

Szoftvertervezés és - fejlesztés Labor

10. Laborgyakorlat

Tusor Balázs
tusor.balazs@nik.uni-obuda.hu

ZH-ból tapasztalatok

- OLVASSUK EL A FELADATOT LEGALÁBB 2x!!!
 - Többdimenziós tömbnél ismerjük fel, hogy mátrixról vagy mátrixszal megoldott listáról van szó!
 - Mátrixnak minden eleme ugyanolyan jellegű
 - Mátrixszal megoldott listának különböző típusúak
 - Ha egy mátrixban csak egy koordinátára kell ellenőrizni valamit (x és y paraméterként van megadva), akkor TELJESEN felesleges az egész tömböt 2 for ciklussal bejárni!
-

ZH-ból tapasztalatok

- Ciklusfeltételbe SOHA ne rakjunk `rnd.Next()`-et!! Ekkor minden körben újragenerálja a számot... a generált számot tegyük inkább egy segédváltozóba!
 - Ha százalékos esély a feladat, akkor számoljuk is ki, hogy tényleg annyit határoztunk-e meg!
 - Pl. 15% valószínűség:
 - `esely = rnd.Next(100)` → 15% esély: `if (esely < 15)` //0...14 ← 15 db
 - `esely = rnd.Next(1, 101)` → 15% esély: `if (esely <= 15)` //1...15 ← 15 db
 - Ha egy tömbből akarunk véletlen elemet kiválasztani, akkor a véletlen paramétere a tömb hossza lesz, nem (hossz+1)!
 - Pl. `a = tomb[rnd.Next(tomb.Length)];`
-

Rövid áttekintés – Tulajdonság

```
class Ember
{
    private string nev;

    public string Nev
    {
        get
        {
            return nev;
        }
        set
        {
            nev = value;
        }
    }
}
```

- Az OOP elvek alapján a publikus adattagok használata nem javasolt
- Helyette privát adattaghoz publikus író-olvasó tulajdonságot készítünk
 - **Get:** ha az értékét olvassuk, ez a blokk fut le
 - **Set:** ha írunk a tulajdonságba, ez a blokk fut le (beírt érték a rejtett value paraméter)
- Lehet csak írható és csak olvasható is az adott tulajdonság (szükségtelen blokkot elhagyjuk)

```
static void Main(string[] args)
{
    Ember geza = new Ember();
    geza.Nev = "Nagy Géza";
    Console.WriteLine(geza.Nev);
}
```

Rövid áttekintés – Tulajdonság

```
class Ember
{
    private int életkor;
    public int Eletkor
    {
        get { return életkor; }
        set
        {
            if (value < 0)
            {
                életkor = value * -1;
            }
        }
    }
}
```

- A tulajdonságok set blokkja alkalmas arra, hogy egy kódrészletet írjunk, ami lefut:
 - Az értékadás után
 - Az adattagba bekerülés előtt
- Bonyolult, erőforrásigényes kódokat nem szabad itt elhelyezni

```
static void Main(string[] args)
{
    Ember geza = new Ember();
    geza.Eletkor = -35;
    Console.WriteLine(geza.Eletkor); //35
}
```

Rövid áttekintés – Önálló Tulajdonság

```
class Ember
{
    private int születésiEv;

    public int SzületésiEv
    {
        get { return születésiEv; }
        set { születésiEv = value; }
    }

    public int Eletkor
    {
        get { return 2019 - születésiEv + 1; }
    }
}
```

- Tulajdonságok nem feltétlenül kapcsolódnak adattagokhoz
- Létrehozhatunk csak olvasható tulajdonságot, amely egy kifejezés értékével tér vissza
- A kifejezés a többi adattag/tulajdonság alapján egy „on-the-fly érték”

Rövid áttekintés – Lambda Tulajdonság

```
class Ember
{
    private int életkor;

    public int Eletkor
    {
        get => életkor;
        set => életkor = value;
    }
}
```

