+36-444-444
Gipsz	Jakab	6000	Kecskemét	Malom köz 1.	
Erős	Áron	2039	Pusztazámor	Alma rét 3.	+36-333-333
Olvasó	Jolán	2040	Budaörs	Fűzes út 2.	+36-111-111, +36-222-222
Argon	Géze	2090	Remeteszőlős	Ordas Köz 6.	+36-555-555, +36-666-666

167. ábra.

#### 4.2.5. Helységadatok módosítása

A helységadat adatfelvitel/módosítás során a felhasználó két menüpont közül választhat, ezek az adatbázisban szereplő adatok módosítása és új adatok felvitele.

Módosítás esetén a két legördülő kombinált lista egyikével kiválasztja az aktuális települést, ezek adatai megjelennek a TextBox komponensekben, majd a módosítást követően kattint a „Módosított adatpár rögzítése” feliratú gombon.



168. ábra.

A kívánt adatok megjelenítéséhez a láthatóság beállítását követően az adatok listáját adatkötéssel a Grid-hez rendeljük, majd kiválasztjuk az első helységet.

```
private void miHelysegekAM_Click(object sender, RoutedEventArgs e)
{
    dgAdatracs.Visibility = Visibility.Collapsed;
    grHelyseg.Visibility = Visibility.Visible;
    grHelyseg.DataContext = cnTelefonszamok.enHelysegek.ToList();
    cbIrsz.SelectedItem = 0;
}
```

A két ComboBox együtt fog változni a „IsSynchronizedWithCurrentItem=True” attribútum használatának köszönhetően. Ezért elegendő hozzájuk egy közös eseménykezelő (cbIrsz\_SelectionChanged) készítése. Amikor bármelyikben változik a kiválasztott elem, akkor a két szövegmező tartalmát frissítjük.

```
private void cbIrsz_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    var enAktualis = ((ComboBox)sender).SelectedItem as enHelyseg;
    cbHelysegnev.SelectedItem = enAktualis;
    tbIrsz.Text = enAktualis.Irsz.ToString();
    tbHelysegnev.Text = enAktualis.Nev;
}
```

A Rögzít gombon történő kattintáskor a gyűjteményben módosítjuk az adatokat.

```
private void btRogzit_Click(object sender, RoutedEventArgs e)
{
```

```

var enAktualis = cbIrsz.SelectedItem as enHelyseg;
enAktualis.Irsz = Int16.Parse(tbIrsz.Text);
enAktualis.Nev = tbHelysegnev.Text;
grHelyseg.Visibility = Visibility.Hidden;
}

```

A fenti metódus nem tartalmazza a szövegmezőbe írt adatok ellenőrzését, ez az olvasó önálló feladata. A gyűjteménybe történő felvétel még nem jelenti az adatbázisban történő tárolást. Ez utóbbi csak a Fájl menü Mentés pontjának választása után történik meg.

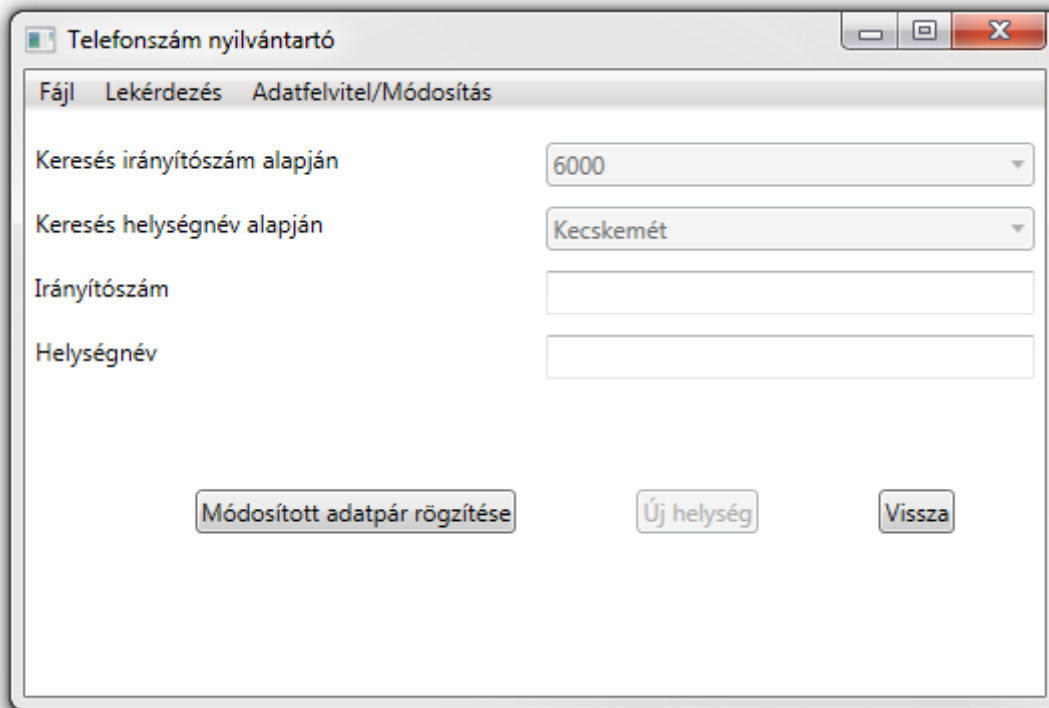
A Vissza gombon történő kattintás hatására eltüntetjük az űrlapot.

```

private void btVissza_Click(object sender, RoutedEventArgs e)
{
    grHelyseg.Visibility = Visibility.Hidden;
}

```

Amennyiben a felhasználó új helységadatokat akar felvinni az adatbázisba, akkor az „Új helység” gombon kattint, ami a két ComboBox letiltását és a szövegmezők ürítését eredményezi.



169. ábra.

```

private void btUjHelyseg_Click(object sender, RoutedEventArgs e)
{
    Beallit(false);
    tbIrsz.Text = "";
    tbHelysegnev.Text = "";
}

```

```

private void Beallit(bool b)
{
    btUjHelyseg.IsEnabled = b;
}

```

```

        cbIrsz.IsEnabled = b;
        cbHelysegnev.IsEnabled = b;
    }

```

A fentiekben ismertetett `btRogzit_Click` metódust kis módosítással alkalmassá tesszük az új adatok rögzítésére is. Az új változatban létrehozunk egy új `enHelyseg` objektumot, majd felvesszük azt a helységek gyűjteményébe.

```

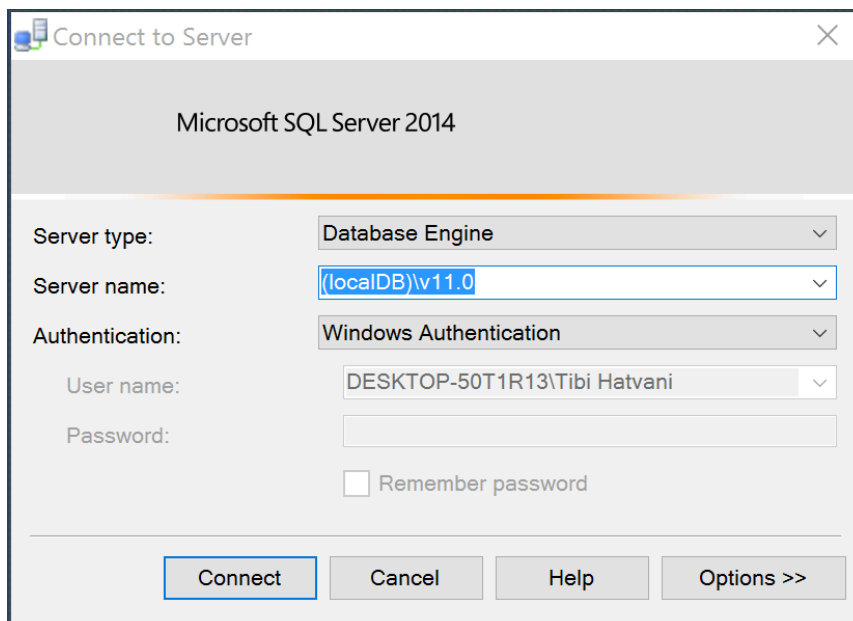
private void btRogzit_Click(object sender, RoutedEventArgs e)
{
    var enAktualis = cbIrsz.SelectedItem as enHelyseg;
    if (!btUjHelyseg.IsEnabled)
    {
        enAktualis = new enHelyseg();
        cnTelefonszamok.enHelysegek.Add(enAktualis);
    }
    enAktualis.Irsz = Int16.Parse(tbIrsz.Text);
    enAktualis.Nev = tbHelysegnev.Text;
    grHelyseg.Visibility = Visibility.Hidden;
}

```

Hasonlóan a módosítás esetéhez, tartós tárolás itt is csak a Mentés menüpont segítségével érhető el.

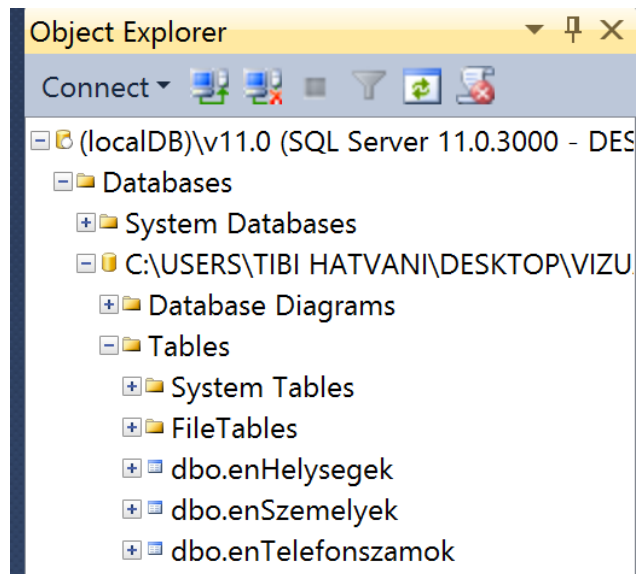
#### 4.3. Az adatbázisban tárolt adatok lementése SQL szkriptbe az adatbázis szerkezettel együtt, majd az adatbázis újbóli létrehozása

A *gyakorlat célja* az, hogy áttekintsük, hogy hogyan tudjuk az adatbázist átvinni egy másik gépre/kiszolgálóra az adatbázis állomány (Telefonszamok.mdf) mozgatása/másolása nélkül. A feladat végrehajtásához a Microsoft SQL Server Management Studio-t és a Visual Studio-t használjuk. Indítsuk el a Microsoft SQL Server Management Studio-t.



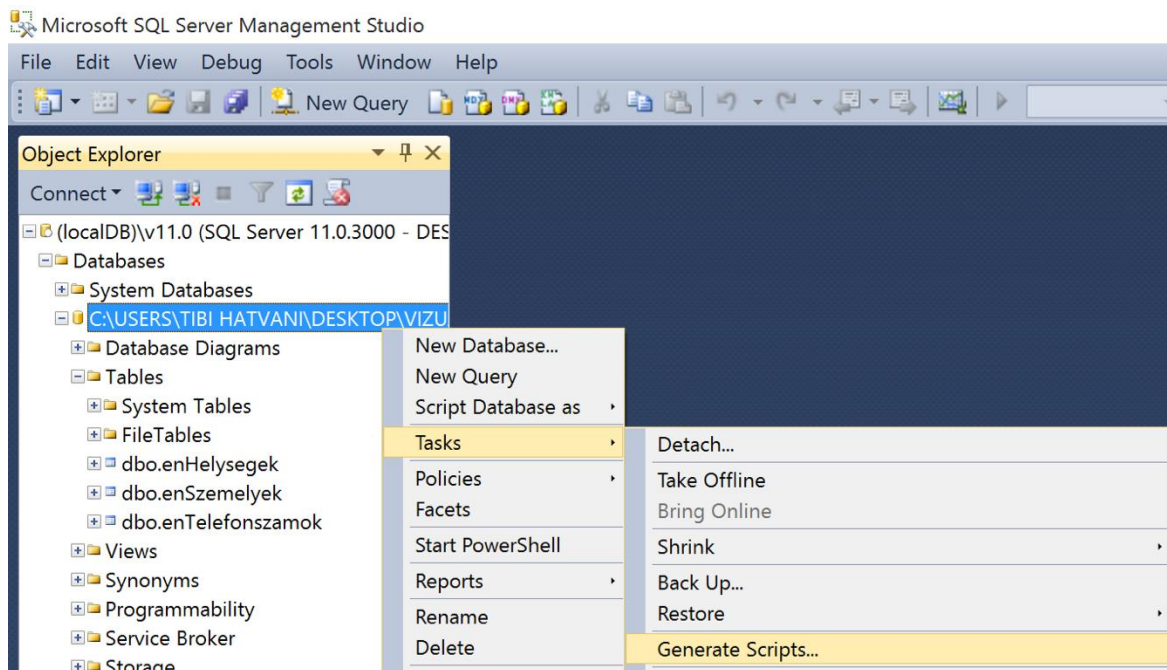
170. ábra.



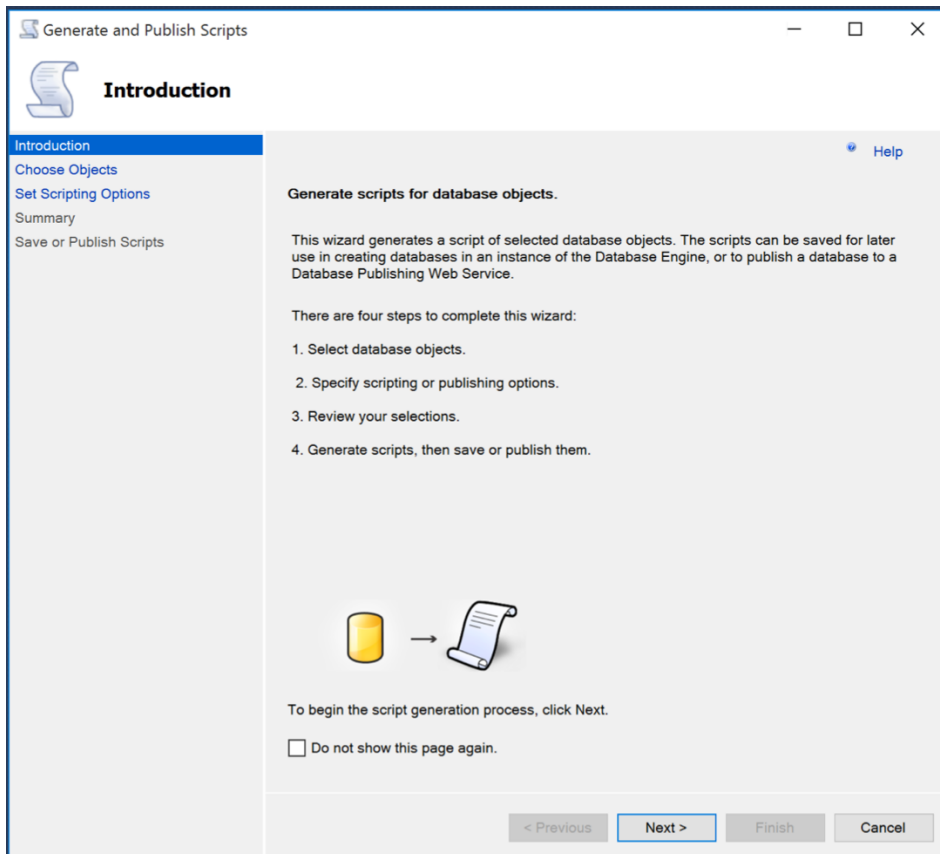


171. ábra.

