

# Objektumorientált programozás C# nyelven

## 1. rész

Osztályok és objektumok  
Mezők és metódusok  
Konstruktor és destruktor  
Névterek és hatókörök  
Láthatósági szintek  
Osztály szintű tagok  
Beágyazott osztályok  
Felbontott típusok

**Készítette:**

**Miklós Árpád**

**Dr. Kotsis Domokos**

# Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

# Osztályok és objektumok

- **Osztály**: belső adatok és a rajtuk műveleteket végző algoritmusok által alkotott egységes struktúra
- **Objektum**: valamely osztály egy tényleges példánya
  - Az objektumok (bizonyos esetekben maguk az osztályok is) a program futása során egymással kommunikálnak
- **Osztály tartalma (az osztály „tagjai”)**:
  - Mezők („field”)
    - Normál és csak olvasható változók, konstansok („constant”)
  - Metódusok („method”)
    - Normál metódusok, konstruktorok („constructor”), destruktorkok („destructor”)
  - Tulajdonságok\* („property”) és indexelők\* („indexer”)
  - Események\* („event”)
  - Operátorok\* („operator”)
  - Beágyazott típusok („nested type”)
    - Osztályok („class”), struktúrák\* („struct”), interfészek\* („interface”), képviselők\* („delegate”)

# Osztályok és objektumok

- **Az osztályok deklarálása a class kulcsszó segítségével történik**

- Az osztályok deklarációja egyben tartalmazza az összes tag leírását és a metódusok megvalósítását
  - Az osztályoknál tehát nincs külön deklaráció (létrehozás) és definíció (kifejtés)

```
class Példaosztály
{
    // Itt kell deklarálnunk az osztály összes tagját (mezőket, metódusokat...)
    // A metódusok konkrét megvalósítását szintén itt kell megadnunk
}
```

- **Osztályok és objektumok tagjainak elérése: „ . ” operátor**

- Példány szintű tagoknál a példány nevét, osztály szintű tagoknál (ezeket lásd később) az osztály nevét kell az operátor elé írunk
  - Az osztály saját tagjainak elérésekor (tehát az osztály saját metódusainak belsejében) nem kötelező kiírni a példány, illetve az osztály nevét

# Mezők

- **Minden változó tagja egy osztálynak (tagváltozó)**
  - Ezeket az adatelemeket nevezzük mezőknek
- **A mezők értéke helyben is megadható (inicializálás)**

```
string jegy = "jeles";  
int j = -10;
```

- **A mezők lehetnek**
  - Olvasható/írható mezők
    - Értékük tetszés szerint olvasható és módosítható
  - Csak olvasható mezők
    - Értékük kizárólag inicializálással vagy konstruktorból állítható be

```
readonly string SosemVáltozomMeg = "I Will Stay The Same";
```

- Konstans mezők
  - Értéküket a fordítóprogram előre letárolja, futási időben sosem módosíthatók

```
const double  $\pi$  = 3.14159265;  
const int összeg = 23 * (45 + 67);
```

# Metódusok

- Minden metódus tagja egy osztálynak (tagfüggvény)
- A metódusok rendelkezhetnek
  - Megadott vagy változó darabszámú paraméterrel (params kulcsszó)

**void** EgyparaméteresMetódus(**bool** feltétel)

**void** TöbbparaméteresMetódus(**int** a, **float** b, **string** c)

**void** MindenbőlSokatElfogadóMetódus(**params object[]** paraméterTömb)

- Visszatérési értékkel
  - Nem kötelező, ha nincs, ezt a void kulcsszóval kell jelölni

**void** NincsVisszatérésiÉrtékem()

**int** EgészSzámotAdokVissza(**float** paraméter)

**string** VálaszomEgyKaraktersorozat(**string** bemenőAdat)

**SajátTípus** Átalakító(**SajátTípus** forrásObjektum, **int** egyikMezőÚjÉrtéke, **string** másikMezőÚjÉrtéke)

- A paraméterek és a visszatérési érték határozzák meg azt a protokolt, amelyet a metódus használatához be kell tartani
  - Szokás ezt a metódus „aláírásának” vagy „szignatúrájának” is nevezni

# Függvények

**Séma:**

**típus függvénynév(paraméterek)  
{függvénytörzs}**

# Függvények típusa

1. A típus a visszaadott érték típusa, vagy **void**.
2. A típust megelőzheti
  - a.) a láthatóság megadása (**private**, **protected**, **public**, stb.),
  - b.) osztály szintű tagnál a **static** szó.



# Függvények paraméterei

**A paramétereknél megadandó a típus és az a név, amelyen a függvénytörzsben a paraméterre hivatkozunk.**

**Referencia típusú paramétereknél a paraméter mind bemenő, mind visszatérő értéket tartalmazhat.**

**Érték típusú paraméterek alapesetben csak „bemenő” paraméterek lehetnek, azaz csak értéket adhatnak a függvény számára.**

**Érték típusú paramétert „kimenő”-vé az **out**, „ki- és bemenő”-vé a **ref** módosító tehet.**

# Függvénytörzs

**A végrehajtandó utasítások, melyek használhatják a bemenő paramétereket.**

**A függvény visszatérő értéke a **return** alapszót követi (ebből több is lehet a program különböző ágain).**

**Visszatérési érték nélküli (**void**) függvélynél – ha a program mindig a függvénytörzs fizikai végénél fejeződik be – a **return** utasítás nem kötelező.**

# Példa ref-re

```
class Ref
```

```
{
```

```
    public void kiír(ref int a)
```

```
    {
```

```
        a=a+5;
```

```
    }
```

```
}
```

```
class Reftpld
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        int x=3;
```

```
        Ref próba=new Ref();
```

```
        próba.kiír(ref x);
```

```
        System.Console.WriteLine(x);
```