- C# 6.0 verzió óta létezik egy ilyen formája is a tulajdonságoknak
- Jellemzői
 - Csak egyszerű értékadás
 - Csak egyszerű kifejezés
 - Bonyolult/több soros kód nem lehetséges
- **Használata ebben a félévben még tilos.**

```
static void Main(string[] args)
{
    Ember geza = new Ember();
    geza.Eletkor = -35;
    Console.WriteLine(geza.Eletkor); //-35
}
```

Rövid áttekintés – Automatikus Tulajdonság

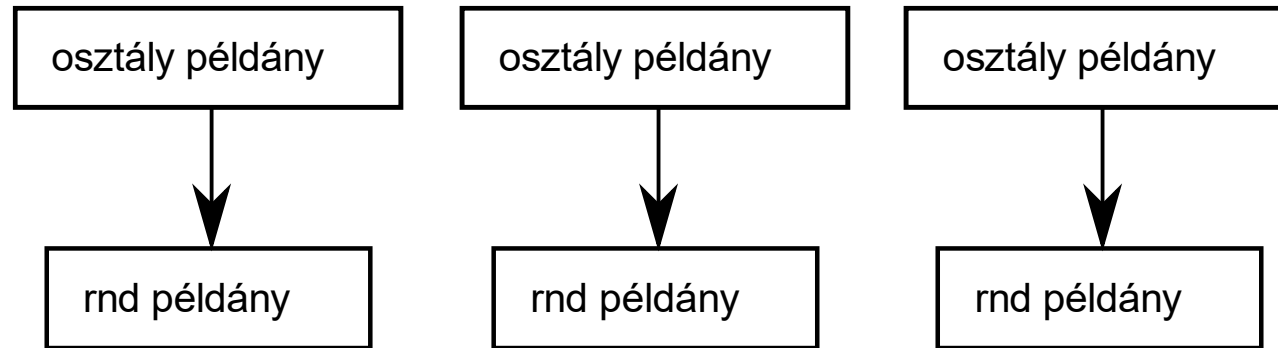
```
class Ember
{
    public string Nev { get; set; }
    public int Eletkor { get; private set; }
    public string Lakohely { get; }
}
```

- Privát adattag és publikus tulajdonság összegyűrve
- Láthatósággal szabályozható
- Getter és setter blokkba nem lehetséges többlet kódot írni
- Használata akkor javasolt, hogyha a tulajdonság célja egyedül az adattag hozzáférés-vezérlése

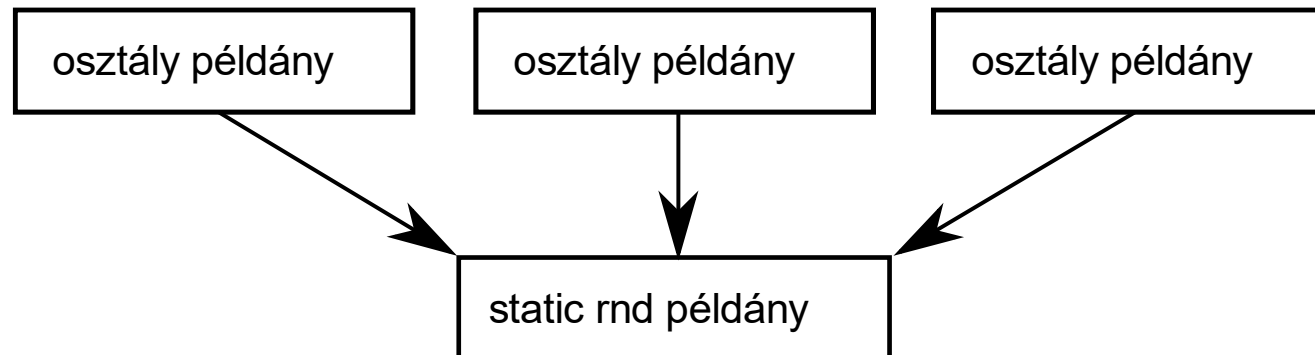
- **Név**
 - Publikusan írható és olvasható
- **Életkor**
 - Publikusan olvasható
 - Csak az osztályon belül írható
- **Lakóhely**
 - Publikusan olvasható
 - Csak az osztályon belül és csak a konstruktorban kaphat értéket