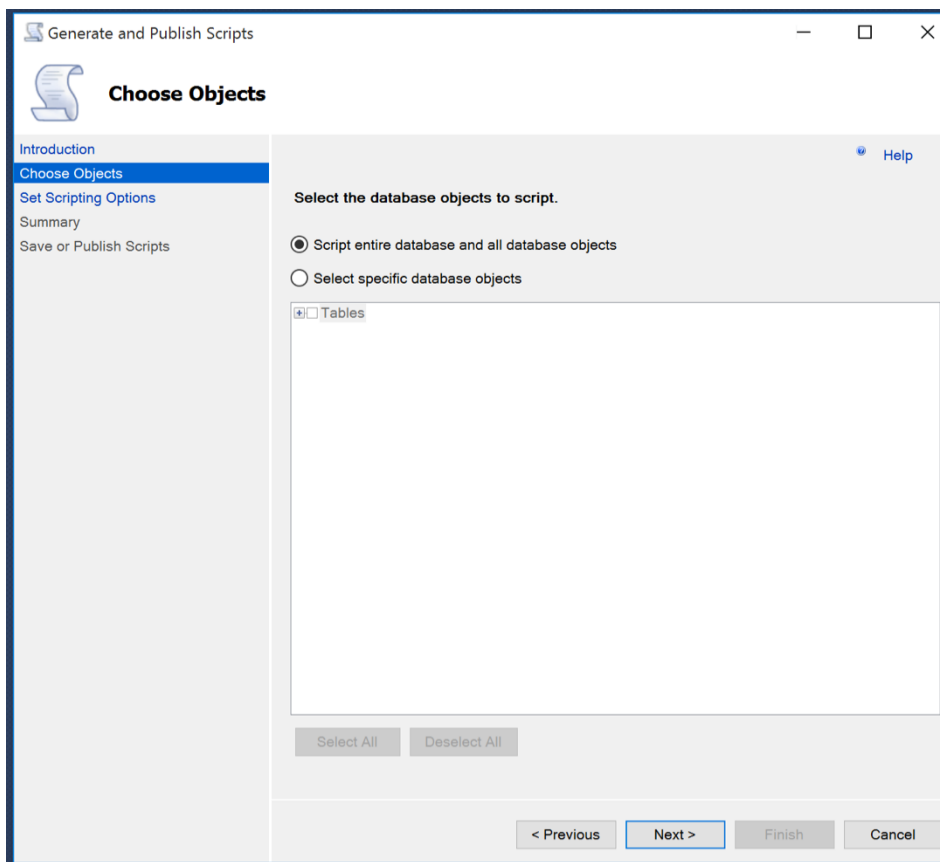
Lementés SQL-szkriptbe:



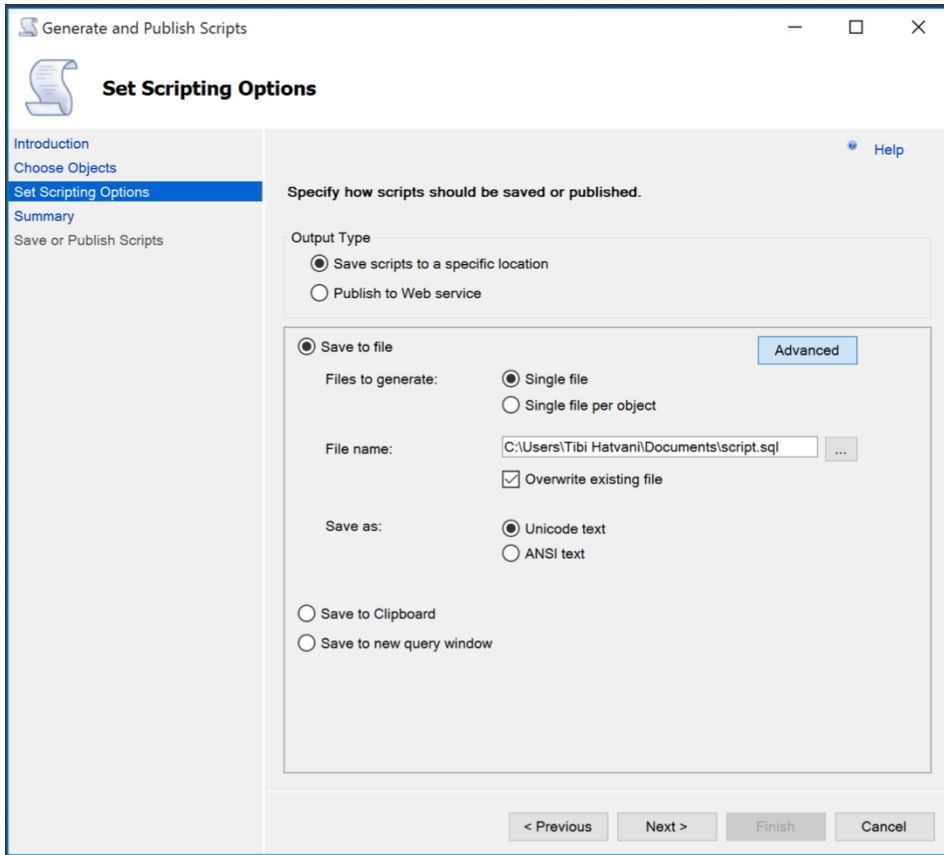
172. ábra.



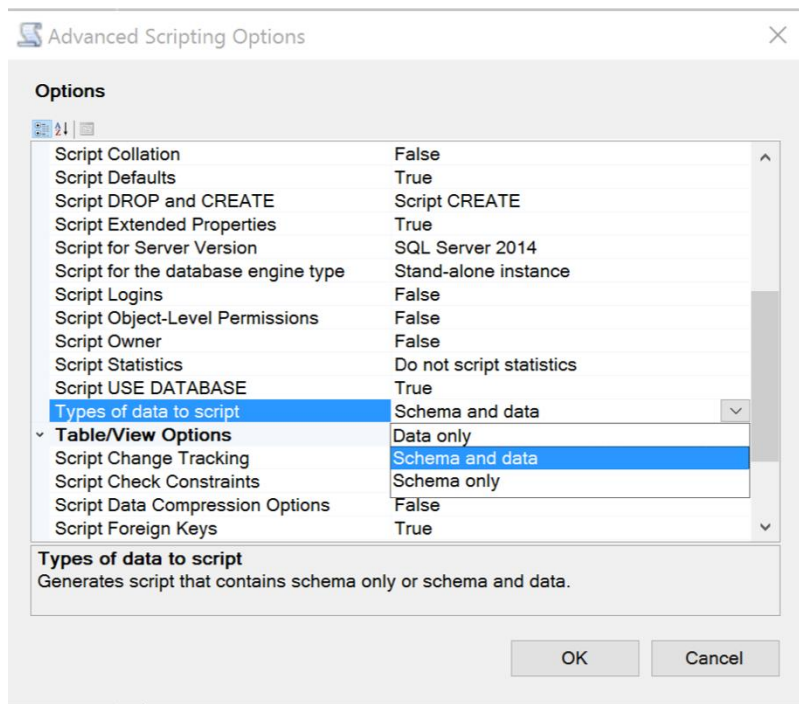
173. ábra.



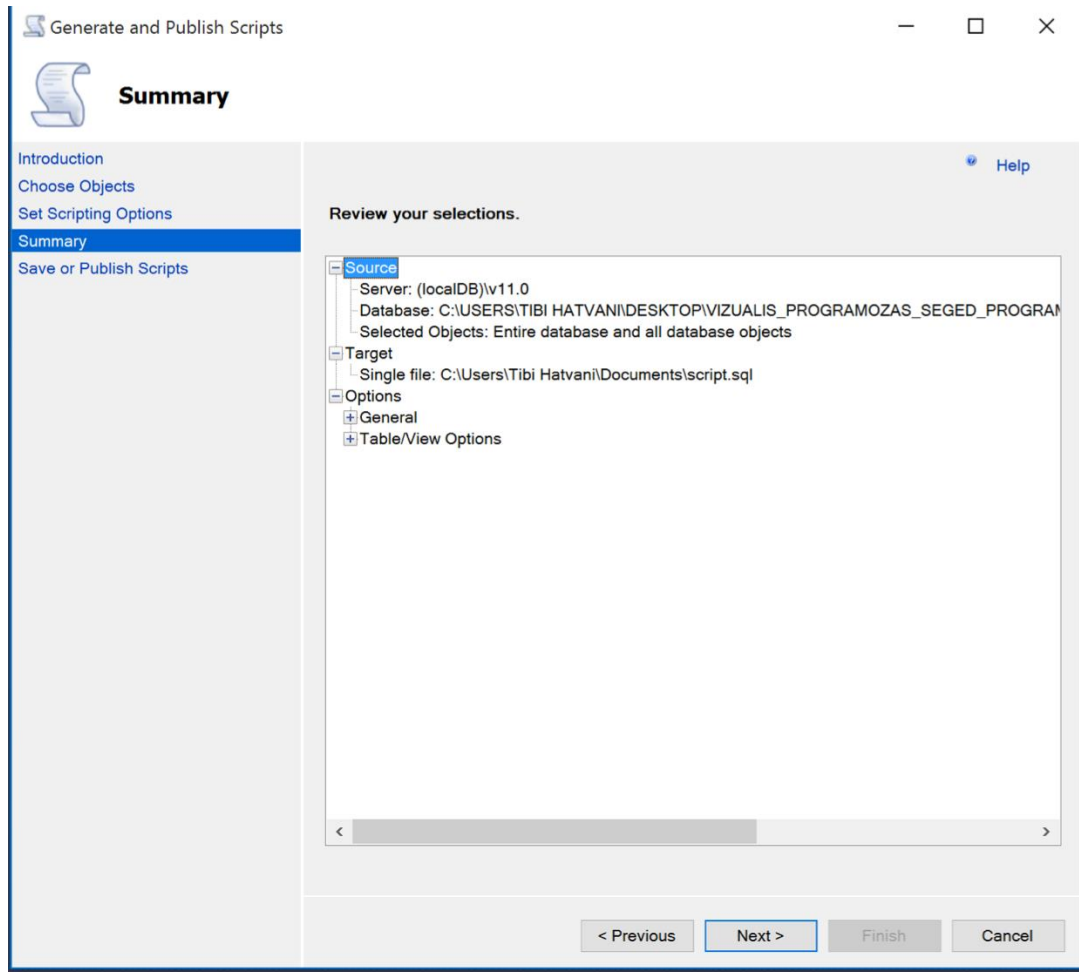
174. ábra.



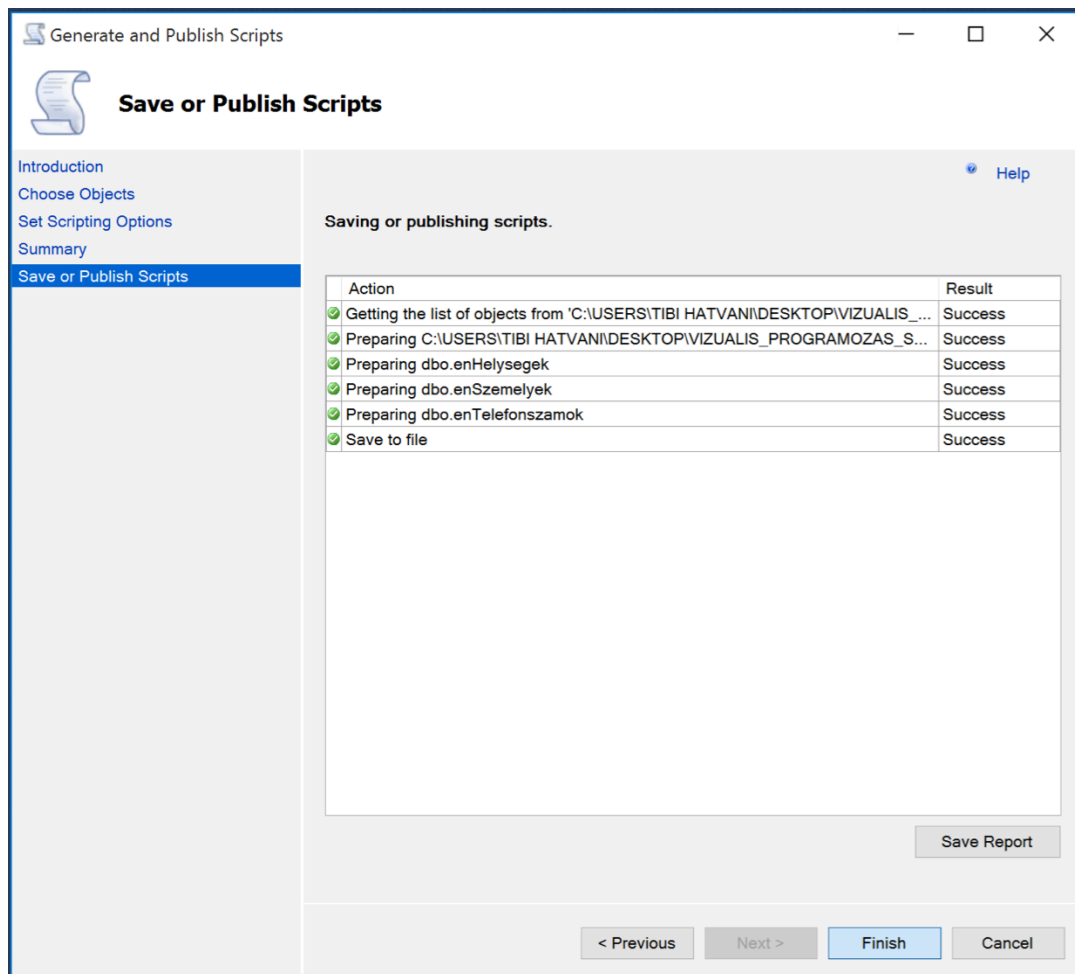
175. ábra.



176. ábra.



177. ábra.



178. ábra.

Visual Studio-ban nyissuk meg a szkriptet. Töröljük belőle az adatbázis fájl létrehozására vonatkozó részt (a kijelölt rész az alábbi képen), majd mentjük el a szkriptet.

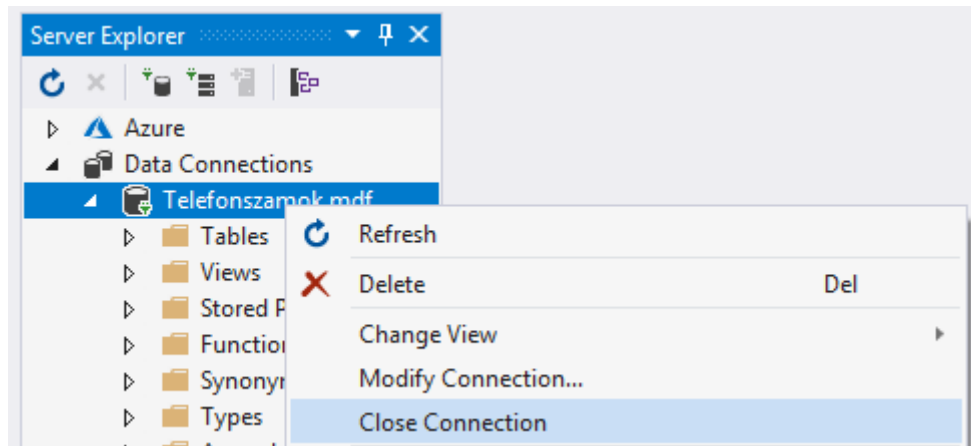
```

script.sql  MainWindow.xaml.cs  MainWindow.xaml
USE [master]
GO
/***** Object: Database [C:\USERS\TIBI HATVANI\DESKTOP\VIZUALIS_PROGRAMOZAS_SEGED_PROGRAMOK\TELEFONSZAMOK_DAL_KONZOL\TELEFONSZAMOKDB
CREATE DATABASE [C:\USERS\TIBI HATVANI\DESKTOP\VIZUALIS_PROGRAMOZAS_SEGED_PROGRAMOK\TELEFONSZAMOK_DAL_KONZOL\TELEFONSZAMOKDB]
CONTAINMENT = NONE
ON PRIMARY
( NAME = N'TelefonSzamokDB', FILENAME = N'C:\Users\Tibi Hatvani\Desktop\Vizualis_programozas_seged_programok\TelefonSzamok_DAL_Konzol\
LOG ON
( NAME = N'TelefonSzamokDB_log', FILENAME = N'C:\Users\Tibi Hatvani\Desktop\Vizualis_programozas_seged_programok\TelefonSzamok_DAL_Kon
GO

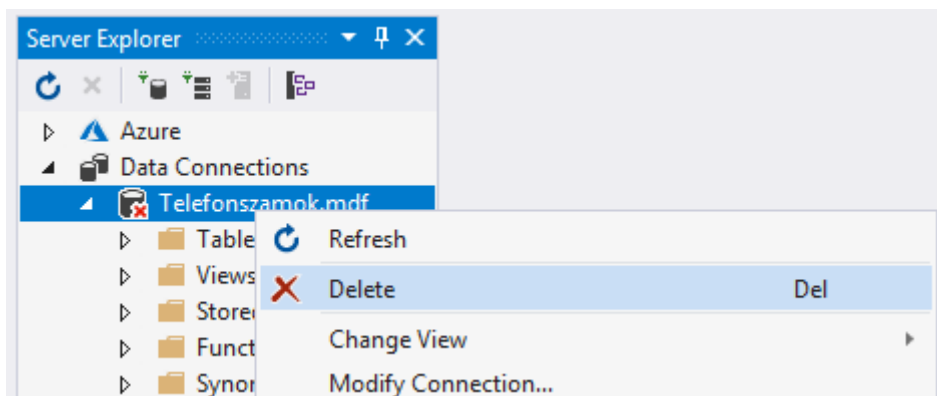
```

179. ábra.