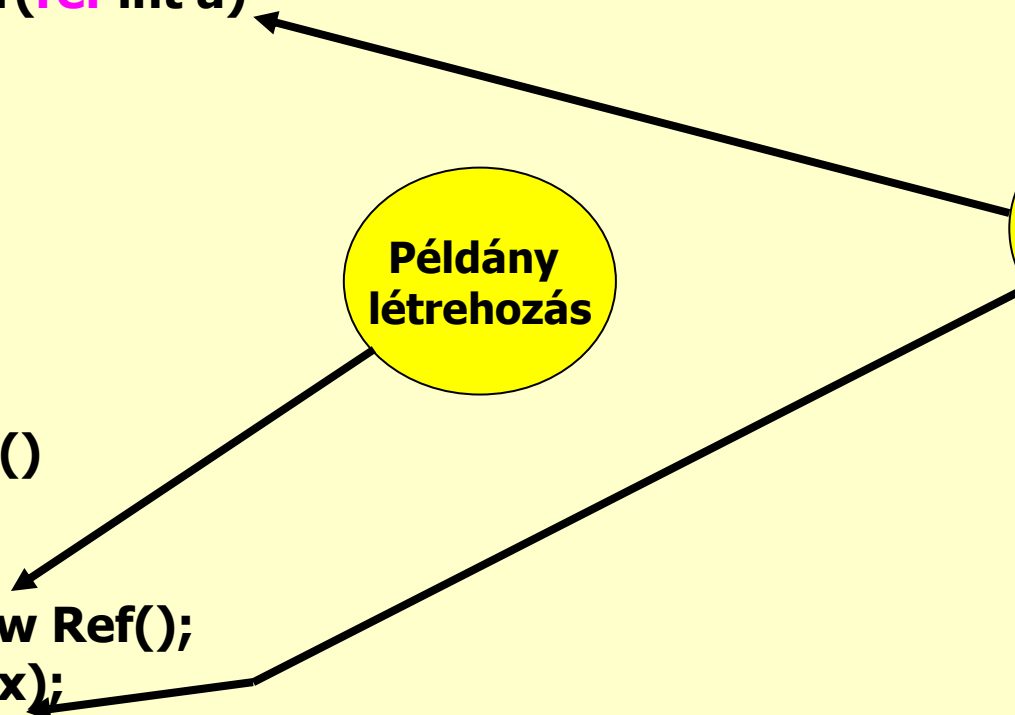
```
        System.Console.ReadLine();
```

```
    }
```

```
}
```

Példány  
létrehozás

Változtatható  
paraméter



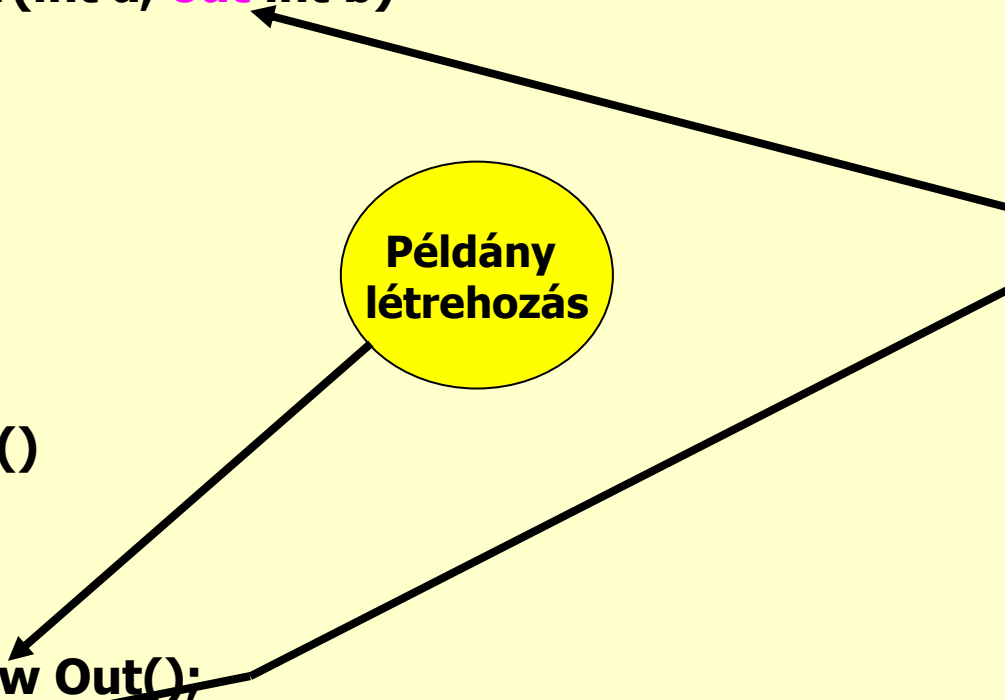
# Példa out-ra

```
class Out
{
    public void kiír(int a, out int b)
    {
        b=a;
    }
}

class Outpld
{
    static void Main()
    {
        int x=3;
        int y=4;
        Out próba=new Out();
        próba.kiír(x,out y);
        System.Console.WriteLine(x+" "+y);
        System.Console.ReadLine();
    }
}
```

Példány  
létrehozás

Kimenő  
paraméter



# Példa a Main() fv. paraméterére

```
class Bepar
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        int i;
```

```
        for (i=0;i<args.Length-1; i=i+1)
```

```
            System.Console.Write(args[i]+" ");
```

```
            if (args.Length>0)
```

```
                System.Console.WriteLine(args[i]);
```

```
                System.Console.ReadLine();
```

```
    }
```

```
}
```

Paraméter

Stringek  
száma

Paraméter  
stringek

# Speciális metódusok – a konstruktor

- **Minden osztálynak rendelkeznie kell konstruktorral**
  - A konstruktor gondoskodik az osztály példányainak létrehozásáról
    - Szokás „példánykonstruktornak” is nevezni
  - A konstruktorok neve mindig megegyezik az osztály nevével
  - Több konstruktort is létrehozhatunk más-más paraméterlistával
    - Egy konstruktor a `this` kulcsszó segítségével meghívhat egy másik konstruktort is
  - Ha mi magunk nem deklarálunk konstruktort, a C# fordító automatikusan létrehoz egy paraméter nélküli alapértelmezett konstruktort
- **Új objektum a `new` operátor segítségével hozható létre**
  - A `new` operátor gondoskodik a megfelelő konstruktor hívásáról
    - Az osztályok konstruktorait kívülről nem kell és nem is lehet más módon meghívni