Rövid áttekintés – Statikus Adattag (az osztályon belül)

```
Random rnd = new Random();
```



```
static Random rnd = new Random();
```



1. Feladat – Személy osztály

- 1.1) Hozzuk létre az alábbi adattagokat privát módon, és a hozzájuk tartozó publikus író-olvasó (get-set) tulajdonságot! NE alkalmazzunk lambda tulajdonságot!
 - `static Random rnd = new Random();`
 - Ezáltal kényelmesebben tudunk véletlenszámokat generálni az osztályon belül
 - `string nev`
 - A Nev tulajdonság egyszerűen állítsa be a nev adattag értékét, illetve adja azt vissza
 - `public string SzuletesiDatum`
 - Ezt oldjuk meg getter-setter blokkban! { get; set; }
 - `int irányitoszam`
 - Az előzőhöz hasonlóan, egyszerű beállítás-lekérés
 - `string haziAllatok`
 - Az illető háziállatainak típusait gyűjtjük egy string-ben, pontosvesszővel elválasztva. Mindig, amikor újabb elemmel gyarapodik, ";" kerül hozzáfűzésre, így tekintsük úgy, hogy az utolsó karakter mindig ";". A `public string [] HaziAllatok` tulajdonságnak csak `get` része legyen, ami null-al térjen vissza, ha az adattag üres (hossza 0), és a tárolt háziállatok tömbjével, ha nem. Figyelem: ügyeljünk arra, hogy az utolsó karakter ";"!

```
private string nev;

public string Nev
{
    get
    {
        return nev;
    }
    set
    {
        nev = value;
    }
}
```

1. Feladat – Személy osztály

- 1.2) Hozzunk létre egy olyan konstruktort, amiben beállíthatjuk az illető nevét, születési dátumát és irányítószámát!
 - A háziállatok legyen üres string ("").
-

1. Feladat – Személy osztály

- **1.3) Hozzunk létre egy üres konstruktort, amiben random módon állítjuk be:**
 - A nevét: használjuk a moodle-ben elérhető `NevGenerator()`-t!
 - Születési dátumát `YYYY.MM.DD.` formátumban:
 - Év: véletlenszám $\in [1920, 2020]$
 - Hónap: $\in [1, 12]$
 - Nap: $\in [1, X[hónap - 1]]$, ahol $X = \{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 \}$
 - Vizsgáljuk meg, hogy az év szökőév-e! Ha igen, akkor $X[1]$ -hez hozzá kell adnunk még egyet, hogy a február 29 napos legyen.
 - Irányítószám: $\in [1000, 9999]$
 - Házi állatok: legyen random ($\in [0, 5]$) db háziállata, ennyiszer vegyünk véletlenszerűen egy állatot a következő listáról, és adjuk hozzá a `haziAllatok`-hoz `(+";")`
 - `{ "kutya", "macska", "aranyhal", "aranyhörcsög", "kígyó", "iguána", "papagáj", "mamagáj" }`
 - Ha 0 db háziállatot dobott a random generátor, akkor a string értéke `" "` legyen!
-