A szkript kipróbálásához először töröljük a meglévő adatbázist. Ehhez zárjuk le Visual Studioban a kapcsolatot, majd töröljük a kapcsolatot.

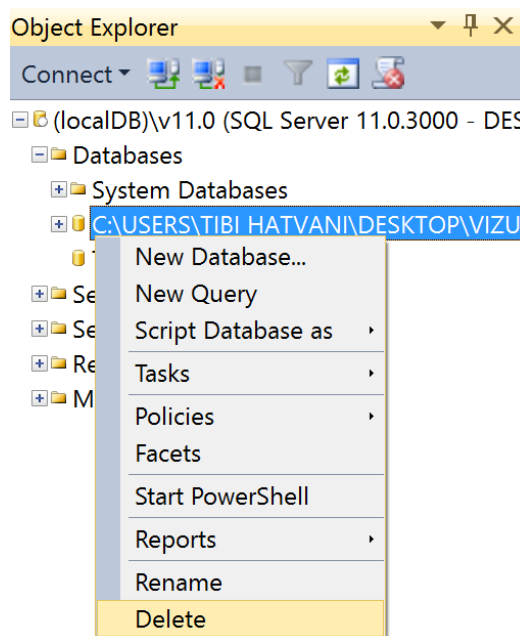


180. ábra.



181. ábra.

Microsoft SQL Management Studio-ban töröljük az adatbázist.

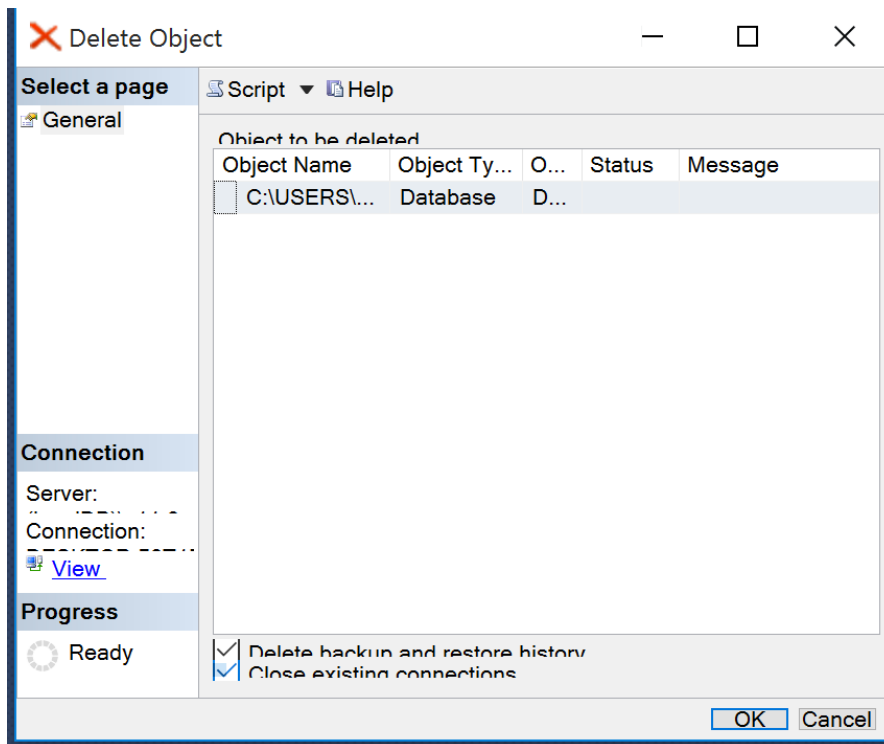


182. ábra.

Jelöljük be a következő rubrikákat a Delete Object párbeszédablak alján:

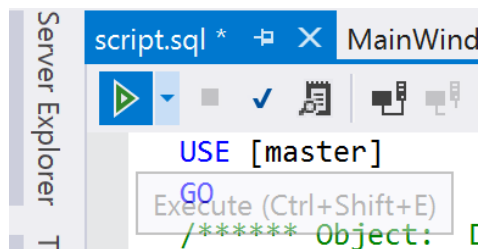
- Delete backup and restore history information for databases

- Close existing connections

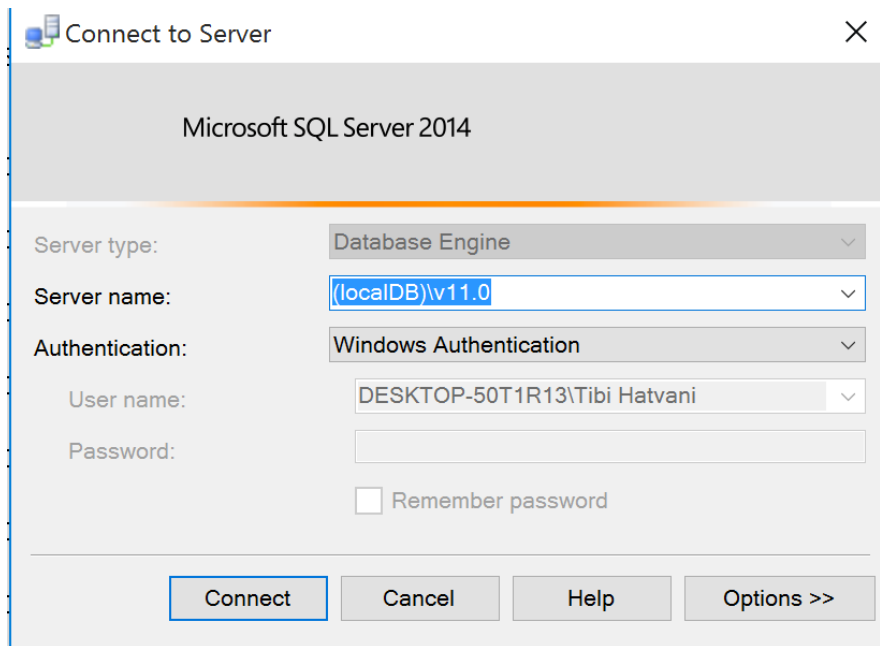


183. ábra.

Futtassuk le a szkriptet a Visual Studioban.



184. ábra.



185. ábra.

Ellenőrizzük le az adatbázis meglétét a Microsoft SQL Management Studio-ban.



## 5. Adatbáziskezelés – Code First Entity Framework

### 5.1. CikkAdatok WPF alkalmazás

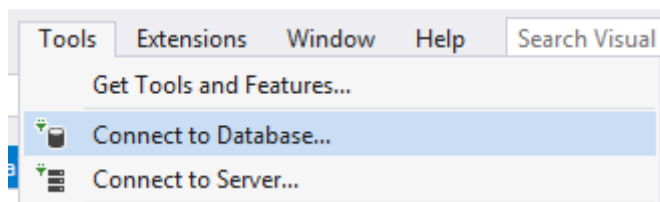
Az alkalmazás célja az, hogy tudományos folyóiratcikkek adatait jelenítsük meg és kezeljük. Elsőként hozzunk létre egy WPF alkalmazást CikkAdatok néven. Az ablakba csak egyetlen TextBox komponenst helyezünk el úgy, hogy kitöltse a teljes ablakot, többsoros legyen és a neve tbAdatok legyen.

```
<Grid>
  <TextBox x:Name="tbAdatok" TextWrapping="Wrap" AcceptsReturn="True"/>
</Grid>
```

Itt fogjuk megjeleníteni az egyszerű lekérdezéseket.

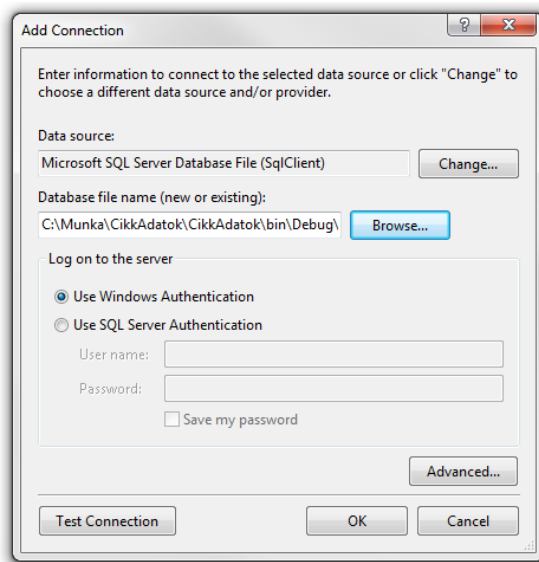
#### 5.1.1. Adatbázis állomány létrehozása

Az adatbázis állományt a Visual Studio segítségével hozzuk létre úgy, hogy az az EXE állománnyal azonos könyvtárba (|DataDirectory| belső változó).

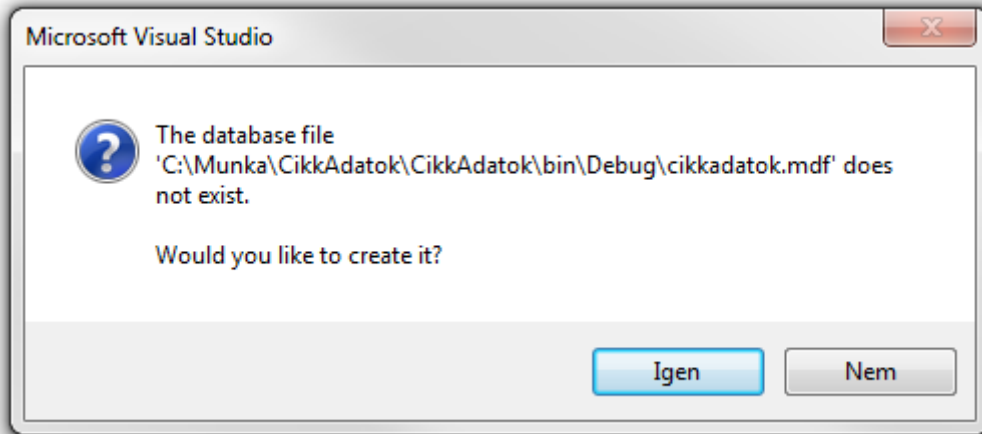


186. ábra.

Kiválasztjuk adatforrásként a Microsoft SQL ServerDatabase File-t és helyként a CikkAdatok\bin\Debug könyvtárat, majd megadjuk a cikkadatok.mdf fájlnévet.

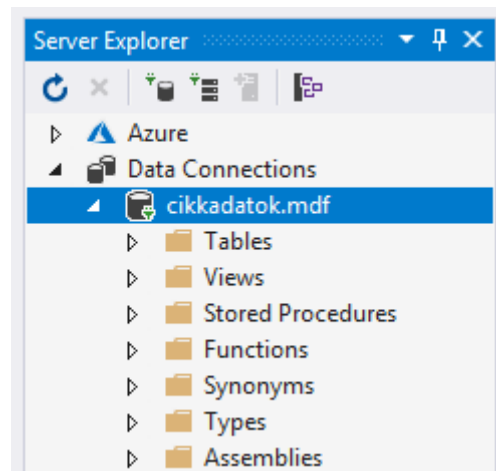


187. ábra.



188. ábra.

Az adatbázis létrejöttét leellenőrizzük a Server Explorer ablakban.



189. ábra.

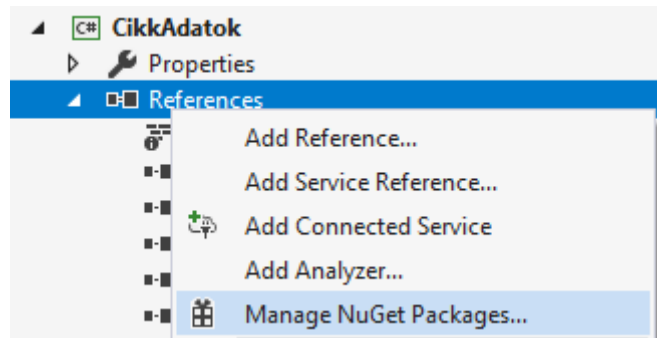
Az App.Config-hoz hozzáadjuk a kapcsolati sztringet a configuration tagen belül a startup tag után:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
  <connectionStrings>
    <add name="csCikkadatok"
          connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;attachdbfilename=|DataDirectory|\cikkadatok.mdf;Integrated Security=true"
          providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

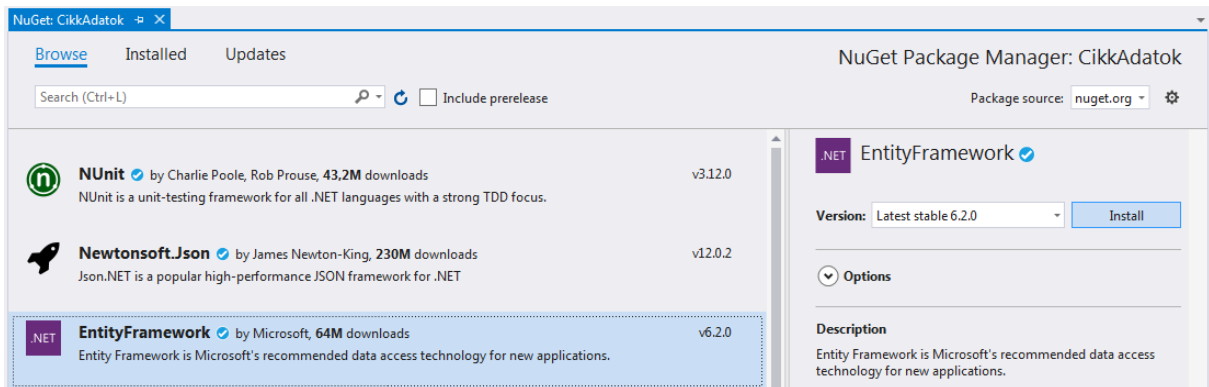
Az idézőjelek között megadott részen belül nem lehet sortörés a konfigurációs állományban. A kapcsolati sztring beillesztése

### 5.1.2. EF osztályok elkészítése

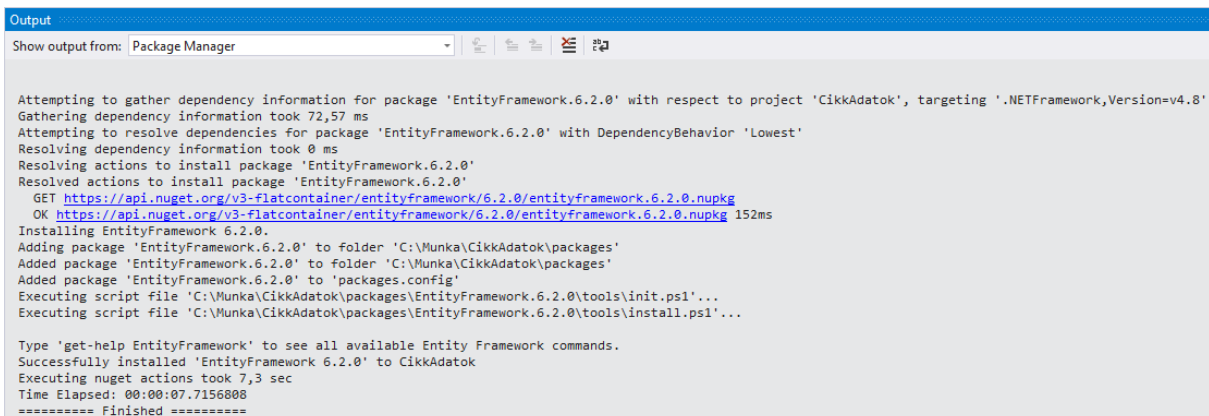
Az EF modell használatához fel kell vennünk az Entity Framework-öt a referenciák közé.



190. ábra.