`System.Object` IgaziŐskövület = `new System.Object()`;

Paraméter nélküli konstruktor hívása

`SajátTípus` példány = `new SajátTípus(25)`;

Egy „int” típusú paraméterrel rendelkező konstruktor hívása

# Speciális metódusok – a destruktor

- **Az osztályoknak nem kötelező destruktorral rendelkezniük**
  - A destruktor neve egy „ ~ ” karakterből és az osztály nevéből áll
- **Az objektumok megszüntetése automatikus**
  - Akkor szűnik meg egy objektum, amikor már biztosan nincs rá szükség
    - Az objektumok megszüntetésének időpontja nem determinisztikus (nem kiszámítható)
  - A futtatókörnyezet gondoskodik a megfelelő destruktor hívásáról
    - Nem kell (és nem is lehet) közvetlenül meghívni az osztályok destruktorait
    - A destruktor nem tudhatja, pontosan mikor hívódik meg

```
class SajátTípus
{
    // Destruktor
    ~SajátTípus()
    {
    }
}
```

# 1. példa

```
class Részvény
{
    private readonly string részvélynév;
    private double részvényárfolyam = 0.0;
    public int Darabszám;

    public Részvény(string név, double árfolyam, int darabszám)
    { // Konstruktor (neve megegyezik az osztály nevével) - beállítja az adatmezők kezdeti értékét
    }
    public void Vétel(int mennyiség)
    { // A paraméterben megadott mennyiségű részvény vásárlása
    }
    public void Eladás(int mennyiség)
    { // A paraméterben megadott mennyiségű részvény eladása
    }
    public void ÁrfolyamBeállítás(double árfolyam)
    { // Az aktuális árfolyam beállítása a paraméterben megadott árfolyam alapján
    }
    public double Érték()
    { // Részvény összértékének kiszámítása
    }
}
```



# 1. példa (folytatás)

```
class Részvénykezelő
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        Részvény IBM = new Részvény("IBM", 77.59, 100);
```

```
        Részvény nVidia = new Részvény("NVDA", 21.49, 100);
```

```
        IBM.Vétel(50);
```

```
        nVidia.Vétel(25);
```

```
        nVidia.ÁrfolyamBeállítás(29.15);
```

```
        nVidia.Eladás(50);
```

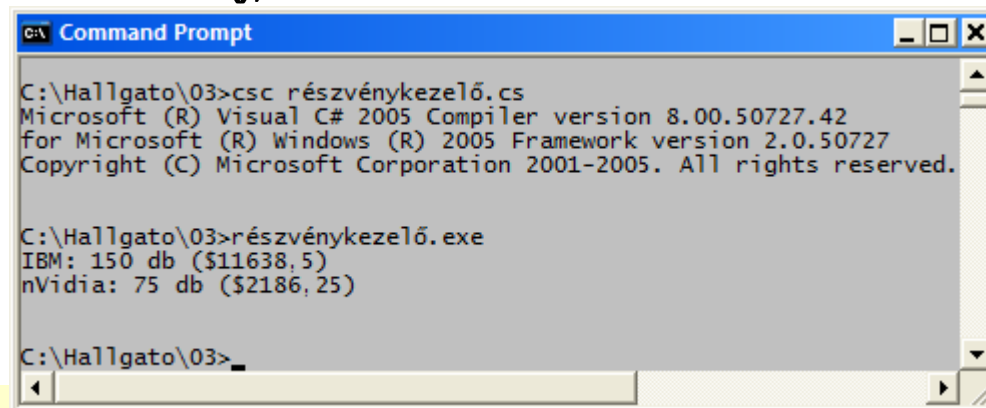
```
        System.Console.WriteLine("IBM: " + IBM.Darabszám + " db ($" + IBM.Érték() + ")");
```

```
        System.Console.WriteLine("nVidia: " + nVidia.Darabszám + " db ($" + nVidia.Érték() + ")");
```

```
        System.Console.ReadLine();
```

```
    }
```

```
}
```



```
C:\ Command Prompt

C:\Hallgato\03>csc részvénykezelő.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\Hallgato\03>részvénykezelő.exe
IBM: 150 db ($11638,5)
nVidia: 75 db ($2186,25)

C:\Hallgato\03>
```

részvénykezelő.cs

# Névterek

- **A névterek az elnevezések tetszőleges logikai csoportosítását teszik lehetővé**
  - Nincs köztük a fizikai tároláshoz (fájlokhoz és mappákhoz)
    - Egy fájlban több névtér, egy névtér több fájlban is elhelyezhető
  - Tetszőlegesen egymásba ágyazhatók
    - A beágyazott névterek tagjait a „ . ” karakterrel választhatjuk el
  - A névtérbe be nem sorolt elemek egy ún. globális névtérbe kerülnek

```
namespace A
```

```
{  
    namespace B  
    {  
        class Egyik {...}  
    }  
}
```

```
...
```

```
A.B.Egyik példa = new A.B.Egyik();
```

x.cs

```
namespace A.B
```

```
{  
    class Másik {...}  
}  
namespace C  
{  
    class Harmadik {...}  
}
```

```
...
```

```
A.B.Másik példa2 = new A.B.Másik();  
C.Harmadik példa3 = new C.Harmadik();
```

y.cs

Ez a két névtér azonos  
(A.B)

# Névterek (folytatás)

- **Minden névre a saját névterével együtt kell hivatkozni**
  - A teljes (minősített) név formája: névtér.elnevezés
  - A névterek importálhatók (hivatkozás céljára előkészíthetők) a `using` kulcsszó segítségével
    - Ezt követően az adott névtérben található elnevezések elé hivatkozáskor nem kell kiírni a névteret, feltéve, hogy az elnevezés így is egyértelműen azonosítható