1. Feladat – Személy osztály

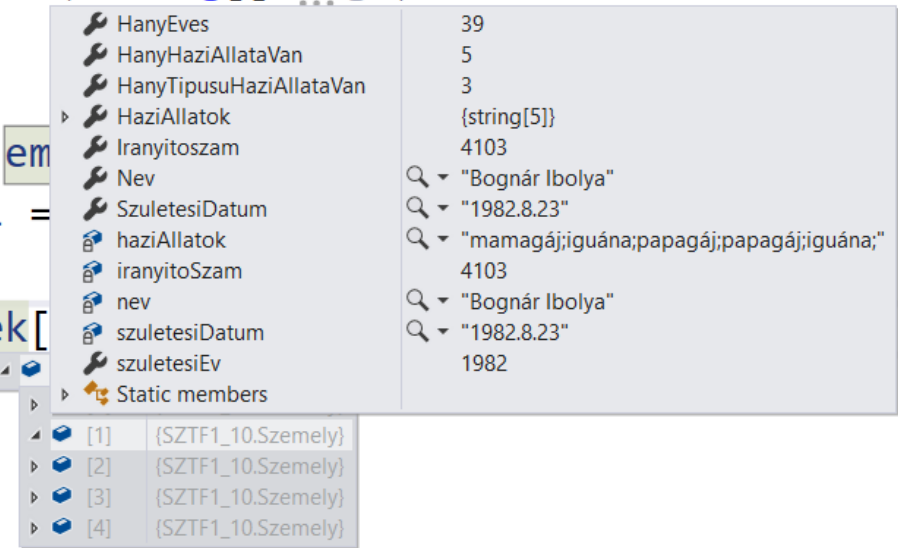
- **1.4) Hozzuk létre az alábbi on-the-fly tulajdonságokat:**
 - `public int születésiEv`
 - Azt adja vissza, hogy melyik évben született az illető (a születési dátumból).
 - `public int HanyEves`
 - Azt adja vissza, hogy hány éves az illető (2021-születésiEv).
 - `public bool VanMacskaja`
 - Térjen vissza annak megfelelően, hogy van-e macskája!
 - `public int HanyHaziAllataVan`
 - Azt adja vissza, hogy hány darab háziállata van az illetőnek. Üres háziállatok sztring esetén 0.
 - `public int HanyTipusuHaziAllataVan`
 - Azt adja vissza, hogy hány különböző fajtájú háziállata van az illetőnek.
 - Egy megoldási mód: használjunk egy segédstringet, amihez hozzáadjuk az egyes háziállat típusokat, ha még nem szerepelnek a stringben!
-

1. Feladat – Személy osztály

■ 1.5) Main

- A Main függvényben hozzunk létre egy 5 személyt tartalmazó tömböt, és töltsük fel 5 random emberrel!
- breakpointtal vagy kiíratással ellenőrizzük, hogy valóban elkészült-e az 5 személy, és az elkészített adattagok, tulajdonságok és konstruktor helyesen működik!

```
static void Main(string[] args)
{
    Szemely[] emberek = new Szemely[5];
    for (int i = 0; i < emberek.Length; i++)
    {
        emberek[i] = new Szemely();
    }
}
```



2. Feladat – Bankszámla osztály

- 2.1) Hozzuk létre az alábbi adattagokat privát módon, és a hozzájuk tartozó publikus író-olvasó (get-set) tulajdonságot! NE alkalmazzunk lambda tulajdonságot!
 - `static` `Random rnd = new Random();`
 - Ezáltal kényelmesebben tudunk véletlenszámokat generálni az osztályon belül
 - `string szamlaszam`
 - A Szamlaszam tulajdonság egyszerűen állítsa be a `szamlaszam` adattag értékét, illetve adja azt vissza
 - `Szemely` tulajdonos
 - Az előzőhöz hasonlóan, egyszerű beállítás-lekérés, annyi különbséggel, hogy a Szemely osztályt használjuk típusként
 - `double egyenleg`
 - Az előzőhöz hasonlóan, egyszerű beállítás-lekérés
 - `string tranzakciok`
 - Ez nagyon hasonló lesz a Szemely osztálynál a háziállatok adattaghoz, annyi különbséggel, hogy egy egész tranzakció adatsorát fogjuk ";"-al elválasztva tárolni.
 - Ugyanúgy, a `public string [] Tranzakciok` tulajdonságnak csak `get` része legyen, ami null-al térjen vissza, ha az adattag üres (hossza 0), és a tárolt tranzakciók tömbjével, ha nem. Figyelem: ügyeljünk arra, hogy az utolsó karakter ";"!
-