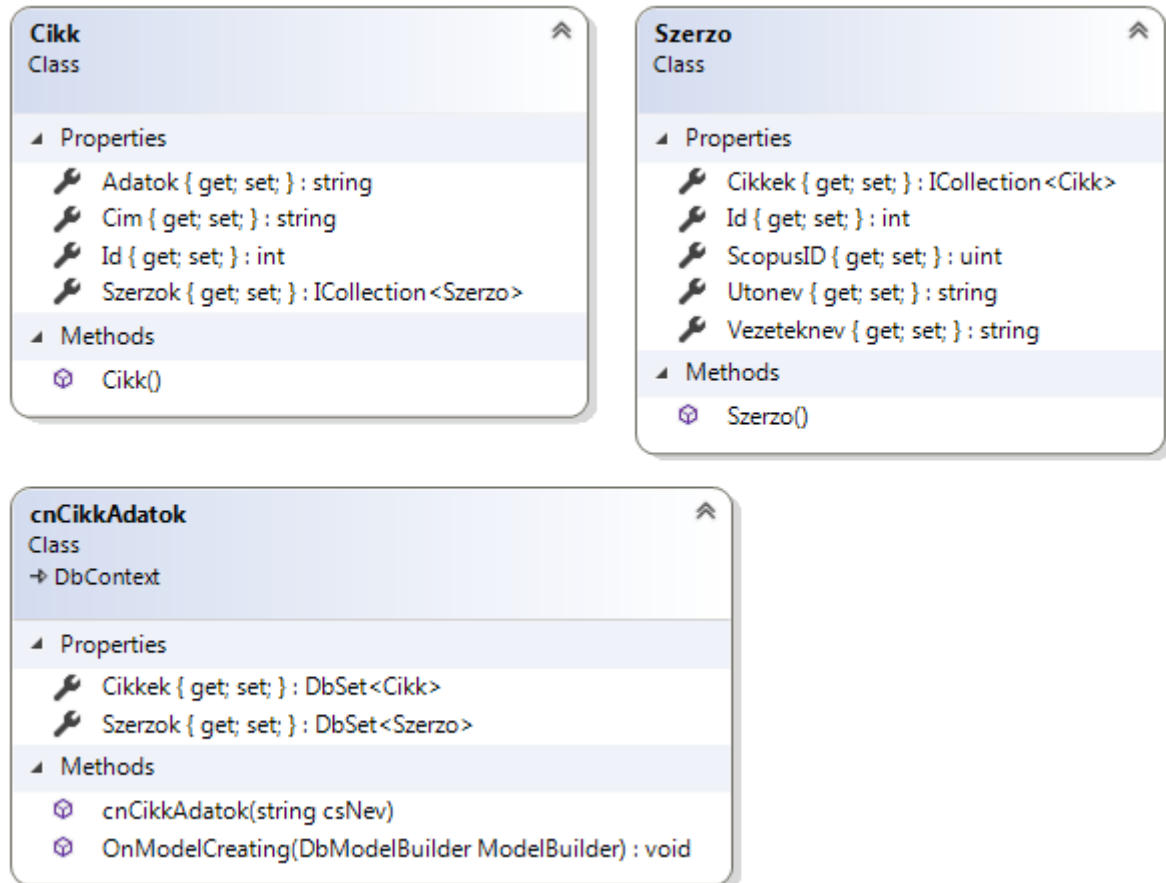


191. ábra.



192. ábra.

Az adatok kezeléséhez és tárolásához két entitás típust (Szerző és Cikk) fogunk használni, és ezek mellett egy entitás konténer osztályra lesz szükségünk. Az egyes adattagokat és típusaikat az alábbi osztálydiagram tartalmazza. Egy szerző több cikket írhat és egy cikknek több szerzője is lehet, ezért a két entitás típus között több a többhöz kapcsolatot állítunk fel. Ez azt jelenti, hogy mindkét entitás típusban a navigációs tulajdonság egy gyűjtemény lesz.



193. ábra.

#### A Szerző entitás osztályban

- Id – a szerző objektum egyedi azonosítója
- ScopusID – a szerző azonosítója a Scopus rendszerben (tudományos adatbázis)
- Vezetéknév és Utonev – a szerző neve
- Cikkek – egy gyűjtemény azon Cikk entitás objektumok referenciáival, amelyek elkészítésében az adott szerző részt vet – ez egy navigációs tulajdonság lesz

#### A Cikk entitás osztályban

- Id – a cikk objektum egyedi azonosítója
- Cim – a cikk címe
- Adatok – a cikk további adatai
- Szerzok – egy gyűjtemény azon Szerző entitás objektumok referenciáival, amely szerzők részt vettek a cikk elkészítésében – ez egy navigációs tulajdonság lesz

A cnCikkAdatok osztály az entitáskonténer feladatát látja el és a két entitás típusnak megfelelően két entitáshalmazt tartalmaz. Az egyes osztályok C# kódja az alábbi. A navigációs tulajdonságként használt gyűjtemény objektumokat a megfelelő osztály konstruktorában hozzuk létre.

```

public class Szerzo
{
    public Szerzo()
    {
        // A szerző által írt cikkek referenciáit tároló gyűjtemény létrehozása.
        Cikkek =new List<Cikk>();
    }
    public int Id { get; set; }
    public string Vezeteknev{ get; set; }
    public string Utonev{ get; set; }
    public UInt32 ScopusID{ get; set; }
    public ICollection<Cikk> Cikkek { get; set; }
}

public class Cikk
{
    public Cikk()
    {
        // A cikket író szerzők referenciáit tároló gyűjtemény létrehozása.
        Szerzok = new List<Szerzo>();
    }

    public int Id { get; set; }
    public string Cim { get; set; }
    public string Adatok { get; set; }
    public ICollection<Szerzo> Szerzok { get; set; }
}

```

A kapcsolati sztring nevét (ld. App.config állomány) át kell adnunk az entitáskonténer osztály konstruktorának

```

public class cnCikkAdatok:DbContext
{
    public cnCikkAdatok(string csNev) : base(csNev) { }
    public DbSet<Szerzo> Szerzok { get; set; }
    public DbSet<Cikk> Cikkek { get; set; }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Definiáljuk a két entitáshalmazt tároló táblák neveit.
        modelBuilder.Entity<Szerzo>().ToTable("Szerzok");
        modelBuilder.Entity<Cikk>().ToTable("Cikkek");
        // Definiáljuk a több a többhöz kapcsolatot a Szerző és a Cikk között.
        modelBuilder.Entity<Szerzo>()
            .HasMany<Cikk>(sz => sz.Cikkek)
            .WithMany(c => c.Szerzok)
            .Map(cs =>
                { // Definiáljuk a kapcsolótábla mezőit és nevét.
                    cs.MapLeftKey("SzerzoRefId");
                    cs.MapRightKey("CikkRefId");
                    cs.ToTable("SzerzoCikk");
                });
        // A szerzőre vonatkozó körkörös referenciák elkerülése érdekében.
        modelBuilder.Entity<Cikk>().HasRequired(
            x => x.Szerzok).WithMany().WillCascadeOnDelete(false);
        base.OnModelCreating(modelBuilder);
    }
}

```

### 5.1.3. Adatfelvitel

Az ablak konstruktorában létrehozzuk az entitás konténer objektumot, majd ha nincs még adat az adatbázisban, akkor felviszünk néhány tesztadatot, és végül meghívjuk az egyszerű lekérdező metódust.

```
public MainWindow()
{
    InitializeComponent();
    cnCikkAdatok = new cnCikkAdatok("name=csCikkAdatok");
    var db = (from x in cnCikkAdatok.Cikkek select x).Count();
    if (db == 0) // Csak egyszer szabad lefuttatni a feltöltést.
        Adatfeltolt();
    Lekerdez();
}
// <summary> Két cikk adatainak felvitele az adatbázisba. </summary>
private void Adatfeltolt()
{
    // Pelusi, D., Mascella, R.: Optimal control algorithms for second order systems,
    // (2013) Journal of Computer Science, 9 (2), pp. 183-197.
    var Szerzo1 = new Szerzo { Vezeteknev = "Pelusi", Utonev = "D." };
    var Szerzo2 = new Szerzo { Vezeteknev = "Mascella", Utonev = "R." };
    var UjCikk1 = new Cikk
    {
        Cim = "Optimal control algorithms for second order systems",
        Adatok = "(2013) Journal of Computer Science, 9 (2), pp. 183-197."
    };
    UjCikk1.Szerzok.Add(Szerzo1);
    UjCikk1.Szerzok.Add(Szerzo2);
    Szerzo1.Cikkek.Add(UjCikk1);
    Szerzo2.Cikkek.Add(UjCikk1);
    cnCikkAdatok.Szerzok.Add(Szerzo1);
    cnCikkAdatok.Szerzok.Add(Szerzo2);
    cnCikkAdatok.Cikkek.Add(UjCikk1);
    // Pelusi, D.; Vazquez, L. ; Diaz, D. ; Mascella, R.: Fuzzy algorithm control
    // effectiveness on drum boiler simulated dynamics,
    // 36th International Conference on Telecommunications and Signal Processing
    // (TSP), 2-4 July 2013, Rome, pp. 272 - 276.
    var UjCikk2 = new Cikk
    {
        Cim = "Fuzzy algorithm control effectiveness on drum boiler simulated "+
            "dynamics",
        Adatok = "36th International Conference on Telecommunications and "+
            "Signal Processing (TSP), 2-4 July 2013, Rome, pp. 272 - 276."
    };
    var Szerzo3 = new Szerzo { Vezeteknev = "Vasquez", Utonev = "L." };
    var Szerzo4 = new Szerzo { Vezeteknev = "Diaz", Utonev = "D." };
    UjCikk2.Szerzok.Add(Szerzo1);
    UjCikk2.Szerzok.Add(Szerzo3);
    UjCikk2.Szerzok.Add(Szerzo4);
    UjCikk2.Szerzok.Add(Szerzo2);
    Szerzo1.Cikkek.Add(UjCikk2);
    Szerzo2.Cikkek.Add(UjCikk2);
    Szerzo3.Cikkek.Add(UjCikk2);
    Szerzo4.Cikkek.Add(UjCikk2);
    cnCikkAdatok.Szerzok.Add(Szerzo3);
    cnCikkAdatok.Szerzok.Add(Szerzo4);
    cnCikkAdatok.Cikkek.Add(UjCikk2);
    cnCikkAdatok.SaveChanges();
}
}
```

### 5.1.4. Egyszerű lekérdezések

```
private void Lekerdez()
{
    var s = "Összes szerző listája:\r\n";
    // Összes szerző listája. Az Include hatására a szerzőkhöz kapcsolódó
    // cikkeket is beolvassa, így a Cikkek navigációs tulajdonság is
    // értéket kap. Egyébként null értéke lenne.
    foreach (var x in cnCikkAdatok.Szerzok.Include(x=>x.Cikkek))
        s += x.Vezeteknev + " " + x.Utonev + "\n";
    // Szerzőlista ablakba.
    tbAdatok.Text = s+"\r\nPelusi cikkeinek adatai:\r\n";

    s = "";
    // Pelusi cikkeinek adatai.
    // Megkeressük Pelusit.
    var P = cnCikkAdatok.Szerzok.Where(
        x => x.Vezeteknev == "Pelusi" && x.Utonev == "D.").FirstOrDefault();
    // Sorrövezzük Pelusi cikkeit.
    foreach (var y in P.Cikkek)
    {
        // Felsoroljuk az aktuális cikk szerzőit.
        var w = y.Szerzok.Aggregate("", (current, z) => current +
            z.Utonev + " " + z.Vezeteknev + ", ");
        // Hozzáadjuk a cikk címét és adatait.
        s += w+ y.Cim + ", " + y.Adatok + "\r\n\r\n";
    }
    // Eredmények ablakba.
    tbAdatok.Text += s;
}
```





## 6. Windows Forms – Adatkötés, adatbáziskezelés

### 6.1. Access adatbázis elérése OLE DB-n keresztül

Készítsünk egy grafikus felülető alkalmazást, ami lehetővé teszi egy Access adatbázisban tárolt hallgatói adatok (Neptun, Név, e-mail cím) lekérdezését (összes adat, e-mail címek listája), új adatok felvitelét, és a felvitt adatok módosítását. A felhasználói felület Windows Forms típusú legyen, kapcsolat nélküli adatbázis elérési modellt használjunk, és típusos DataSet-ben tároljuk a memóriában az adatokat.

#### 6.1.1. A felhasználói felület létrehozása

Hozzunk létre egy Windows Forms Application típusú C# alkalmazást (.NET Framework) *Lista* néven. A form neve legyen *frmFoablak*, az őt tartalmazó állomány neve legyen *frmFoablak.cs*, az ablak felirata legyen: Access adatbázis elérése OLE DB-n keresztül.

Helyezzünk el a formon egy menüt (MenuStrip), aminek a neve legyen: *msFomenu*.

Menüpontok:

**Fájl:** Name=tsmiFajl

**Mentés:** Name=tsmiMentes

**Kilépés:** Name=tsmiKilepes

**Lekérdezés:** Name=tsmiLekerdezés

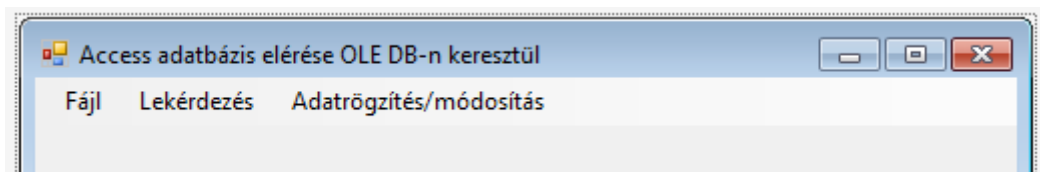
**Összes adat:** Name=tsmiOsszesAdat

**E-mail címek listája:** Name=tsmiEmailLista

**Adatrögzítés/módosítás:** Name=tsmiAdatrogzitesModositas

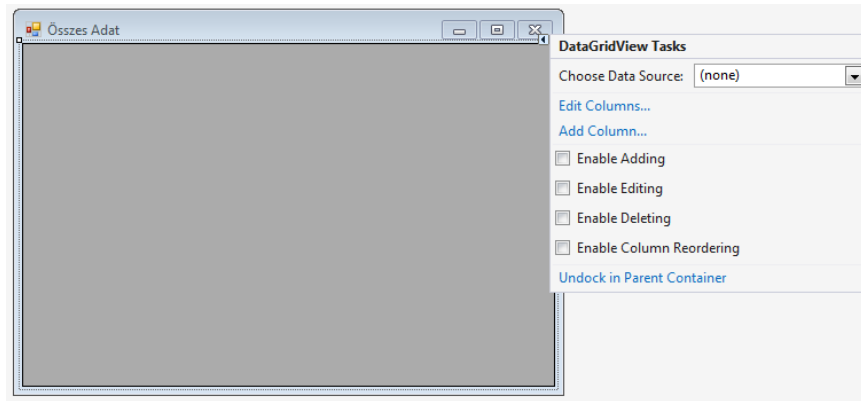
**Adatrögzítés:** Name=tsmiAdatrogzites

**Módosítás:** Name=tsmiModositas



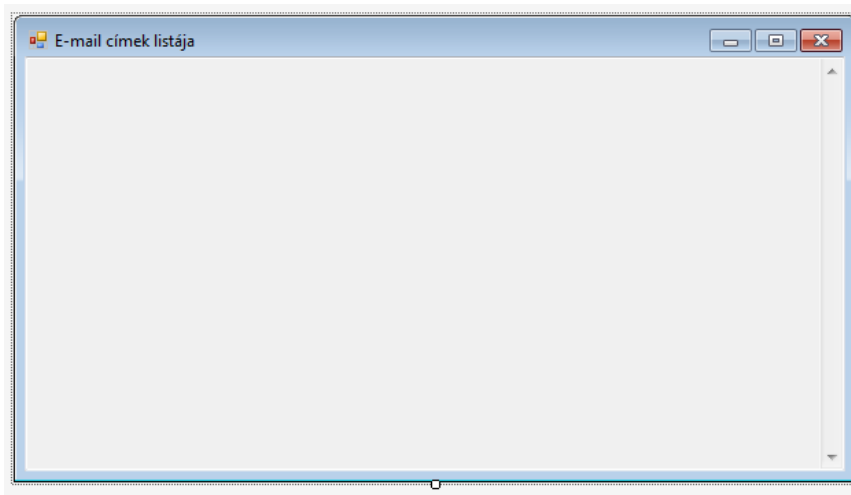
194. ábra.

Hozzunk létre egy új formot (Project menü, Add Windows Form..., Templates: Windows Form, Name=*frmOsszesAdat.cs*) *frmOsszesAdat* néven, Text="Összes Adat". Helyezzünk el rajta egy DataGridView komponenst Name=*dgvRacs*, Dock=Fill. A hozzáadás, szerkesztés és törlés legyen letiltva.