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

- A névtereknek importálás helyett álnév is adható
  - Célja a hosszú, de importálni nem kívánt névterek egyértelmű rövidítése

```
using System;  
using SOAP = System.Runtime.Serialization.Formatters.Soap;  
...  
SOAP.SoapFormatter formázó = new SOAP.SoapFormatter();  
Console.WriteLine(formázó);
```

Importált névtér

Nem importált névtér álnévvel

- A névterek Microsoft által javasolt formátuma:  
Cégnév.Technológia.Funkció[.Design]
  - Példa: Microsoft.VisualBasic.CompilerServices

# Hatókörök

- **Kijelöli a változók érvényességi tartományát**
  - Nem azonos a névtérrel (amely a hivatkozás módját szabályozza)
- **A C# hatókörre vonatkozó szabályai:**
  - Osztályok tagváltozói csak ott érhetők el, ahol az osztály is elérhető
  - Helyi változók a deklarációjukat tartalmazó blokk vagy metódus lezárásáig („ }”) érhetők el
  - A for, foreach, while, do...while utasításokban deklarált helyi változók csak az adott utasítás belsejében érhetők el
  - Ha egy változó érvényes, de nem azonosítható egyértelműen, akkor a C# a hivatkozást a legbelső érvényességi tartományra vonatkoztatja
    - Azonos érvényességi tartományon belül azonos néven nem hozhatók létre változók
    - A tagváltozók érvényességi tartományában létrehozhatók azonos nevű helyi változók
    - Ebben az esetben a legbelső változó „elrejt” a vele azonos nevű, hozzá képest külső szinten elhelyezkedő változókat
    - Ha a legbelső érvényességi tartományban egy azonos nevű külső tagváltozót kívánunk elérni, akkor példány szintű változók esetén a this kulcsszót, osztály szintű változók esetén az osztály nevét kell a változó elé írunk „ . ” karakterrel elválasztva

# Láthatósági szintek

- Láthatósági (hozzáférési) szintek a C# nyelvben

Szint	Hozzáférési lehetőség az adott taghoz	Megjegyzés
<b>public</b>	Korlátlan	
<b>protected</b>	Adott osztály és leszármazottai*	
<b>internal</b>	Adott program, adott osztály	.NET
<b>protected internal</b>	Adott program, adott osztály és leszármazottai*	.NET <sup>1</sup>
<b>private</b>	Adott osztály	

<sup>1</sup> A .NET biztosít még egy további láthatósági szintet („protected and internal”), amelyet a C# nyelv nem támogat

- A névterek láthatósága mindig **public**
- A típusok (osztályok) láthatósága **public** vagy **internal** (alapértelmezés)
- Az osztályok tagjainak láthatósága tetszőlegesen megválasztható
  - A tagok láthatósága alapértelmezésben mindig **private**
  - A beágyazott típusok (osztályok) láthatóság szempontjából normál tagoknak minősülnek (láthatóságuk tetszőlegesen megadható, alapértelmezésben **private**)
  - A felsorolások elemeinek és az interfészek\* tagjainak láthatósága mindig **public**

# Metódusok átdefiniálása

- Egy osztályon belül is létrehozhatunk több azonos nevű, de eltérő paraméterlistával és visszatérési értékkel rendelkező metódust
  - Ezzel a technikával ugyanazt a funkciót többféle paraméterekkel és visszatérési értékkel is meg tudjuk valósítani ugyanazon a néven
  - Logikusabb, átláthatóbb programozási stílust tesz lehetővé

```
class Részvény
{
    ...
    public void Vétel(int mennyiség)
    { // A paraméterben megadott mennyiségű részvény vásárlása
        darabszám += mennyiség;
    }
    public void Vétel(int mennyiség, double árfolyam)
    { // A paraméterben megadott mennyiségű részvény vásárlása a megadott árfolyam beállításával
        darabszám += mennyiség;
        this.árfolyam = árfolyam;
    }
}
```

# Osztály szintű tagok

- **Az osztály szintű mezők az osztály saját adatmezői**
  - Minden osztály csak egyet tárol ezekből a mezőkből, függetlenül a később létrehozott példányok számától
    - Ezeket a mezőket tehát nem a példányok, hanem maga az osztály birtokolja
- **Az osztály szintű metódusok magán az osztályon működnek**
  - Akkor is hívhatók, ha még egyetlen példány sem létezik az osztályból
    - Csak osztály szintű mezőket használhatnak
    - Osztály szintű metódusoknál nem létezik aktuális objektum, így this paraméter sem
    - Konkrét példányt nem igénylő feladatra is alkalmasak (pl. főprogram megvalósítása)

// ...és most már minden kulcsszót könnyen felismerhetünk ebben a "bonyolult" programban ☺

```
class ElsőProgram
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, C# World");
    }
}
```

# Osztály szintű tagok (példa, 1. rész)

```
using System;
```

```
class Példányszámláló
```

```
{
```

```
    public static int Darabszám;
```

Osztály szintű adatmező

```
    static Példányszámláló()
```

Osztály szintű konstruktor  
(egyik fő célja az osztály szintű mezők  
kezdeti értékének beállítása)

```
{
```

```
        Darabszám = 0;
```

```
}
```

```
    public Példányszámláló()
```

Konstruktor

```
{
```

```
        Darabszám++;
```

```
}
```

```
    ~Példányszámláló()
```

Destruktor

```
{
```

```
        Darabszám--;
```

```
        Console.WriteLine("Megszűnt egy példány. A fennmaradók száma: " + Darabszám);
```

```
}
```

```
}
```

példányszámláló.cs



# Osztály szintű tagok (példa, 2. rész)

```
class PéldányszámlálóTeszt
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        Példányszámláló teszt = new Példányszámláló();
```

```
        Console.WriteLine("Létrehoztam egy példányt");
```

```
        Console.WriteLine("Példányszám: " + Példányszámláló.Darabszám);
```

```
        for (int i = 0; i < 10; i++)
```

```
            new Példányszámláló();
```

