Objektumorientált programozás X.

Osztályszintű tagok

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.

Objektumorientált programozás X.

Osztályszintű tagok

Névterek

Gyakorlás

Osztályszintű tagok

- **Példánytól független adattartalom és viselkedés reprezentálható velük**
  - A típushoz (osztályhoz) tartoznak, nem egyedi példányokhoz
  - Elérhetőek akkor is, ha egyetlen példány sincs
- **A static kulcsszóval képezzük őket az osztálydeklaráción belül, és egy adott példány neve helyett a típus nevével hivatkozunk rájuk**

```
static string[] errorMessages; //osztályszintű privát stringtömb
public static int currentConnections;   //osztályszintű publikus int
public static void InitConnections()  { ... } //osztályszintű függvény
```

  - Osztályszintű tag lehet: adattag, metódus, tulajdonság, konstruktor (stb.), sőt maga az osztály is lehet statikus
  - Az eddig ismert (nem static kulcsszóval deklarált) tagok "példányszintűek" voltak

Osztályszintű adattagok

- **Egyetlen másolat létezik belőlük, függetlenül a példányok számától (ellentétben a példányszintű adattagokkal, amikből minden példánynak van egy-egy darab)**
  - Minden const adattag is osztályszintű

```
public class Osztaly
{
    public static int kozos;
    public int sajat;

    public Osztaly(int sajat)
    {
        this.sajat = sajat;
    }

    public void KiirSajat() { Console.WriteLine(sajat); }
    public void KiirKozos() { Console.WriteLine(kozos); }
}
```

```
Osztaly.kozos = 4; //az osztály nevével érjük el
Osztaly o1 = new Osztaly(3);
Osztaly o2 = new Osztaly(6);
o1.KiirSajat();  o1.KiirKozos();
o2.KiirSajat();  o2.KiirKozos();
```

Osztályszintű adattagok

- **Klasszikus példa osztályszintű adattagra: egyedi ID-t adunk minden példánynak**

```
public class Haromszog
{
    double a;
    double b;
    double c;
    double uniqueId; //ennek a háromszögnek az egyedi ID-ja

    public double UniqueId { get { return uniqueId; } }

    static long currentId = 100000;

    public Haromszog(double a, double b, double c)
    {
        this.a = a; this.b = b; this.c = c;
        uniqueId = currentId;
        currentId++;
    }
}
```

```
Haromszog h = new Haromszog();
Console.WriteLine(h.UniqueId);
Haromszog h2 = new Haromszog();
Console.WriteLine(h2.UniqueId);
```

```
h egyedi ID-ja: 100000
h2 egyedi ID-ja: 100001
```

Készítsünk osztályt, amelyről minden pillanatban meg tudjuk mondani, hogy hány példánya létezik éppen!

```
public class Peldanyszamlalo
{
    private static int osszesPeldany = 0;
    public static int OsszesPeldany { get { return osszesPeldany; } }

    public Peldanyszamlalo()
    {
        osszesPeldany++;
    }

    ~Peldanyszamlalo()
    {
        osszesPeldany--;
    }
}
```

```
Peldanyszamlalo peldany = new Peldanyszamlalo();
Peldanyszamlalo[] masikPeldanyok = new Peldanyszamlalo[10];
for (int i = 0; i < masikPeldanyok.Length; i++)
    masikPeldanyok[i] = new Peldanyszamlalo();

Console.WriteLine("összes példány: " + Peldanyszamlalo.OsszesPeldany);
```

Osztályszintű metódusok

- **Osztályszintű metódusok csak osztályszintű tagokkal dolgozhatnak**
  - Csak osztályszintű adattagokat használhatnak, csak ilyen metódusokat hívhatnak stb. – Ha egyáltalán használnak tagot. Sokszor az osztályszintű metódusok csak a paramétereikkel dolgoznak
- **Nincs „this” referencia**
  - this definíció: „referencia az aktuális objektumra”. De itt nincs aktuális objektum, sőt nem biztos, hogy egyáltalán létezik akár egyetlen példány is

Az osztályszintű tulajdonságok getter, setter metódusaira, valamint az osztályszintű konstruktorra ugyanezek a szabályok vonatkoznak.