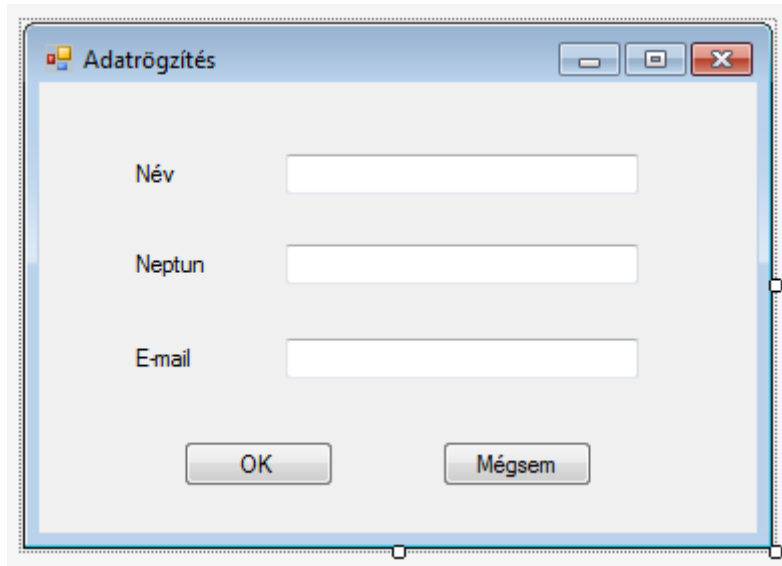
195. ábra.

Hozunk létre egy formot *frmEmailLista.cs* néven az e-mail címek megjelenítésére *Name=frmEmailLista*, *Text="E-mail címek listája"*. Helyezzünk el egy szerkesztőmezőt (Text Box) a formon. *Name=tbEmail*, *Text=""*, *Dock=Fill*, *ReadOnly=True*, *ScrollBars=Vertical*, *Multiline=True*.



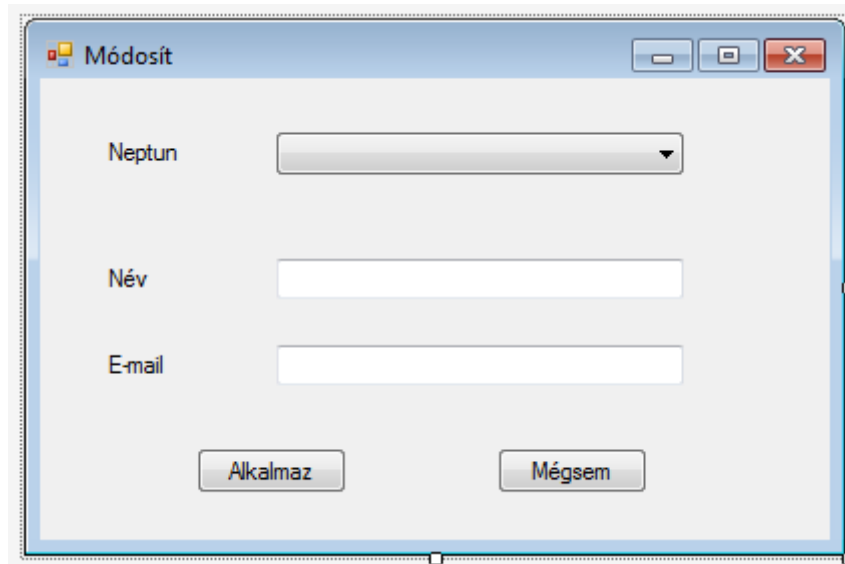
196. ábra.

Hozunk létre egy új formot (Project menü, Add Windows Form..., Templates: Windows Form, *Name= frmAdatrogzites.cs*). Az ablak osztályának neve *frmAdatrogzites* legyen. A form fejlécébe helyezzük el az „Adatrögzítés” szöveget. Az ablak tartalmazzon három címkét (Label): Név, Neptun, E- mail felirattal, három szerkesztőmezőt (TextBox): *Name=tbNev*, *Name=tbNeptun*, *Name=tbEmail* néven. Helyezzünk el két nyomógombot (Button) a formon: *Name=btOK*, *DialogResult=OK*, *Text=OK*, és *Name=btMegsem*, *Text=Mégsem*, *DialogResult= Cancel*.



197. ábra.

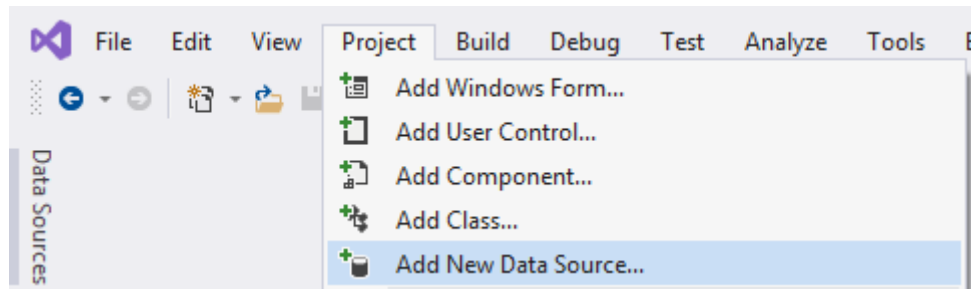
Hozzunk létre egy új formot (Project menü, Add/ Windows Form..., Templates: Windows Form, Name= *frmModosit.cs*). Az ablak osztályának neve *frmModosit* legyen. A form fejlécébe helyezzük el a „Módosít” szöveget. Az ablak tartalmazzon három címkét (Label): Név, Neptun, E-mail felirattal, két szerkesztőmezőt (TextBox): Name=*tbNev*, Name=*tbEmail* néven, és egy kombinált listaablakot (ComboBox): Name=*cbNeptun*, DropDownStyle=DropDownList. Helyezzünk el két nyomógombot (Button) a formon Name=*btAlkalmaz*, Text=Alkalmaz, és Name=*btMegsem*, Text=Mégsem, DialogResult= Cancel



198. ábra.

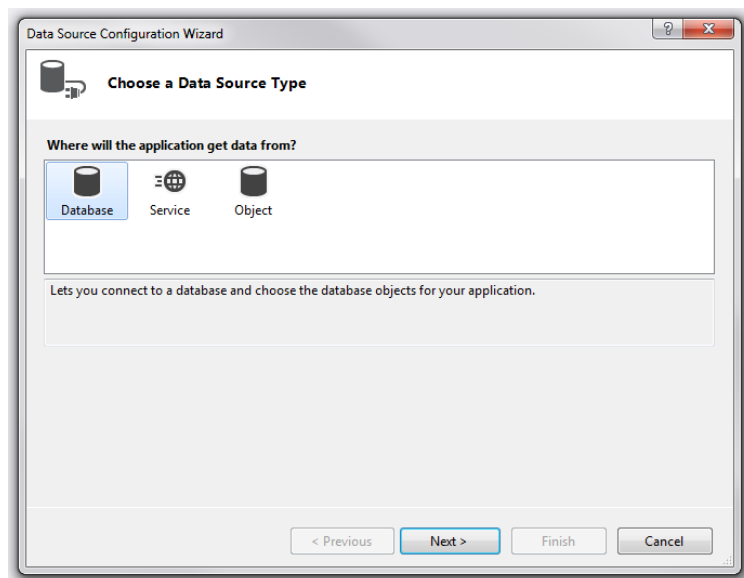
### 6.1.2. Adatforrás megadása és a típusos adatkezelő osztályok legenerálása

Másoljuk be a *list.mdb* állományt a projektünk könyvtárába. A Visual Studio Project menüjében válasszuk az Add/ New Data Source...-t. A varázslóban válasszuk a DataBase, Next, DataSet, Next, New Connection ... -t

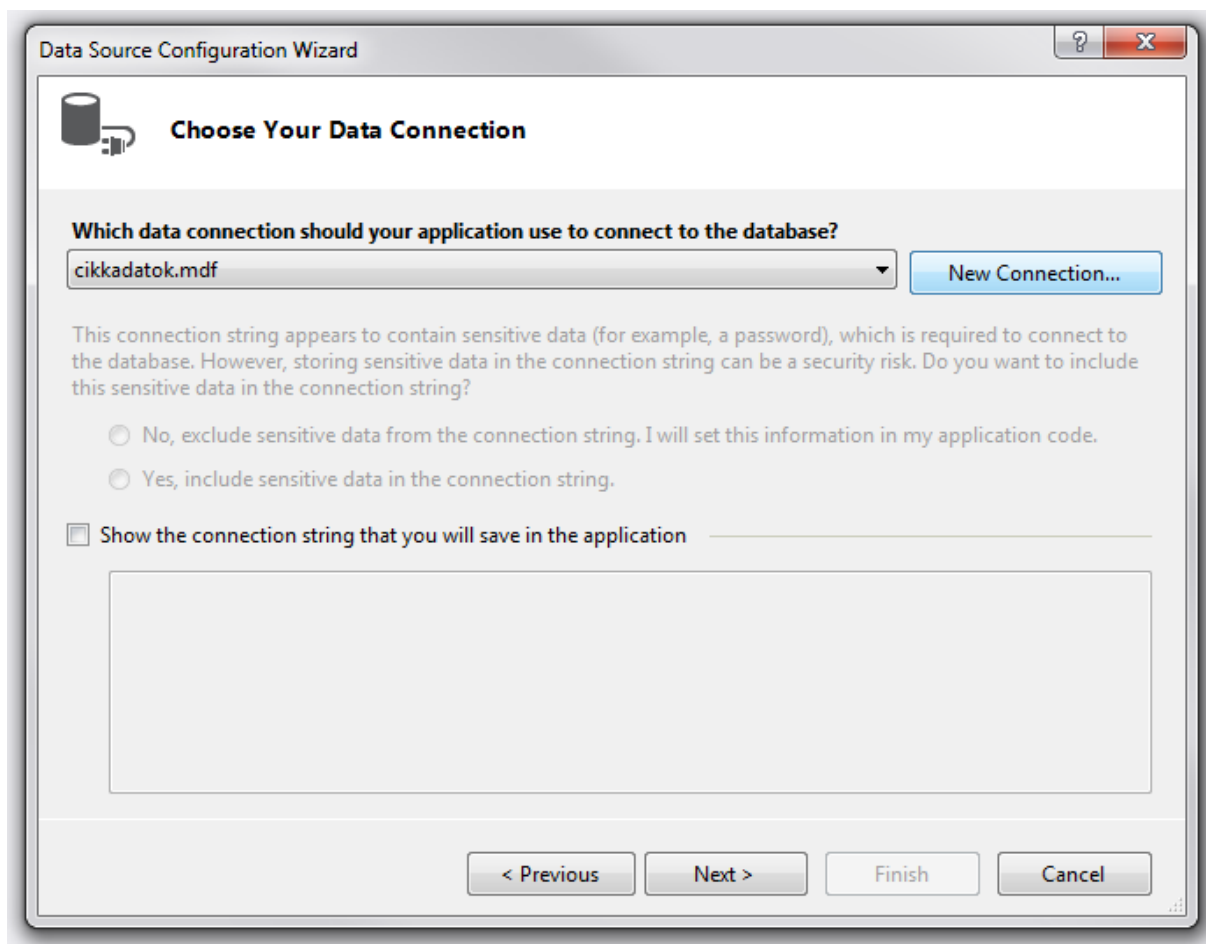


199. ábra.

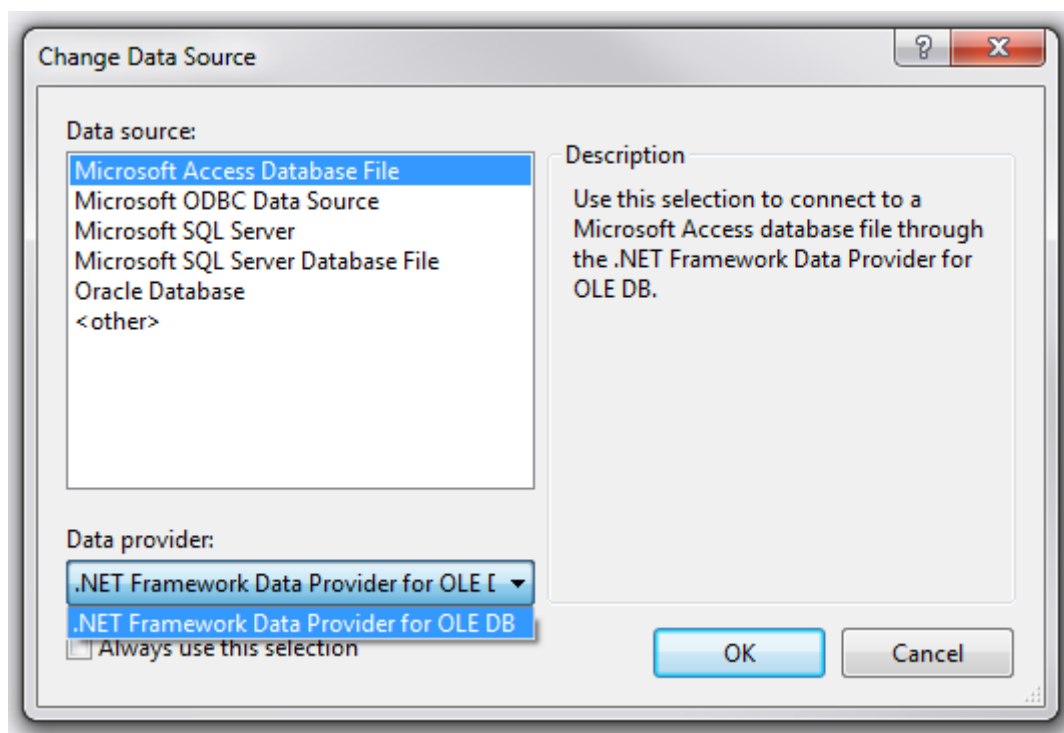
A felkínált Data Source típusokból választuk a Database-t majd a Dataset-et majd pedig adjunk hozzá egy új kapcsolatot. A Data Source *mellett Change után* állítsuk be a Microsoft Access Database File-t, majd Data Providernél a .NET Framework Data Provider for OLE DB-t. (Fontos, hogy amennyiben .accdb Access adatbázist használnánk, abban az esetben Data Source alatt <other> és Data Provider-nél szintén a .NET Framework Data Provider for OLE DB lehetőséget választjuk. A segédletben alkalmazott Microsoft Access Database File lehetőség csak .mdb kiterjesztés esetén működik.)



200. ábra



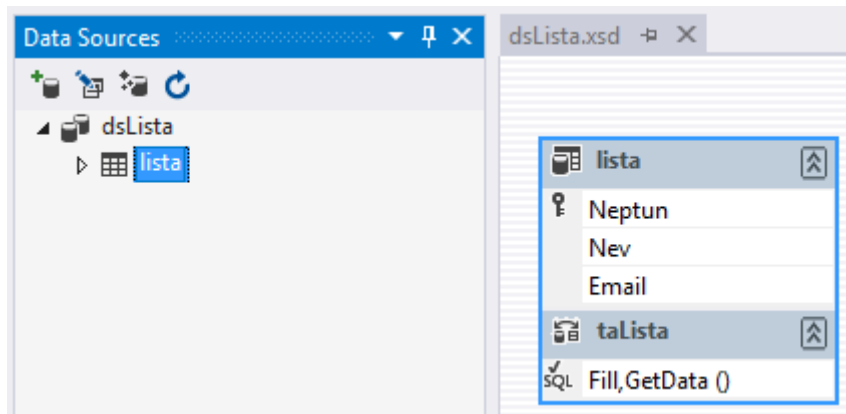
201. ábra



202. ábra.

OK után a felugró ablakban tallózzuk ki a projektbe bemásolt list.mdb fájlt.

A DataSet neve legyen *dsLista*. A Designerben állítsuk át a tábla nevét *lista*-ra, és a táblaadapter nevét *taLista*-ra.



203. ábra.

A típusos DataSet létrehozásakor a Visual Studio egy sor osztályt generál az adatbázis és a benne lévő tábla alapján. Nézzük meg őket a Class View ablakban.