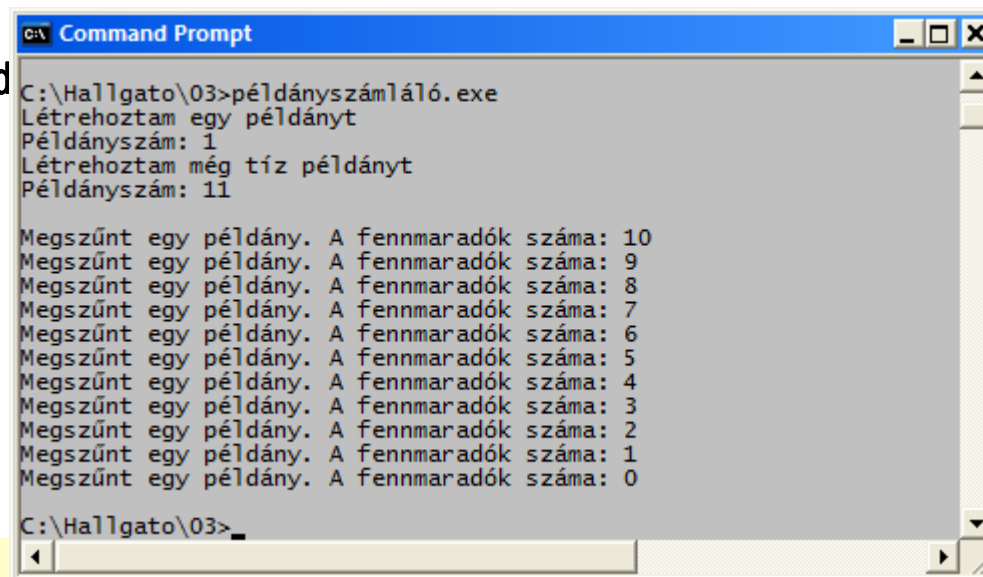
```
        Console.WriteLine("Létrehoztam még tíz példányt");
```

```
        Console.WriteLine("Példányszám: " + Példányszámláló.Darabszám);
```

```
        Console.Read
```

```
    }
```

```
}
```



```
Command Prompt
C:\Hallgato\03>példányszámláló.exe
Létrehoztam egy példányt
Példányszám: 1
Létrehoztam még tíz példányt
Példányszám: 11

Megszűnt egy példány. A fennmaradók száma: 10
Megszűnt egy példány. A fennmaradók száma: 9
Megszűnt egy példány. A fennmaradók száma: 8
Megszűnt egy példány. A fennmaradók száma: 7
Megszűnt egy példány. A fennmaradók száma: 6
Megszűnt egy példány. A fennmaradók száma: 5
Megszűnt egy példány. A fennmaradók száma: 4
Megszűnt egy példány. A fennmaradók száma: 3
Megszűnt egy példány. A fennmaradók száma: 2
Megszűnt egy példány. A fennmaradók száma: 1
Megszűnt egy példány. A fennmaradók száma: 0

C:\Hallgato\03>
```

példányszámláló.cs

# **Feladat 1.**

**Készítsen programot a háromelemű valós vektorok kezelésére az alábbi menüpon-  
tokkal.**

- **Beolvasás**
- **Kiíratás**

**Használjon statikus (osztály szintű)  
metódusokat!**

# Megoldás stat. metódusokkal I.

```
class vektordef
```

```
{
```

```
public static void beolvas(float [] partomb)
```

```
{
```

```
for(int i=0;i<3; i++)
```

```
{System.Console.WriteLine(„Az”+(i+1)+”-ik elem: ”);
```

```
partomb[i]=float.Parse(System.Console.ReadLine());}
```

```
}
```

```
public static void kiír(float [] partomb)
```

```
{
```

```
for(int i=0;i<3; i++)
```

```
{System.Console.WriteLine(partomb[i]);}
```

```
return;
```

```
}
```

```
}
```

A sorrend  
teszőleges

Kötelezően  
a metódus  
neve előtt

Nincs visszatérő érték,  
egy kimenet van, így a  
„return” nem kötelező

# Megoldás stat. metódusokkal II.

```
class vektorkez  
{  
    static void Main()  
    {  
        float [] vektor=new float[3];  
        vektordef.beolvas(vektor);  
        vektordef.kiír(vektor);  
    }  
}
```

# Feladat 1.b.

**Készítse el az 1. Feladatban leírtakat  
objektum példány használatával!**

# Megoldás példánnyal I.

```
class vektordef
{
    public void beolvas(float [] partomb)
    {
        for(int i=0;i<3; i++)
        {System.Console.WriteLine(„Az”+(i+1)+”-ik elem: ”);
          partomb[i]=float.Parse(System.Console.ReadLine());}
    }

    public void kiir(float [] partomb)
    {
        for(int i=0;i<3; i++)
        {System.Console.WriteLine(partomb[i]);}
        return;
    }
}
```

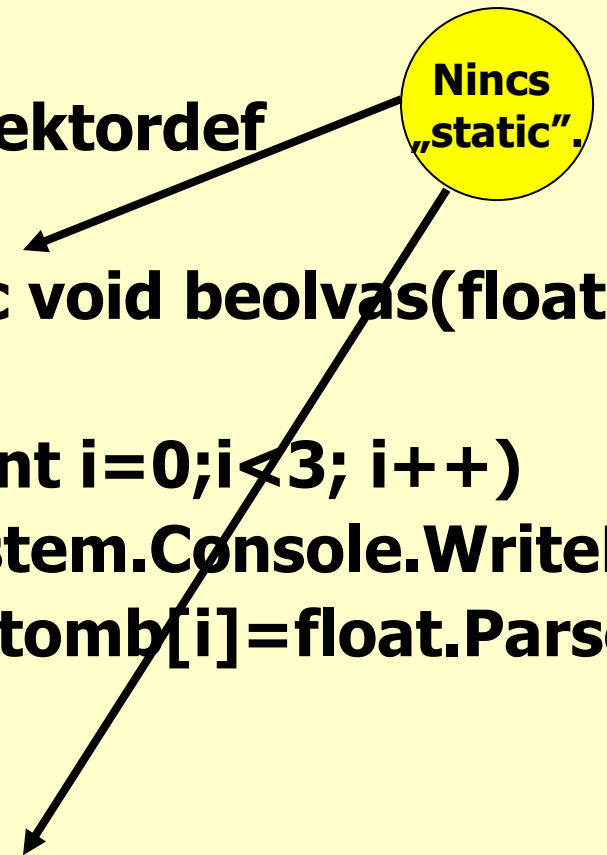
Nincs „static”.