2. Feladat – Bankszámla osztály

- 2.2) Írjunk egy metódust az osztályon belül, ami előállítja a számlaszámot:
 - `private string SzamlaszamotEloallit()`
 - 16 db számból álljon, ahol az első számjegy $\in [1,9]$, a többi $\in [0,9]$, a formátuma pedig:
XXXXXXXX-XXXXXXXX
-

2. Feladat – Bankszámla osztály

- **2.3) Ezek birtokában gyakoroljuk egy kicsit a konstruktorokat! Ilyeneket hozzunk létre:**
 - `public Bankszamla(string szamlaszam, Szemely tulajdonos, double egyenleg)`
 - Beállítja a számlaszámot, tulajdonost és egyenleget, a tranzakciók legyen egy üres string.
 - `public Bankszamla(Szemely tulajdonos, double egyenleg)`
 - Beállítja a tulajdonost és egyenleget, a tranzakciók legyen egy üres string, a számlaszámot a 2.2-es feladatban megírt függvénnyel állítsuk elő.
 - `public Bankszamla(Szemely tulajdonos)`
 - Beállítja a tulajdonost, a tranzakciók legyen egy üres string, a számlaszámot a 2.2-es feladatban megírt függvénnyel állítsuk elő, az egyenleg legyen $\in [0.0, 300000.0]$
 - `public Bankszamla()`
 - A tulajdonos legyen egy újonnan (parameter nélkül) előállított személy, a tranzakciók legyen egy üres string, a számlaszámot a 2.2-es feladatban megírt függvénnyel állítsuk elő, az egyenleg legyen $\in [0.0, 300000.0]$
-

2. Feladat – Bankszámla osztály

- 2.4) Írjunk egy metódust az osztályon belül, ami string-ként visszaadja egy bankszámlához tartozó tulajdonos nevét, számlaszámát és egyenlegét:
 - `public string GetString()`
 - 2.5) Írjunk egy fizetést végző metódust az osztályon belül: ha egyenlege engedi, akkor a kapott összeggel csökkentse azt és térjen vissza igazzal, különben hamissal!
 - `public bool Fizet(int osszeg)`
-

2. Feladat – Bankszámla osztály

- 2.6) Írjunk egy metódust az osztályon belül, amivel át tudunk utalni egyik bankszámláról a másikra:
 - `public bool Atutalas(BankszamlA fogado, double osszeg)`
 - Ha nincs elég pénze az illetőnek (akitől meghívjuk a függvényt), akkor csak térjen vissza hamissal
 - Különben a saját egyenlegét csökkentsük $osszeg * 1.01$ -nek megfelelő értékkel, ugyanis a bank is leveszi az átutalási díjat; valamint a fogadó egyenlegét növeljük $osszeg$ -nek megfelelő értékkel
 - A sikeres tranzakciót logoljuk mindkét számla tranzakciós string-jében:
 - A küldő oldalon `"K:fogado.szamlaszam:osszeg;"`
 - A fogadó oldalon `"F:fogado.szamlaszam:osszeg;"` formában!
-

2. Feladat – Bankszámla osztály

▪ 2.7) A Main()-ben:

- Készítsünk egy 5 elemű Bankszámlák tömböt, mindegyikhez a neki megfelelő indexű, korábban elkészített személyt állítsuk be tulajdonosnak! A számlaszám és egyenleg legyen random, azaz azt a konstruktort hívjuk meg, aminek ez felel meg.
 - Írassuk ki a generált bankszámlák adatait!
 - Szimuláljunk 20 darab átutalást:
 - Ennyiszer generáljunk i és j számpárt úgy, hogy $i \neq j \rightarrow i$. bankszámláról j . bankszámlára utaljunk egy max. 100000 ft-nyi összeget! (i és j értékét nyilvánvalóan a bankszámlák tömb hossza korlátozza)
 - Írjuk ki, ha meghíusult a tranzakció!
-