### 6.1.3. A kód elkészítése

#### A főablak

Hozunk létre egy-egy adattagot az adatkezeléshez szükséges objektumok (DataSet, TableAdapter) számára, majd a konstruktorban hozzuk létre az objektumokat.

```

/// <summary>
/// Az adatok memória beli tárolására szolgáló típusos dataset.
/// </summary>
dsLista dsLista;
/// <summary>
/// A lista tábla típusos TableAdptere.
/// </summary>
taLista taLista;
/// <summary>
/// A főablak konstruktora.
/// </summary>
public frmFoablak()
{
    InitializeComponent();
    //DataSet objektum létrehozása. Itt fogjuk tárolni lokálisan az //adatokat.
    dsLista = new dsLista();
    //Table Adapter objektum létrehozása. Ez gondoskodik majd a //tábla feltöltéséről
    és a későbbi szinkronizálásról az //adatbázissal.
    taLista = new taLista();
    try
    {
        //A memóriabeli lista tábla feltöltése adatbázissal.
        taLista.Fill(dsLista.lista);
    }
    catch(Exception e)
    {
        MessageBox.Show("Nem sikerült megnyitni az adatbázis állományt!\r\n" +
            e.Message, "Hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    tsmiMentes.Enabled = false;
}

```

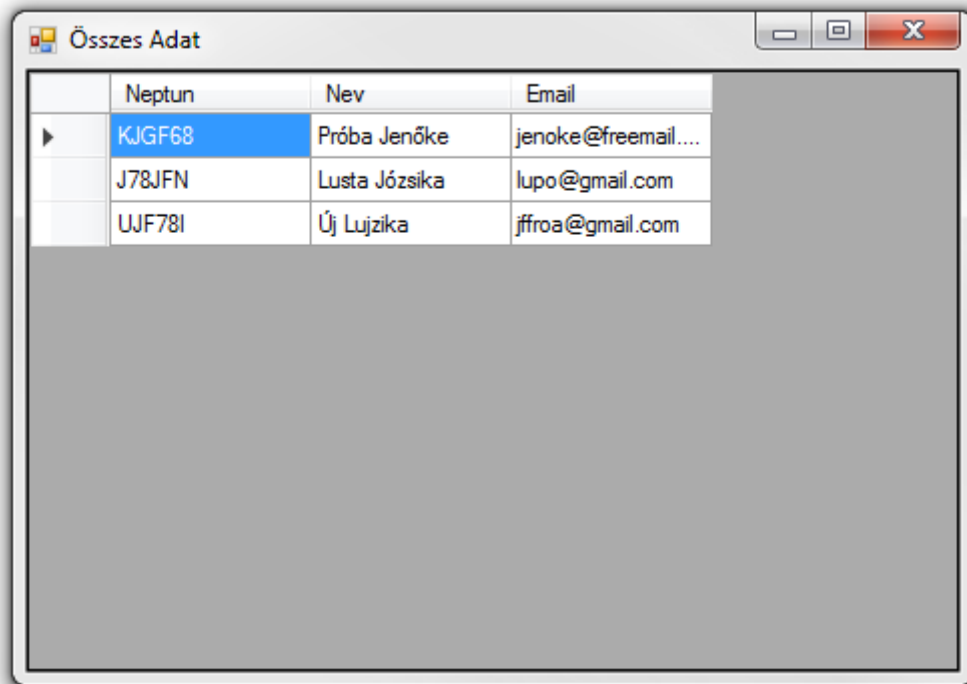
## Összes adat menüpont

Az összes adat egyszerre történő megjelenítése egy DataGrid komponens segítségével az frmOsszesAdat formon történik. Az adatok megjelenítéséhez létre kell hoznunk az adatkötést, azaz a lista táblát meg kell adnunk adatforrásként az adatrács számára. Mivel az adatrács komponens alapból private hozzáférésű, és így a főablak osztályából nem tudnánk módosítani a DataSource tulajdonságának értékét, ezért tervezési nézetben először állítsuk a komponens hozzáférését internal-ra (Modifiers=Internal).

```

/// <summary>
/// Lekérdezés/Összes Adat menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tsmiOsszesAdat_Click(object sender, EventArgs e)
{
    //Csak megjelenítésre használt párbeszédablak. Létrehozzuk a //form objektumot.
    frmOsszesAdat foa = new frmOsszesAdat();
    //A formon lévő adatrácsban beállítjuk az adatforrást, a lista //táblából vegye
    az adatokat.
    foa.dgvRacs.DataSource = dsLista.lista;
    //Modálisan megjelenítjük a formot (párbeszédablakot).
    foa.ShowDialog();
}

```



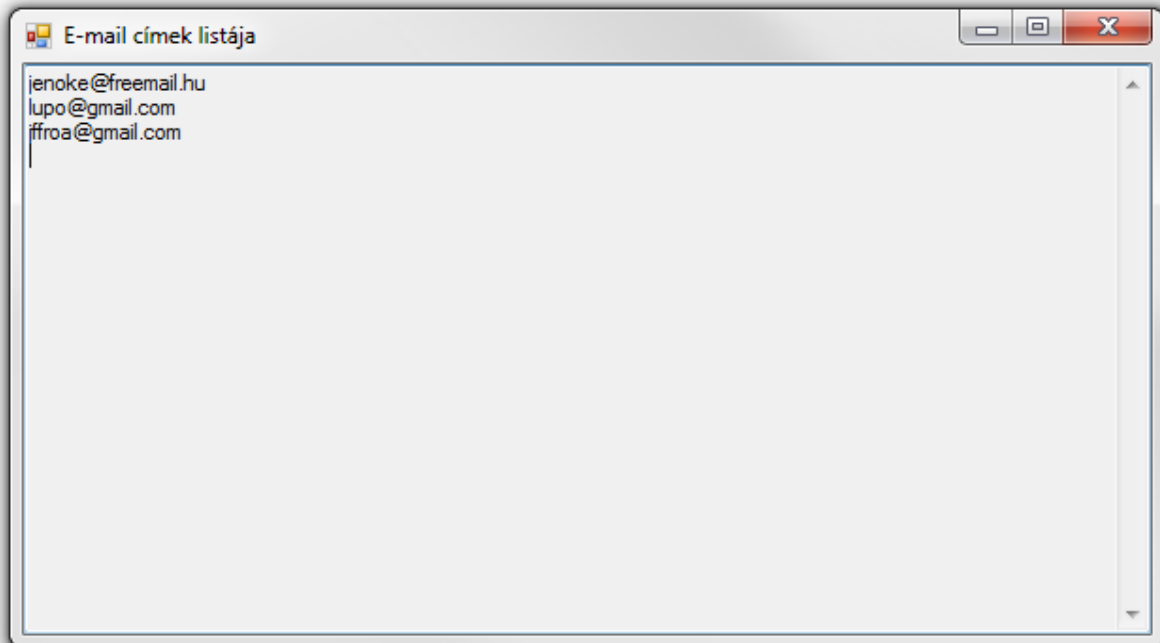
204. ábra.

## E-mail címek listája menüpont

Az e-mail címek megjelenítése egy TextBox komponens segítségével az frmEmailLista formon történik. Az adatok megjelenítéséhez először készítünk egy sztringet, amiben összegyűjtjük a címeket, majd a sztring tartalmát megjelenítjük a formon.

Mivel a szövegmező komponens alpból private hozzáférésű, így a főablak osztályából nem tudnánk módosítani a DataSource tulajdonságának értékét, ezért tervezési nézetben először állítsuk a komponens hozzáférését internal-ra (Modifiers=Internal).

```
/// <summary>
/// Lekérdezés/E-mail címek listája menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tsmiEmailLista_Click(object sender, EventArgs e)
{
    //Csak megjelenítésre használt párbeszédablak. Létrehozzuk a //form objektumot.
    frmEmailLista fel = new frmEmailLista();
    //Létrehozunk egy sztringet, amiben összeállítjuk a //megjeleníteni kívánt
    //tartalmat.
    string s = "";
    //Egyesével hozzáadjuk az email címeket.
    foreach(dsLista.listaRow sor in dsLista.lista.Rows)
        s += sor.Email + "\r\n";
    //A sztring tartalmát elhelyezzük a textboxban
    fel.tbEmail.Text = s;
    //Modálisan megjelenítjük a formot(párbeszédablakot).
    fel.ShowDialog();
}
```



205. ábra.



## Adatrögzítés menüpont

Az frmAdatrogzites formon elhelyezett szövegmezők segítségével fogjuk bekérni a felhasználótól az adatokat. Mindhárom komponens alából private hozzáférésű, ezért az ablak osztályán kívülről nem érhető el. Amennyiben a főablak osztályának egy metódusából szeretnénk kiolvasni ezen mezők tartalmát, két megoldás közül választhatunk. Vagy internal hozzáférésűvé tesszük a komponenseket, mint az előző két form esetében, vagy az frmAdatrogzites osztályon belül készítünk egy-egy tulajdonságot a három szerkesztőmezőhöz, amin keresztül lekérdezhető a bennük tárolt szöveg a formon kívülről is. Most ezen második megoldást fogjuk megvalósítani.

```

/// <summary>
/// Neptun kód.
/// </summary>
public string Neptun
{
    get { return tbNeptun.Text; }
}
/// <summary>
/// Név.
/// </summary>
public string Nev
{
    get { return tbNev.Text; }
}
/// <summary>
/// E-mail cím.
/// </summary>
public string Email
{
    get { return tbEmail.Text; }
}

```

Set elérők most nem szükségesek. A formon szintaktikailag ellenőriznünk kell a bevitt adatok helyességét. Ennek érdekében mindhárom szövegmező esetén készítünk eseménykezelőt a Validating eseményhez.

```

/// <summary>Név ellenőrzése. summary>
/// <param name="sender">A tbNev TextBox.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tbNev_Validating(object sender, CancelEventArgs e)
{
    if(Nev.Length == 0) //Ha nincs beírva név.
    {
        e.Cancel = true;
        MessageBox.Show("A Név mező kitöltése kötelező!", "Hiba",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    foreach(char c in Nev.ToCharArray())//Csak betű és szóköz elfogadott.
    {
        if(!Char.IsLetter(c) && c != ' ')
        {
            e.Cancel = true;
            MessageBox.Show("Csak betű és szóköz elfogadott!", "Hiba",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
    }
}
}

```

```

/// <summary>
/// Neptun kód ellenőrzése.
/// </summary>
/// <param name="sender">A tbNeptun TextBox.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tbNeptun_Validating(object sender, CancelEventArgs e)
{
    //Ha a felhasználó nem adott meg Neptun kódot.
    if (Neptun.Length == 0)
    {
        e.Cancel = true;
        MessageBox.Show("A Neptun mező kitöltése kötelező!", "Hiba",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    //Csak betű és pont megadása megengedett.
    foreach (char c in Neptun.ToCharArray())
    {
        if (!Char.IsLetter(c) && c != '.')
        {
            e.Cancel = true;
            MessageBox.Show("Csak betű és pont megadása lehetséges!", "Hiba",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
    }
}
/// <summary>
/// E-mail cím ellenőrzése.
/// </summary>
/// <param name="sender">A tbEmail TextBox.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tbEmail_Validating(object sender, CancelEventArgs e)
{
    //Az e-mail címet reguláris kifejezéssel ellenőrizzük.
    if(!Regex.IsMatch(Email, @"^([\w-\.]*)@(\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.|\)|" + @"(([\w-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\?)$"))
    {
        MessageBox.Show("A megadott e-mail cím érvénytelen!", "Adatbeviteli hiba",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        e.Cancel = true;
    }
}
}

```

Visszatérve a főablakhoz az Adatrögzítés menüpont eseménykezelője az alábbi.

```

/// <summary>
/// Adatrögzítés menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tsmiAdatrogzites_Click(object sender, EventArgs e)
{
    //Adatbevitelre használt párbeszédablak. Létrehozzuk a form //objektumot.
    frmAdatrogzites fa = new frmAdatrogzites();
    //Megjelenítjük a formot
    if(fa.ShowDialog() == DialogResult.OK)
    {

```

```

//Ha OK gombbal zárta be a felhasználó az //párbeszédablakot, akkor kiolvassuk
a három adatot, //létrehozunk egy új sort a táblában, és elhelyezzük benne //a
kiolvasott adatokat.
dsLista.lista.AddlistaRow(fa.Neptun, fa.Nev, fa.Email);
//Figyelem!Az Neptun mező egyediségét nem ellenőriztük. Ha a //felhasználó egy
már korábban bevitt Neptun kódot ad meg, //akkor kivétel keletkezik, és a
program leáll.
//Önálló otthoni feladat: oldja meg az ellenőrzést!
//Módosítottuk a tábla tartalmát, mentés szükséges lehet.
tsmiMentes.Enabled = true;
}
}

```

## Módosítás menüpont

A Módosítás menüponthoz kapcsolódó frmModosit form megvalósítása kicsit bonyolultabb mint, az adatrögzítésnél alkalmazott társáé. Mivel itt egy létező készletből kell kiválasztani a módosítani kívánt rekordot, ezért az Neptun kód (elsődleges kulcs) mezője nem szövegmezővel, hanem kombinált listaablakkal lett megvalósítva. A párbeszédablak megjelenítését követően a felhasználó választja ki, hogy melyik rekordot akarja módosítani, ezért a formból hozzá kell férnünk a teljes táblához. Ezt úgy oldjuk meg, hogy az frmMódosít osztályban létrehozunk egy tulajdonságot, aminek értéket adva beállíthatjuk a ComboBox adatforrását (adatkötés létrehozása).

```

/// <summary>
/// Tulajdonság a memóriabeli adattábla elérésének támogatásához.
/// </summary>
public dsLista.listaDataTable dtLista
{
    //Ha lekérdezik az értéket
    get
    {
        //Visszaadjuk a ComboBox adatforrásaként megadott tábla //referenciáját.
        return (dsLista.listaDataTable)cbNeptun.DataSource;
    }
    //Ha értéket adnak a tulajdonságnak
    set
    {
        //Beállítjuk a táblát a ComboBox adatforrásaként.
        cbNeptun.DataSource = value;
        //Beállítjuk a ComboBox-ban megjelenő mezőt.
        cbNeptun.DisplayMember = "Neptun";
    }
}
}

```

Emellett az frmModosit osztályban létrehozunk egy adattagot, amiben nyílván fogjuk tartani a tábla aktuális rekordját (lrAktualis), valamint készítünk egy adattag/tulajdonság párost annak követésére, hogy módosított-e valamit a felhasználó. Erre azért lesz szükség, mert ettől függően engedélyezzük a főablak Mentés menüpontját. Kezdőértékét hamisra állítjuk a konstruktorban.

```

/// <summary>
/// Az aktuális rekord a táblában.
/// </summary>
private dsLista.listaRow lrAktualis;
/// <summary>
/// Nyilvántartja, hogy történt-e módosítás a Módosít ablak
/// segítségével.
/// </summary>

```

```
public bool voltValtozas{ get; set; }
/// <summary>
/// A módosít ablak konstruktora.
/// </summary>
public frmModosit()
{
    InitializeComponent();
    //Kezdetben "nem volt változás".
    voltValtozas = false;
}
```

Ha a felhasználó egy új Neptun kódot választ ki a listából, akkor a hozzá tartozó név és e-mail értékeket bemásoljuk a két szerkesztőmezőbe. Ehhez a ComboBox SelectedIndexChanged eseményéhez készítünk eseménykezelőt.

```
/// <summary>
/// A ComboBox-ban változott a kiválasztott Neptun kód.
/// </summary>
/// <param name="sender">A ComboBox objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void cbNeptun_SelectedIndexChanged(object sender, EventArgs e)
{
    //Megkeressük, hogy melyik az aktuális rekord a táblában.
    var sor = from x in dtLista
    where x.Neptun == ((string)((DataRowView)cbNeptun.SelectedItem).Row[0])
    select x;
    if (sor.Count() > 0)
    {
        //Ha megvan az aktuális rekord, tároljuk a sor objektum referenciáját.
        lrAktualis = sor.ElementAt(0);
        //Bemásoljuk a TextBox-ba a nevet.
        tbNev.Text = lrAktualis.Nev;
        //Bemásoljuk a TextBox-ba az e-mail címet.
        tbEmail.Text = lrAktualis.Email;
    }
}
```

A módosított értékeket itt is ellenőriznünk kell hasonlóan az adatrögzítés esetéhez. Másoljuk le egyszerűen az ott használt két eseménykezelőt (tbEmail\_Validating és tbNév\_Validating), majd őket a megfelelő szövegmezők Validating eseményéhez (Properties ablak). Az Alkalmaz nyomógombon történő kattintás hatására a memóriabeli táblában módosítani kell a név és e-mail mezők tartalmát a szövegmezőkben megadottak szerint.

```
/// <summary>
/// Kattintás az Alkalmazás gombon.
/// </summary>
/// <param name="sender">A nyomógomb.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void btAlkalmaz_Click(object sender, EventArgs e)
{
    if(lrAktualis != null)
    {
        //Ha van aktuális rekord kiválasztva a táblában. //Tároljuk a táblában az új nevet.
        lrAktualis.Nev = tbNev.Text;
        //Tároljuk a táblában az új e-mail címet.
        lrAktualis.Email = tbEmail.Text;
        //Módosítást hajtottunk végre a táblában.
        voltValtozas = true;
    }
}
```

```
}
```

Visszatérve a főablak osztályába most már elkészíthetjük a Módosítás menüpont eseménykezelőjét.

```
/// <summary>
/// Módosítás menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tsmiModositas_Click(object sender, EventArgs e)
{
    //Adatbevitelre használt párbeszédablak.
    //Létrehozuk a form objektumot.
    frmModosit fm = new frmModosit();
    // Átadjuk a form számára az adattáblára irányuló referenciát, //így a
    //párbeszédablakból közvetlenül elérhető a tábla tartalma.
    fm.dtLista = dsLista.lista;
    //Megjelenítjük a formot.
    fm.ShowDialog();
    //Ha volt változás a táblában.
    if (fm.voltValtozas)
    {
        //Módosítottuk a tábla tartalmát, mentés szükséges lehet.
        tsmiMentes.Enabled = true;
    }
}
}
```

### Mentés menüpont

A Mentés menüpont feladata az, hogy a memóriában tárolt táblát szinkronizálja az adatbázisban tárolt táblával, azaz érvényesítse a felhasználó által végrehajtott módosításokat. Mivel erre a funkcióra a kilépéshez kapcsolódóan is szükségünk lesz, ezért a feladatot megvalósító utasításokat egy külön metódusban helyezük el.

```
/// <summary>
/// Szinkronizálja a memóriában található táblát az adatbázissal,
/// azaz az adatfelviteleket és módosításokat érvényesíti az
/// adatbázisban.
/// </summary>
private void Mentes()
{
    try
    {
        //Szinkronizáljuk a memóriában található táblát az //adatbázissal, azaz az
        //adatfelviteleket és módosításokat //érvényesítjük az adatbázisban.
        taLista.Update(dsLista);
        //Mentés nem szükséges.
        tsmiMentes.Enabled = false;
    }
    catch(Exception e)
    {
        MessageBox.Show("Nem sikerült a szinkronizálási művelet!\r\n"+e.Message,
            "Hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
/// <summary>
/// Mentés menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
```

```
private void tsmiMentes_Click(object sender, EventArgs e)
{
    //Szinkronizáljuk a memóriában található táblát az //adatbázissal, azaz az
    //adatfelviteleket és módosításokat //érvényesítjük az adatbázisban.
    Mentes();
}
```

### Kilépés menüpont

A kilépés végrehajtása előtt amennyiben engedélyezett a mentés (történt módosítás a táblában), felkínáljuk a felhasználónak a mentés lehetőségét.