Kötelezően a metódus neve előtt

Nincs visszatérő érték, egy kimenet van, így a „return” nem kötelező

1.2 2006. szeptember 8.

# Megoldás obj. pld.-al I.



```
class vektordef
{
    public void beolvas(float [] partomb)
    {
        for(int i=0;i<3; i++)
        {System.Console.WriteLine("Az" +(i+1) + "-ik elem: ");
          partomb[i]=float.Parse(System.Console.ReadLine());}
    }

    public void kiír(float [] partomb)
    {
        for(int i=0;i<3; i++)
        {System.Console.WriteLine(partomb[i]);}
    }
}
```

Nincs „static”.

# Megoldás példánnyal II.

```
class vektorkez
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        float [] vektor=new float[3];
```

```
        vektordef sajvekt=new vektordef();
```

```
        sajvekt.beolvas(vektor);
```

```
        sajvekt.kiír(vektor);
```

```
    }
```

```
}
```



Oblektum  
példány  
létrehozása.



## **Feladat 2.**

**Egészítse ki a háromelemű valós vektorok kezelésére készített programot az alábbi menüpontokkal.**

- **Skaláris szorzat.**
- **Skalárral való szorzás.**

**Használjon objektum példányt!**

# Skaláris szorzat

```
public float skalarissz(float [] partomb1,float [] partomb2)  
{  
    float s=0;  
    for(int i=0;i<3; i++)  
        {s=s+partomb1[i]*partomb2[i];}  
    return s;  
}
```

# Szorzás skalárral a.

A visszaadott érték



```
public void skalarralesz(float [] partomb, float k)
{
    for(int i=0;i<3; i++)
    {
        partomb[i]=partomb[i]*k;
    }
}
```

# Szorzás skalárral b.

```
public float [] skalarralsz(float [] partomb, float k)
{
    float [] p=new float[3];
    for(int i=0;i<3; i++)
    {
        p[i]=partomb[i]*k;
    }
    return p;
}
```



A visszaadott érték

# Megoldás obj. pld-al főprg. a.

```
class vektorkez
{
    static void Main()
    {
        float [] vektor1=new float[3];
        float [] vektor2=new float[3];
        float [] vektor3=new float[3];
        float v;
        vektordef sajvekt=new vektordef();
        sajvekt.beolvas(vektor1);
        sajvekt.beolvas(vektor2);
        sajvekt.skalarralesz(vektor1,3);
        vektor3=vektor1;
        v=sajvekt.skalarissz(vektor3,vektor2);
        sajvekt.kiir(vektor3);
        System.Console.WriteLine(v);
        System.Console.ReadLine();
    }
}
```

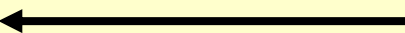
Oblektum  
példány  
létrehozása.

A visszaadott érték

# Megoldás obj. pld-al főprg. b.

```
class vektorkez
{
    static void Main()
    {
        float [] vektor1=new float[3];
        float [] vektor2=new float[3];
        float [] vektor3=new float[3];
        float v;
        vektordef sajvekt=new vektordef();
        sajvekt.beolvas(vektor1);
        sajvekt.beolvas(vektor2);
        vektor3=sajvekt.skalarrralsz(vektor1,3);
        v=sajvekt.skalarissz(vektor3,vektor2);
        sajvekt.kiír(vektor3);
        System.Console.WriteLine(v);
        System.Console.ReadLine();
    }
}
```

Oblektum  
példány  
létrehozása.



A visszaadott érték



## **Feladat 3.**

**Adott, hogy egy vállalat három gépkocsija hány km-t futott az első illetve a második félévben.**

**Ismert továbbá az első féléves üzemeltetési költség autók szerint km-enként. Ez a költség a második félévre 5%-kal emelkedett.**

**Határozza meg a gépkocsik éves összköltségét!**

```
class Auto  
{ static void Main()  
    { float [] km1=new float[3];  
        float [] km2=new float[3];  
        float [] kolt1=new float[3];  
        float [] kolt2=new float[3];  
        float v1,v2,koltseg;  
        vektordef sajvekt=new vektordef();  
        System.Console.WriteLine("Első félévi km.:");  
        sajvekt.beolvas(km1);  
        System.Console.WriteLine("Második félévi km.:");  
        sajvekt.beolvas(km2);  
        System.Console.WriteLine("Első félévi költség:");  
        sajvekt.beolvas(kolt1);  
        for (int i = 0; i < 3; i++)  
            kolt2[i]=kolt1[i];  
        sajvekt.skalarralesz(kolt2,(float)1.05);  
        v1=sajvekt.skalarissz(km1,kolt1);  
        v2=sajvekt.skalarissz(km2,kolt2);  
        koltseg=v1+v2;  
        System.Console.Write("A teljes költség összege: ");  
        System.Console.WriteLine(koltseg);  
        System.Console.ReadLine();  
    } }
```



## **Feladat 4.**

**Egészítse ki a háromelemű valós vektorok kezelésére készített programot az alábbi menüpontokkal:**

- Oszlopvektor és sorvektor szorzata (lehet különböző hosszúságú).**
- Kiírás két dimenziós tömbre ugyanazon a néven, mint a vektorok esetén (nem ismert az oszlop és sorhossz)!**

**Használjon objektum példányt!**

# Az oszlop és sorvektor szorzata

```
public float[,] osvsz(float[] partomb1, float[] partomb2)
{
    float[,] p = new float[partomb1.Length, partomb2.Length];
    for (int i = 0; i < partomb1.Length; i++)
        for (int j = 0; j < partomb2.Length; j++)
            p[i, j] = partomb1[i] * partomb2[j];
    return p;
}
```