2. Feladat – Bankszámla osztály

- 2.8) Visszatérve a Bankszamlak osztályra, hozzuk létre az alábbi on-the-fly tulajdonságokat:
 - `public int` AtutalasokSzama
 - Azt adja vissza, hogy hány tranzakciója volt az adott bankszámlának.
 - `public double` KapottPenzekOsszege
 - Azt adja vissza, hogy összesen mennyi pénzt kapott az adott bankszámla, vagyis azokra a tranzakciókra kell nézni, melyek első karaktere "F".
-

2. Feladat – Bankszámla osztály

- **2.9) A Bankszamlak osztályon belül hozzuk létre az alábbi metódusokat:**
 - `public string[] KapcsolatosSzamlakIsmetlodesNelkul()`
 - Egy olyan listát ad vissza a tranzakcióiról, melyen egy számlaszám csak egyszer szerepelhet, függetlenül attól, hogy küldő vagy fogadó volt-e.
 - Ismételten elővehetünk egy segédstring-et, amiben ezeket gyűjthetjük
 - Elegendő a bankszámla azonosítóval
 - `public int HanyEmbernekUtalt()`
 - Azt adja vissza, hogy hány olyan egyedi bankszámla van, aminek utalt az adott példány (a tranzakció első karaktere "K"). Itt is figyelniünk kell arra, hogy egy számlát csak egyszer számoljunk bele.
 - `public string[] SzamlakAkiktolTobbszorKapott()`
 - Adja vissza azon string bankszámla azonosítók listáját, melyektől egynél többször kapott átutalást.
-

2. Feladat – Bankszámla osztály

- 2.9) Az osztályon kívül hozzuk létre az alábbi metódusokat:
 - `static string` `KiAlegNagylelkubb(Bankszamlak[] bankszamlak)`
 - Adja vissza azon számlatulajdonos nevét, aki a legtöbb pénzt utalta másoknak.
 - Használjuk a `HanyEmbernekUtaIt()` osztálymetódust!
 - `static string[]` `KiKaptaAlegkevesebbPenzt(Bankszamlak[] bankszamlak)`
 - Keressük meg, hogy melyik bankszámlára érkezett a legkevesebb pénz! (KapottPenzekOsszege tulajdonság)
 - Adjuk vissza egy string tömbben a kapott bankszámla tulajdonosának nevét és azt a pénzmennyiséget, amit kapott!
 - Teszteljük ezeket a main-ből!
-

Feladatok – Extra extra (kalandvágóknak)

Böleny vadász játékot készítünk. A bölenycsorda 10 bölenyből áll, amelyek egy 5x5-ös játéktéren a 0,0 koordinátából indulnak, és az 5,5-be akarnak menni. Minden körben minden böleny véletlenszerűen lép egyet balra (x+1), felfelé (y+1) vagy átlósan balra felfelé (x+1,y+1), a pálya határain belül maradva.

- A felhasználó minden körben adjon be egy lövést.
 - Ha eltalált egy bölenyt, az meghalt.
 - A bölenyek győznek, ha bármelyik elér a célba, és a felhasználó győz, ha minden böleny meghalt.
 - Egy bölenyt egy Böleny típusú objektummal reprezentáljon, amelyben legalább a következő tagok legyenek:
 - X,Y – a böleny aktuális x és y koordinátája
 - Lep() – hatására a böleny lépjen egyet véletlenszerűen
 - Egy konstruktor, amely létrehozza a bölenyt, és egyszer lépteti, hogy ne a 0,0-n kezdjen.
 - Tavolsag() – megadja az adott bölenynek a céltól való távolságát (légvonalban).
 - A 10 darab bölenyt tömbben helyezze el.
 - Minden körben számolja meg és írja ki, hogy a játékos lövése hány bölenyt talált el (ezeket a tömbből vegye ki), aztán léptesse a megmaradt bölenyeket, majd írja ki a célhoz legközelebb lévő bölenynek a céltól való távolságát.
-