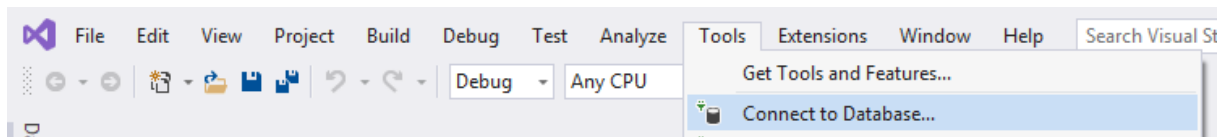
```
/// <summary>
/// Kilépés menüpont.
/// </summary>
/// <param name="sender">A menüpont objektum.</param>
/// <param name="e">Kiegészítő paraméterek.</param>
private void tsmiKilepes_Click(object sender, EventArgs e)
{
    if (tsmiMentes.Enabled)
    {
        // Ha a mentés menüpont engedélyezett, azaz történt módosítás a // legutóbbi
        // szinkronizálás óta, akkor felkínáljuk a
        // felhasználónak a mentés lehetőségét.
        if (MessageBox.Show("Kívánod menteni a
        változásokat?", "Kérdés", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
        DialogResult.Yes)
        {
            //Ha a felhasználó akar menteni, akkor //szinkronizálunk.
            Mentes();
        }
    }
    Application.Exit();
}
```

## 6.2. Adatbáziselérés ODBC-n keresztül utasításokkal, C#-ban

### 6.2.1. Előkészítés – Access adatbázis lemásolása, ODBC DSN létrehozása

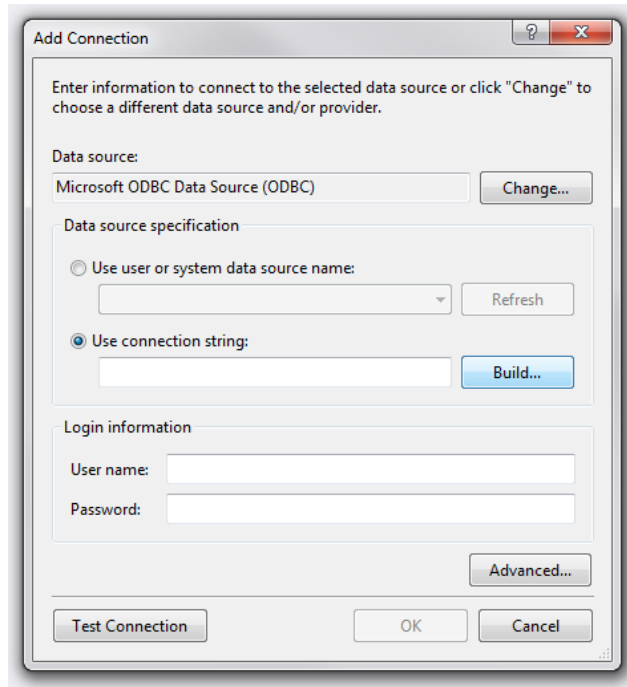
Másoljuk le az alábbiakat: list.mdb

Indítsuk el a Visual Studio 2019-et. Válasszuk ki a Tools menü Connect to Database menüpontját. A Data Source részben válasszuk a Microsoft ODBC Data Source (ODBC)-t, majd kattintsunk az OK gombon.



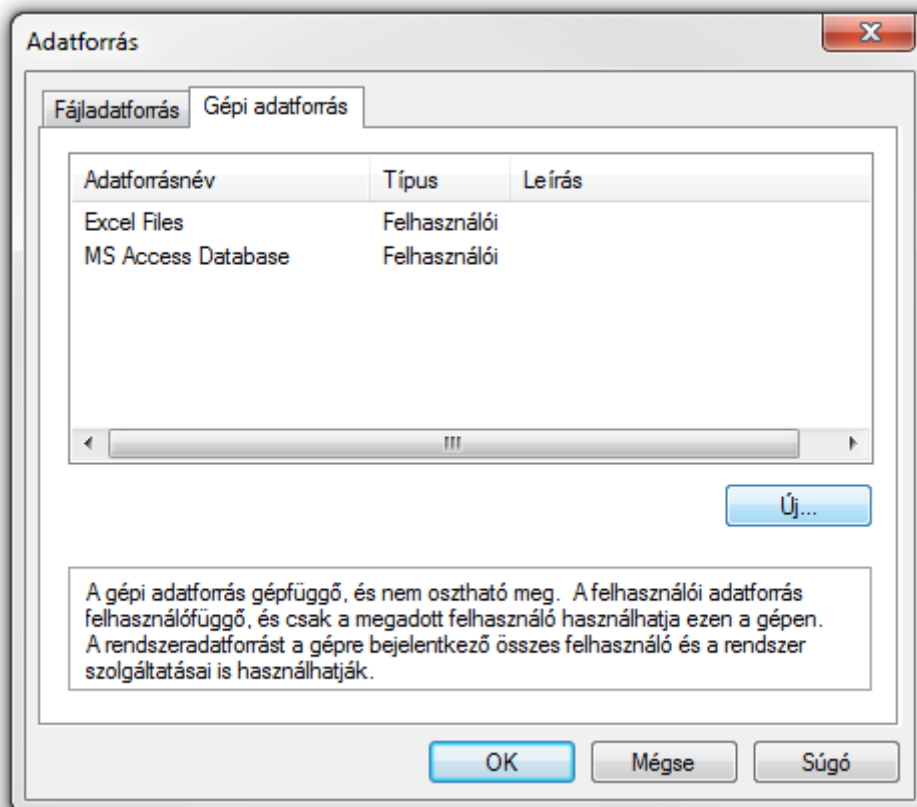
206. ábra.

Az Add Connection ablak Data source specification csoportjában válasszuk ki a Use connection string-et, majd kattintsunk a Build... gombon.



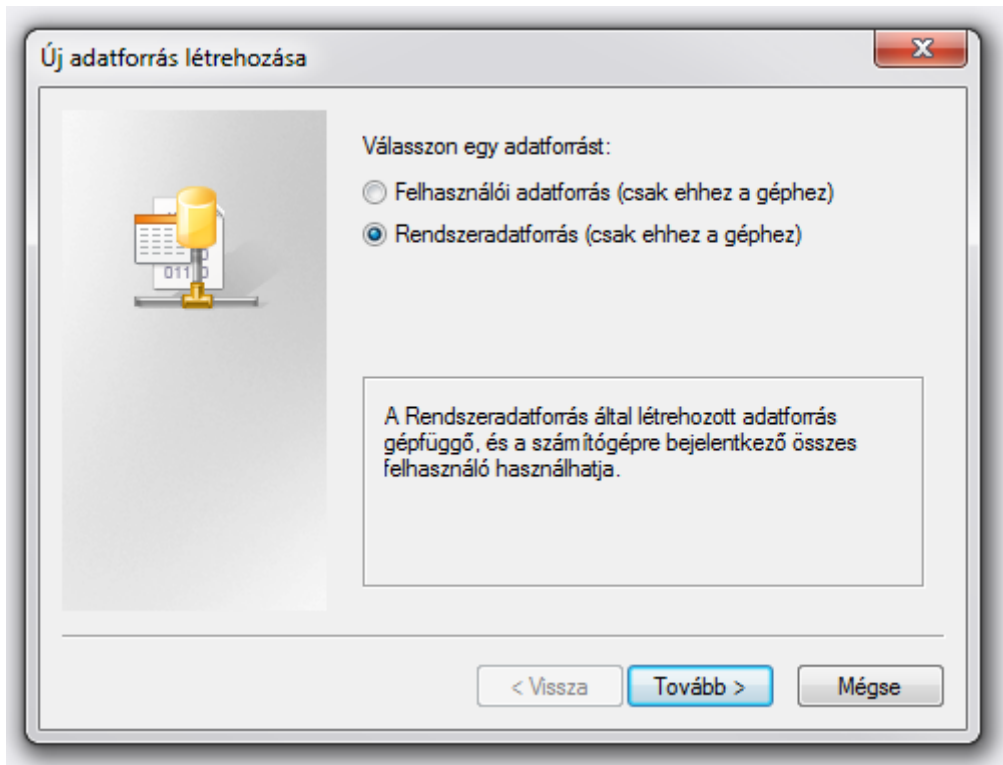
207. ábra.

A megjelenő *Select Data Source* párbeszédpanelen válasszuk ki a *Machine Data Source* fület, majd kattintsunk a *New...* gombon.



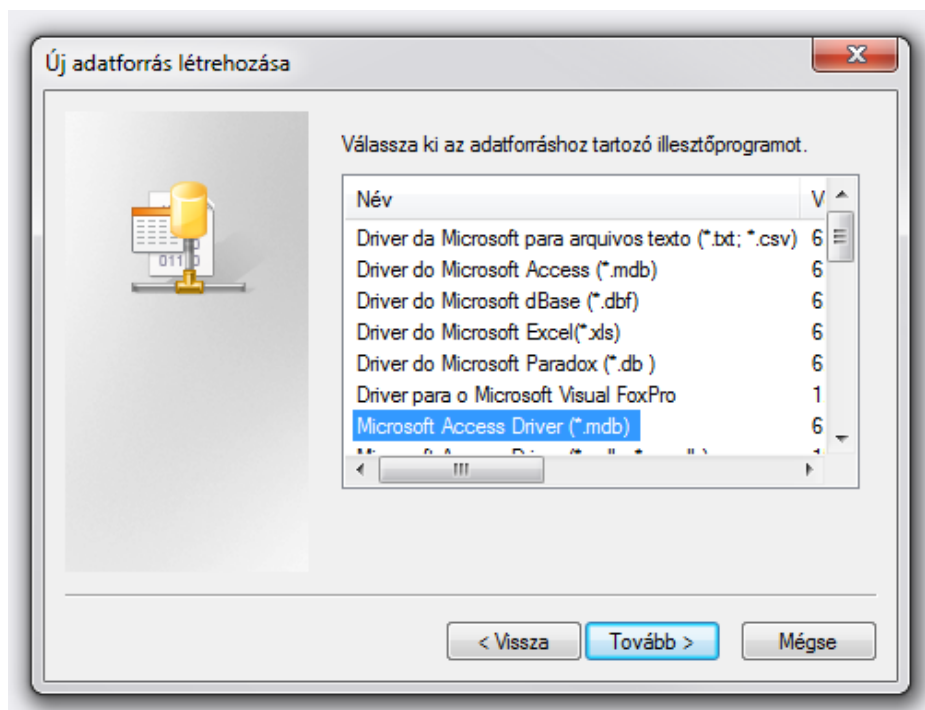
208. ábra.

Itt válasszuk ki a *System Data Source*-t, majd kattintsunk a *Next* gombon.



209. ábra.

A megjelenő listában válasszuk ki a *Microsoft Access Driver (\*.mdb)* -t, majd kattintsunk a *Next gombon*, majd a *Befejezés gombon*.

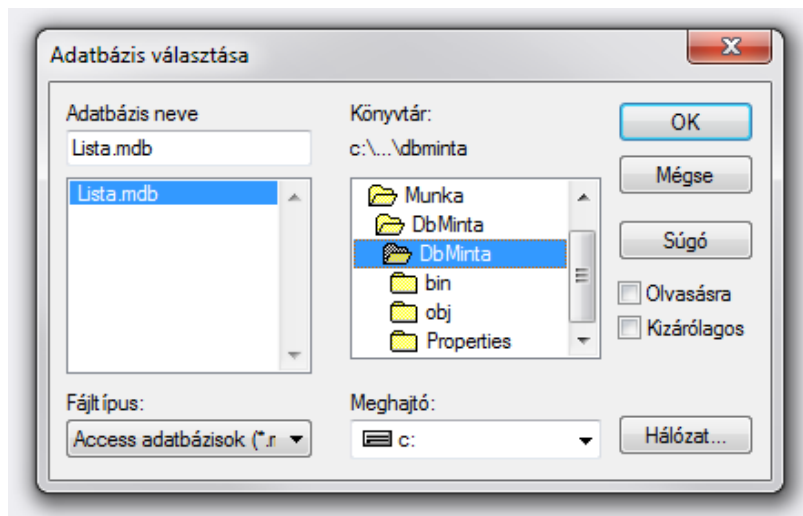


210. ábra.

Az *ODBC Microsoft Access Setup* ablakban a *Database* csoportban kattintsunk a *Select... gombon*. A felbukkanó párbeszédablakban válasszuk ki az előzőekben lemásolt *list.mdb*



állományt. Ezután *Data Source Name*-ként adjuk meg az *swt1-et*, és *kattintsunk az OK gombon*, majd zárjuk be OK-val az *Select Data Source* ablakot.



211. ábra.

Nem kell azonosítót és jelszót megadni. Ezt követően zárjuk be OK-val az *Add Connection* ablakot. Kattintsunk a *Test Connection gombon* az *Add Connection* ablakban a kapcsolat ellenőrzése érdekében, majd az *OK gombbal* zárjuk be az ablakot. Ekkor a *Server Explorer* ablakban a *DataConnections* mappában egy új kapcsolat jelenik meg.



212. ábra.

### 6.2.2. Alkalmazás létrehozása

Hozunk létre egy Windows Forms Application típusú C# alkalmazást a *C:\munka* könyvtárban *DbMinta* néven. A form neve legyen *frmFoablak*, az őt tartalmazó állomány neve legyen *frmFoablak.cs*, az ablak felirata legyen: *Adatbáziskezelés ODBC-n keresztül*.

Helyezzünk el a formon egy menüt (*MenuStrip*), aminek a neve legyen: *msFomenü*.  
Menüpontok:

**Kilépés:** Name=*tsmiKilepes*.

Készítsen a *Kilépés* menüpont *Click* eseményéhez egy eseménykezelőt, és helyezze el abban a kilépést biztosító utasítást: `Application.Exit();`

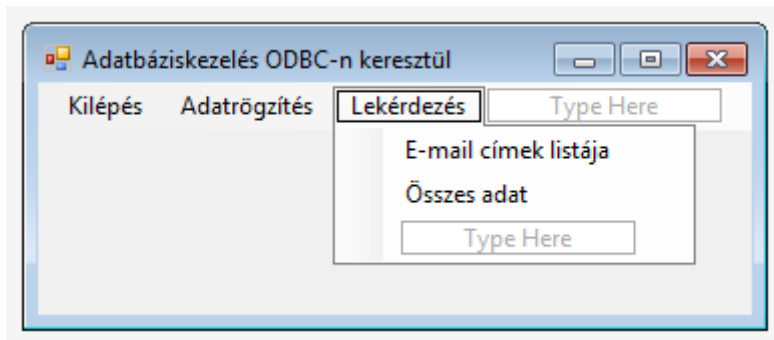
**Adatrögzítés:** Name=*tsmiAdatrogzites*, eseménykezelő: lásd később.