# Kétdimenziós tömb kiírása

```
public void kiír(float[,] partomb)
{
    for (int i = 0; i < partomb.GetLength(0); i++)
    {
        for (int j = 0; j < partomb.GetLength(1); j++)
        {
            System.Console.Write(partomb[i, j]+"  ");
        }
        System.Console.WriteLine();
    }
    return;
}
```

## **Feladat 5.**

**Adott, egy vállalat három gépkocsijának üzemeltetési költsége autók szerint km-enként.**

**Ismert a leggyakoribb 4 úticél távolsága. Határozza meg a gépkocsik költségét ezekhez az úticélokhoz gépkocsinként és úticélonként!**

# A módosított vektor beolvasó

```
public void beolvas(float[] partomb)
{
    for (int i = 0; i < partomb.Length; i++)
    {
        System.Console.WriteLine("add meg az" + (i + 1) + "-ik
elemet! ");
        partomb[i] = float.Parse(System.Console.ReadLine());
    }
    return;
}
```

# Megoldás

```
float[] céltáv = new float[4];
```

```
...
```

```
System.Console.WriteLine("A főbb úticélok költsége autónként: ");  
sajvekt.kiír(sajvekt.osvsz(kolt1, céltáv));
```

```
...
```

# A this paraméter

- **A példány szintű metódusokban szükség lehet rá, hogy hivatkozni tudjunk arra az objektumra, amelyik a metódust éppen végrehajtja**
- **E hivatkozás a rejtett this paraméter segítségével valósul meg**
  - A rejtett this paraméter minden példány szintű metódusban az aktuális objektumot jelöli
    - Osztály szintű tagok esetén ez a paraméter nem létezik (az osztály szintű tagokat lásd később)
  - Nem kell deklarálni, ezt a fordítóprogram automatikusan megteszi
  - Általában a következő esetekben használatos:
    - Az aktuális objektumot paraméterként vagy eredményként szeretnénk átadni
    - Az érvényes hatókörön belül több azonos nevű tag található (pl. egymásba ágyazott hatókörök vagy névterek esetén), így ezek a tagok csak segítséggel azonosíthatók egyértelműen

# A this paraméter (példa)

- Milyen nehézség adódott volna, ha az 1. példában az alábbi mezőneveket használjuk?

```
class Részvény
```

```
{
```

```
    private string név;
```

```
    private double árfolyam;
```

```
    public int darabszám;
```

```
    public Részvény(string név, double árfolyam, int darabszám)
```

```
{
```

```
        részvéynév = név;
```

```
        részvényárfolyam = árfolyam;
```

```
        Darabszám = darabszám;
```

```
}
```

```
    public void ÁrfolyamBeállítás(double árfolyam)
```

```
{
```

```
        részvényárfolyam = árfolyam;
```

```
}
```

```
}
```

**Probléma:** hogyan tudjuk módosítani a „Részvény” osztály „név”, „árfolyam” és „darabszám” nevű mezőit?



# A this paraméter (példa)

- **Megoldás a this paraméter segítségével**

```
class Részvény
{
    private string név;
    private double árfolyam;
    public int darabszám;

    public Részvény(string név, double árfolyam, int darabszám)
    {
        this.név = név;
        this.árfolyam = árfolyam;
        this.darabszám = darabszám;
    }

    public void ÁrfolyamBeállítás(double árfolyam)
    {
        this.árfolyam = árfolyam;
    }
}
```

# Beágyazott osztályok

- **Az osztályok tetszőleges mélységben egymásba ágyazhatók**
  - Az egymásba ágyazással logikai tartalmazást jelezhetünk
    - Az egymásba ágyazás nem jelent hierarchikus alá-, illetve fölérendelést
  - A beágyazott típusok (osztályok) láthatóság szempontjából normál tagoknak minősülnek (láthatóságuk tehát tetszőlegesen megadható, alapértelmezésben private)
- **Beágyazott osztályokra azok teljes (minősített) nevével hivatkozhatunk**
  - A hivatkozás formája: osztály.beágyazottosztály

# Beágyazott osztályok (példa)

```
using System;
```

```
class KülsőOsztály
```

```
{
```

```
    public class BelsőOsztály
```

```
    {
```

```
        public void Üzenő()
```

```
        {
```

```
            Console.WriteLine("Hurrá, belül vagyunk!");
```

```
        }
```

```
    }
```

```
    public void Üzenő()
```

```
    {
```

```
        Console.WriteLine("Kívül vagyunk.");
```

```
    }
```

```
}
```

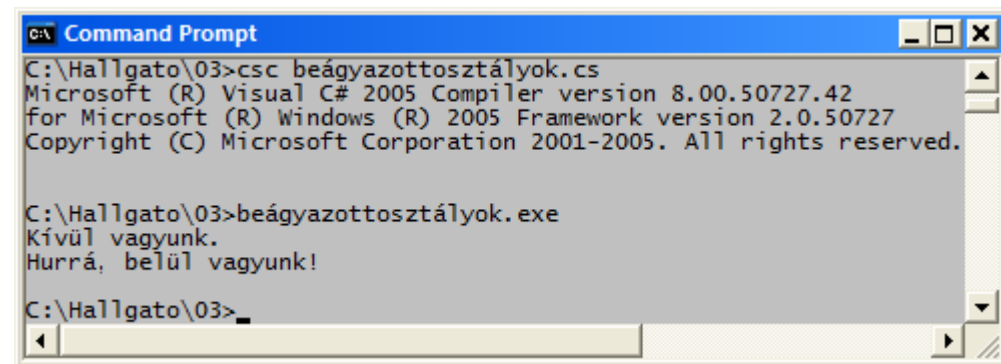
```
...
```

```
KülsőOsztály K = new KülsőOsztály();
```

```
KülsőOsztály.BelsőOsztály B = new KülsőOsztály.BelsőOsztály();
```

```
K.Üzenő();
```

```
B.Üzenő();
```



```
C:\Hallgato\03>csc beagyazottosztalyok.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\Hallgato\03>beagyazottosztalyok.exe
Kívül vagyunk.
Hurrá, belül vagyunk!

C:\Hallgato\03>
```

beagyazottosztalyok.cs

# Felbontott típusok

- **A felbontott típusok több fizikai részre osztott, logikai szempontból viszont egységes típusok**
  - Példa: egy-egy osztály forráskódja elosztva, több fájlban is tárolható
  - A felbontott típusok minden részét a partial kulcsszóval kell megjelölni
  - Előnye, hogy a típusok úgy oszthatók meg több programozó vagy automatikus kódgenerátor között, hogy fizikailag nem kell osztozniuk a forrásfájlokon
    - Különválasztható (és ezáltal külön fejleszthető és verzionálható) az osztályok automatikusan, illetve kézzel előállított része
    - Különválasztható az egyes osztályok kódján dolgozó fejlesztőcsapatok munkája is
- **A felbontott típusok elemeit a C# fordító összefésüli**
  - A fordító úgy kezeli az elemeket, mintha egy fájlban, egy típusdefinícióként hoztuk volna létre őket
    - Ellentmondás esetén a fordítás nem lehetséges
    - A felbontott típusok elemei csak együtt, egyszerre fordíthatók le
      - Nem lehetséges tehát már lefordított osztályokat utólag ilyen technikával bővíteni

# Felbontott típusok (példa)

```
partial class Részvény
```

```
{
```

```
    private readonly string részvényténv;
```

```
    private double részvényárfolyam = 0.0;
```

```
pub
```

```
pub
```

```
pub
```

```
}
```

```
class Részvény
```

```
{
```

```
    private readonly string részvényténv;
```

```
    private double részvényárfolyam = 0.0;
```

```
    public int Darabszám;
```

```
    public Részvény(string név, double árfolyam, int darabszám) {...}
```

```
partial
```

```
{
```

```
pub
```

```
pub
```

```
pub
```

```
pub
```

```
}
```

```
    public void Vétel(int mennyiség) {...}
```

```
    public void Eladás(int mennyiség) {...}
```

```
    public void ÁrfolyamBeállítás(double árfolyam) {...}
```

```
    public double Érték() {...}
```

Részvény\_KovácsJános.cs

Részvény\_SzabóPéter.cs

## **Feladat 4.**

**Készítsen „Vonatdef” osztályt, melyben A „beolvas” metódusban megadható egy vonat neve, az állomások neve, az indulások ideje (max 100 db), és ezek az osztály adattételeiben tárolódnak.**

**Az üres állomásnév jelentse a beolvasás végét.**

**A „kiír” metódus kiírja azokat a képernyőre.**

**Készítsen „Vonatkez” osztályt, melyben a fenti osztály két példányába beírja az adatokat, majd kiírja azokat a képernyőre.**

# Vonatdef osztály: adattételek

```
class Vonatdef  
{  
    private string vonatnév;  
    private string [,] állomások= new string[100,2];  
    private int hányállomás=0;  
}
```

# Vonatdef osztály: beolvas

```
public void beolvas()
{
    string s;
    System.Console.WriteLine("add meg a vonat nevét! ");
    vonatnév=System.Console.ReadLine();
    for(int i=0;i<100; i++)
    {
        System.Console.WriteLine("add meg az" + (i+1) + "-ik állomás nevét! ");
        s=System.Console.ReadLine();
        if (s=="") return;
        hányállomás++;
        állomások[i,0]=s;
        System.Console.WriteLine("add meg az" + (i+1) +
                                   "-ik állomásról indulás idejét! (oo:pp) ");
        állomások[i,1]=System.Console.ReadLine();
    }
    return;
}
```



# Vonatdef: kiír

```
public void kiír()
{
    System.Console.WriteLine("Vonat neve: „
                               +vonatnév+\".");
    for(int i=0;i<hányállomás; i++)
    {
        System.Console.Write("Állomás neve: "+
                               állomások[i,0]+" ,");
        System.Console.WriteLine("indulás ideje: "+
                                   állomások[i,1]+\".");
    }
    return;
}
```

# Vonatkez osztály

```
class Vonatkez  
{  
  static void Main()  
  {  
    Vonatdef Arrabona=new Vonatdef();  
    Arrabona.beolvas();  
    Arrabona.kiír();  
    Vonatdef Helikon=new Vonatdef();  
    Helikon.beolvas();  
    Helikon.kiír();  
  }  
}
```

## Feladat 4a.

**Készítsen a „Vonatdef” osztályban olyan metódust, mely a vonat nevét képes megváltoztatni.**

# A névváltoztató metódus I.

```
public void névvált()  
{  
    System.Console.WriteLine  
        ("Add meg az új vonatnevet! ");  
    vonatnév=System.Console.ReadLine();  
}
```

## **Feladat 4b.**

**Végezze el ugyanezt a „beolvas” metódot átdefiniálásával.**

# A névváltoztató metódus II.

```
public void beolvas(string vonatnév)
```

```
{
```

```
    this.vonatnév=vonatnév;
```

```
}
```

Ha azonos  
a paraméter és  
az adattétel  
neve

Van  
paraméter

## Használat:

```
Vonatdef Arrabona=new Vonatdef();
```

```
Arrabona.beolvas();
```

```
Arrabona.kiír();
```

```
System.Console.WriteLine("Add meg az új vonatnevet! ");
```

```
Arrabona.beolvas(System.Console.ReadLine());
```

```
Arrabona.kiír();
```

# Másik névváltoztató metódus

```
public void beolvas(int x)
{
    this.vonatnév=System.Console.ReadLine();
}
```

Van  
paraméter

## Használat:

```
Vonatdef Arrabona=new Vonatdef();
Arrabona.beolvas();
Arrabona.kiír();
System.Console.WriteLine("Add meg az új vonatnevet! ");
Arrabona.beolvas(123);
Arrabona.kiír();
```

Mindegy, csak  
int legyen