**Lekérdezés:** Name=*tsmiLekerdezes*,

Legyen benne két almenüpont:

**Email címek listája:** Name=*tsmiEmailCimekListaja*

**Összes adat:** Name=*tsmiOsszesAdat*

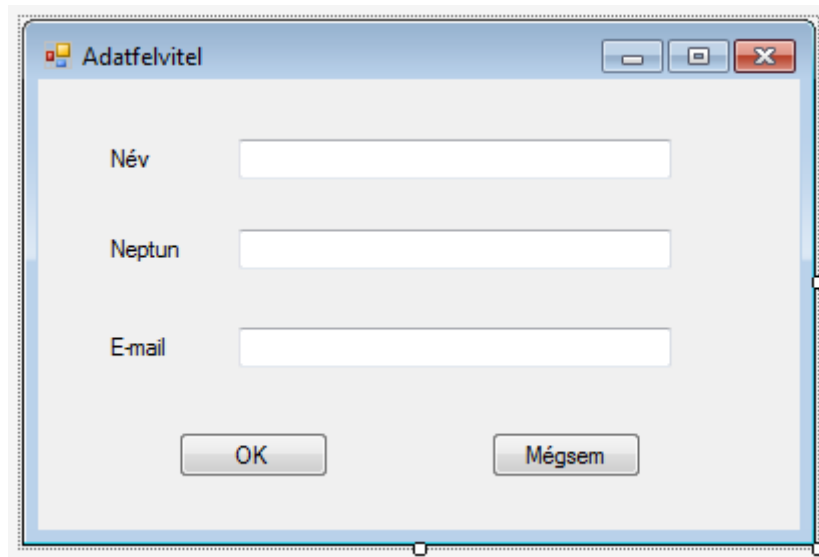


213. ábra.

Az frmFooablak Form kódjában helyezzük el:

```
using System.Data.Odbc;
```

Hozunk létre egy új formot (Project menü, Add Windows Form..., Templates: Windows Form, Name=frmFelvitel.cs). Tartalmazzon három címkét (Label): Név, Neptun, e-mail felirattal, három szerkesztőmezőt (TextBox): Name=tbNev, Name=tbNeptun, Name=tbEmail néven. Helyezzünk el két nyomógombot (Button) a formon Name=btOK, Text=OK, DialogResult=OK és Name=btMegsem, Text=Mégsem, DialogResult=Cancel. A form fejlécébe helyezzük el az „Adatfelvitel” szöveget.



214. ábra.

Az frmFelvitel kódjában helyezzük el:

```
using System.Text.RegularExpressions;
```

Készítsünk egy-egy tulajdonságot a három szerkesztőmezőhöz, amin keresztül beállítható és lekérdezhető a bennük tárolt szöveg a formon kívülről is.

```
public string Neptun  
{  
    get { return tbNeptun.Text; }  
}  
public string Nev  
{  
    get { return tbNev.Text; }  
}
```

```
public string Email
{
    get { return tbEmail.Text; }
}
```

Készítsünk egy ellenőrző eseménykezelőt az frmFelvitel tbNev szerkesztőmezőjéhez, ami a mező elhagyásakor (Validating esemény) aktivizálódik. Csak betűt és szóközt fogadhat el.

```
private void tbNev_Validating(object sender, CancelEventArgs e)
{
    string nev = tbNev.Text;
    for(int i = 0; i<nev.Length; i++)
    {
        if(!Char.IsLetter(nev[i]) && nev[i] != ' ')
        {
            e.Cancel = true;
            MessageBox.Show("A névben csak betű és szóköz állhat!", "Adatbeviteli hiba",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
    }
}
```

Készítsünk egy ellenőrző eseménykezelőt az frmFelvitel tbNeptun szerkesztőmezőjéhez, ami a mező elhagyásakor (Validating esemény) aktivizálódik. Csak betűt és pontot fogadhat el.

```
private void tbNeptun_Validating(object sender, CancelEventArgs e)
{
    string neptun = tbNeptun.Text;
    for (int i = 0; i < neptun.Length; i++)
    {
        if (!Char.IsLetter(neptun [i]) && neptun[i] != '.')
        {
            e.Cancel = true;
            MessageBox.Show("Az Neptun kódban csak betű és pont állhat!", "Adatbeviteli hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
            break;
        }
    }
}
```

Készítsünk egy ellenőrző eseménykezelőt az frmFelvitel tbEmail szerkesztőmezőjéhez, ami a mező elhagyásakor (Validating esemény) aktivizálódik:

```
private void tbEmail_Validating(object sender, CancelEventArgs e)
{
    string email = tbEmail.Text;
    if (!Regex.IsMatch(email, @"^([\w-\.]*)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.|\.))" + @"(([\w-]+\.)+)([a-zA-Z]{2,4}|[0-9]{1,3})(\?)?$"))
    {
        MessageBox.Show("A megadott e-mail cím érvénytelen!", "Adabeviteli hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
        e.Cancel = true;
    }
}
```

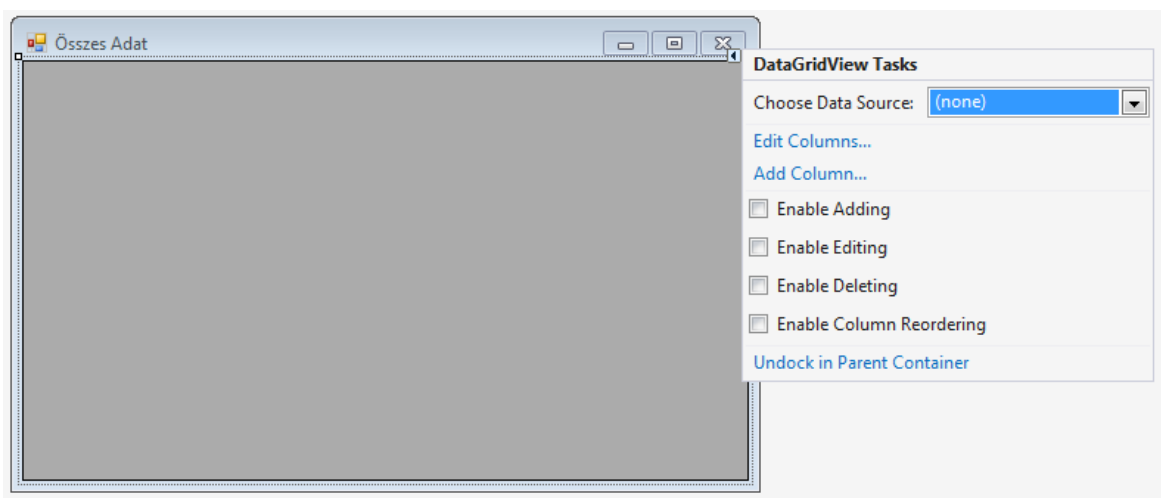
Készítsünk egy eseménykezelőt az frmFelvitel OK gombjához (Click esemény), ami csak akkor engedi bezárni a párbeszédablakot, ha minden mező ki van töltve.

```
private void btOk_Click(object sender, EventArgs e)
{
    bool vanHiba = false;
    if(tbNev.Text.Length == 0)
    {
        MessageBox.Show("A név megadása kötelező!", "Adatbeviteli hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
        vanHiba = true;
    }
    if(tbNeptun.Text.Length == 0)
    {
        MessageBox.Show("Az Neptun kód megadása kötelező!", "Adatbeviteli hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
        vanHiba = true;
    }
    if(tbEmail.Text.Length == 0)
    {
        MessageBox.Show("Az e-mail cím megadása kötelező!", "Adatbeviteli hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
        vanHiba = true;
    }
    if (!vanHiba)
    {
        DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

Készítsünk egy eseménykezelőt az frmFoablak Adatrögzítés menüpontjához (Click esemény), ami megjeleníti a párbeszédablakot, és annak sikeres lezárása esetén kapcsolatot nyit az adatbázishoz, beszúrja az adatbázisba a friss adatokat, majd lezárja a kapcsolatot. Kezeli az esetleges kivételeket.

```
private void tsmiAdatrogzites_Click(object sender, EventArgs e)
{
    //Adatbeviteli párbeszédablak létrehozása.
    frmFelvitel ff = new frmFelvitel();
    //Párbeszédablak megjelenítése és OK gomb esetén adatfelvitel.
    if(ff.ShowDialog() == DialogResult.OK)
    {
        try
        {
            //ODBC kapcsolat objektum létrehozása
            OdbcConnection kapcsolat = new OdbcConnection();
            //Kapcsolódási sztring definiálása
            kapcsolat.ConnectionString = "DSN=SWT1;" + "UID=ADMIN";
            //Adatfelviteli SQL parancs definiálása
            OdbcCommand parancs = new OdbcCommand(
                "INSERT INTO lista (Neptun,Nev,Email)" + "VALUES ('" + ff.Neptun + "','" +
                ff.Nev + "','" + ff.Email+"')", kapcsolat);
            //Kapcsolat megnyitása.
            parancs.Connection.Open();
            //SQL parancs végrehajtása
            parancs.ExecuteNonQuery();
            //Kapcsolat zárása
            kapcsolat.Close();
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message, "Adatbázis hiba", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}
```

Hozzunk létre egy új formot (Project menü, Add Windows Form..., Templates: Windows Form, Name=frmOsszesAdat.cs) frmOsszesAdat néven, text="Összes Adat". Helyezzünk el rajta egy DataGridView komponenst (Name=dgvRacs, Dock=Fill). A hozzáadás, szerkesztés és törlés legyen letiltva.



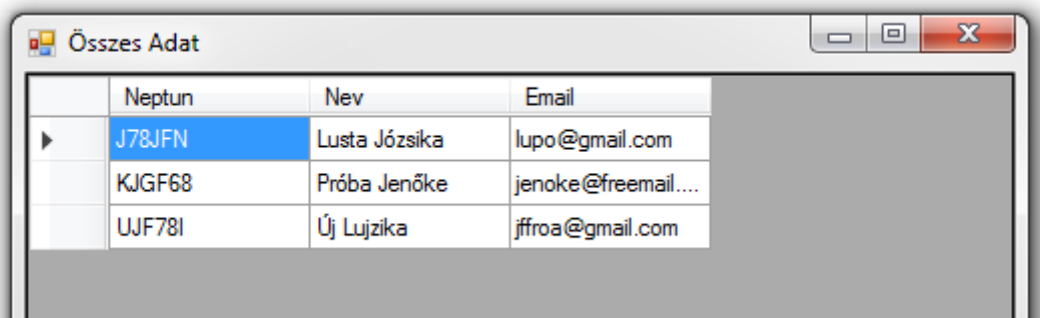
215. ábra.

Hozzunk létre az frmOsszesAdat form osztályában egy csak írható tulajdonságot az adatkötés támogatására:

```
public DataTable AdatForras
{
    set { dgvRacs.DataSource = value; }
}
```

Készítsünk egy eseménykezelőt az frmFoablak Lekérdezés menü Összes adat menüpontjához (Click esemény), melyben létrehozunk egy frmOsszesAdat típusú párbeszédablakot, egy adatbázis kapcsolat objektumot és definiálunk egy adatbázis kapcsolati sztringet. Megnyitjuk az adatbázis kapcsolatot. Létrehozunk egy OdbcDataAdapter objektumot, beállítjuk a lekérdezési parancs tulajdonságát. Létrehozunk egy DataSet objektumot, az adapterrel feltöltjük az adathalmaz objektumot. Beállítjuk a párbeszédablakon levő rács adatforrásaként az adathalmazt. Zárjuk az adatbázis kapcsolatot. Megjelenítjük a párbeszédablakot.

```
private void tsmiOsszesAdat_Click(object sender, EventArgs e)
{
    frmOsszesAdat foa = new frmOsszesAdat();
    OdbcConnection kapcsolat = new OdbcConnection();
    kapcsolat.ConnectionString = "DSN=SWT1;UID=ADMIN";
    try
    {
        kapcsolat.Open();
        OdbcDataAdapter adapter = new OdbcDataAdapter();
        adapter.SelectCommand = new OdbcCommand("SELECT * FROM lista", kapcsolat);
        DataSet dataSet = new DataSet();
        adapter.Fill(dataSet);
        foa.AdatForras = dataSet.Tables["Table"];
        kapcsolat.Close();
        foa.ShowDialog();
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Adatbázis hiba", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```



216. ábra.

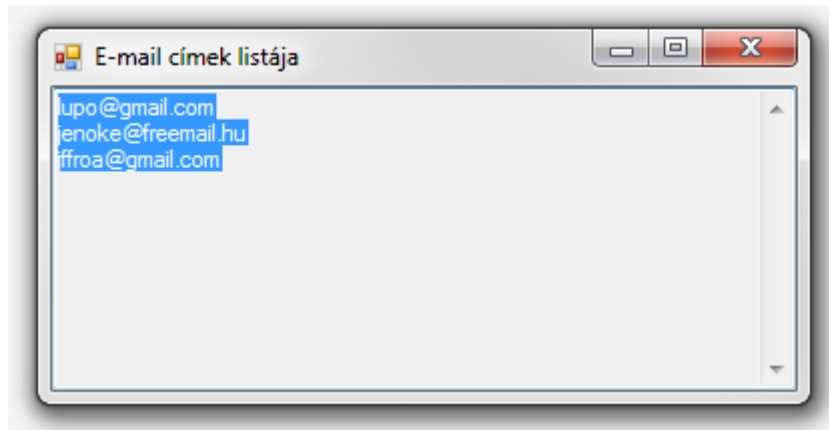
Fordítsuk és futtassuk le a programot, próbáljuk ki az adatfelvitelt és az összes adat lekérdezését.

Hozzunk létre egy formot az e-mail címek megjelenítésére (Name=frmEmailLista, Text=E-mail címek listája). Helyezzünk el egy szerkesztőmezőt a formon (Text Box Name=tbEmail, Text="", Dock=Fill, ReadOnly=True, ScrollBars=Vertical, Multiline=True).

Hozzunk létre egy tulajdonságot az frmEmailLista osztályban a tbEmail szerkesztőmező elérésére:

```
public string EmailCimek
{
    get { return tbEmail.Text; }
    set { tbEmail.Text = value; }
}
```

Készítsünk egy eseménykezelőt az frmFoablak e-mail címek listája menüpontjához (Click esemény). Ebben létrehozunk egy kapcsolat objektumot. Definiálunk egy kapcsolat sztringet. Létrehozunk egy SQL parancs objektumot. Megnyitjuk a kapcsolatot. Adatlekérő objektumot hozunk létre. Létrehozzuk a párbeszédablakot. Töröljük a szövegmező tartalmát. Egyenként lekérjük a rekordokat, és mindegyikből az e-mail címet bemásoljuk a párbeszédablak szövegmező egy új sorába. Lezárjuk az adatlekérő objektumot. Lezárjuk az adatkapcsolatot az adatbázissal.



217. ábra.

```
private void tsmiEmailCimekListaja_Click(object sender, EventArgs e)
{
    try
    {
        // Kapcsolat objektum létrehozása
        OdbcConnection kapcsolat = new OdbcConnection();
        // Kapcsolat sztring definiálása
        kapcsolat.ConnectionString = "DSN=SWT1;UID=admin";
        // SQL parancs objektum létrehozása
        OdbcCommand parancs = new OdbcCommand("SELECT email FROM lista WHERE email IS NOT NULL", kapcsolat);
        // Kapcsolat megnyitása
        kapcsolat.Open();
        // Adatlekérő objektum létrehozása
        OdbcDataReader olvaso = parancs.ExecuteReader();
        // Létrehozzuk a párbeszédablakot
        frmEmailLista fem = new frmEmailLista();
        // Töröljük a szövegmező tartalmát
        fem.EmailCimek = "";
        // Egyenként lekérjük a rekordokat, és mindegyikből az e-mail címet
        // bemásoljuk a párbeszédablak szövegmező egy új sorába
    }
}
```

```
while (olvaso.Read())
{
    string email = olvaso.GetString(0);
    fem.EmailCimek += email + "\r\n";
}
// Lezárjuk az adatlekérő objektumot
olvaso.Close();
// Lezárjuk az adatkapcsolatot az adatbázissal
kapcsolat.Close();
fem.ShowDialog();
}
catch (Exception exc)
{
    MessageBox.Show(exc.Message, "Adatbázis hiba", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}
```

### 6.2.3. Házi Feladat

Egészítse ki a programot az adatbázis módosításának lehetőségével. Egy űrlapon kérje be az Neptun kódot, keresse ki az annak megfelelő rekordot, jelenítse meg, és tegye lehetővé a felhasználó számára, hogy módosítsa azt, majd a módosított rekordot másolja az adatbázisba.