- **Számtalan példa az osztálykönyvtárban:**

```
i = int.Parse(s); //statikus metódus
Console.WriteLine(s); //statikus metódus
```

  - Vagy pl. a főprogram megvalósítása is egy tetszőleges osztály Main nevű statikus metódusával történik

Osztályszintű metódusok

```
class Vektor
{
    private double x;
    public double X { get { return x; } set { x = value; } }
    private double y;
    public double Y { get { return y; } set { y = value; } }

    public Vektor(double x, double y)
    {
        this.x = x; this.y = y;
    }

    public static Vektor Parse(string str)
    {
        int vesszo = str.IndexOf(',');
        int x = int.Parse(str.Substring(0, vesszo));
        int y = int.Parse(str.Substring(vesszo+1));
        return new Vektor(x,y);
    }
}
```

```
Console.WriteLine("Írj be egy vektort (x,y formában): ");
string s = Console.ReadLine();
Vektor v = Vektor.Parse(s);
```

Osztályszintű metódusok

- **Sok esetben ugyanazt a funkcionalitást statikus és nem statikus formában is meg tudjuk valósítani**

```
class Vektor
{
    private double x;
    public double X { get { return x; } set { x = value; } }
    private double y;
    public double Y { get { return y; } set { y = value; } }
    ...
    public Vektor Osszead(Vektor b)
    {
        return new Vektor(x + b.x, y + b.y);
    }

    public static Vektor Osszead(Vektor a, Vektor b)
    {
        return new Vektor(a.x + b.x, a.y + b.y);
    }
}
```

```
Vektor v = new Vektor(1, 1);
Vektor v2 = new Vektor(2, 3);
Vektor sum = Vektor.Osszead(v, v2);
Vektor sum2 = v.Osszead(v2);
Console.WriteLine(sum.X + "," + sum.Y);
Console.WriteLine(sum2.X + "," + sum2.Y);
```

```
3,4
3,4
```

Osztályszintű vagy példányszintű metódus?

- **Legyen példányszintű, ha az adott példány állapotát módosítja**

```
public void Szoroz(double s)
{
    x = x * s;
    y = y * s;
}
```

- **Legyen osztályszintű, ha az adott példánytól független a működés, amit reprezentál**
- **Gyakran bizonytalan**
  - Ízlés és a fejlesztői közösség szokása dönthet, sokan a példányszintű metódusokat preferálják ilyenkor
- **Egyéb szempontok:**
  - Ha bárhonnan, bármikor el kell tudni érni (kisebb, gyakran használt segédfüggvények), akkor sokszor statikus (pl. Math osztály függvényei)
  - A statikus függvények kicsit gyorsabbak
    - Szinte soha nem jelentős a teljesítmény, amit nyerünk így
  - Bizonyos esetekben kötelező egy adott metódust statikusnak megírni
    - Pl. főprogram, operátorok, kiegészítő metódusok stb.

Statikus osztály

- **A static kulcsszó az osztálydeklaráció előtt azt jelzi, hogy az osztály csak osztályszintű tagokat tartalmaz**
  - Nem példányosítható (vagyis nem használhatjuk egy változó típusául)
  - Nem vesz részt az öröklésben (később)
  - Betöltődése automatikusan történik legkésőbb az első használat előtt, ismeretlen időben
  - A memóriában marad, amíg a program fut
- **Példák az osztálykönyvtárból:**
  - System.Console
  - System.Convert
  - System.Math

Statikus osztály

```
public static class Math
{
    public const double E = 2.7182818284590451;
    public const double PI = 3.1415926535897931;
    ...
    public static int Max(int val1, int val2)
    {
        if (val1 < val2)
        {
            return val2;
        }
        return val1;
    }
    ...
    public static int Sign(int value)
    {
        if (value < 0)
        {
            return -1;
        }
        if (value > 0)
        {
            return 1;
        }
        return 0;
    }
    ...
}
```

Statikus konstruktor

- **Osztályszintű tagok inicializálására használjuk**
  - Statikus és nem statikus osztályokban egyaránt lehet ilyen
  - Ugyanaz a neve, mint az osztályé
  - Nincsenek paraméterei, sem láthatósága
  - Kézzel nem hívható, automatikusan hívódik az első példány létrejötte előtt, vagy bármelyik osztályszintű tagra való hivatkozáskor (ismeretlen időben)

```
class Osztaly
{
    static Osztaly() { ... }
}
```

- **Statikus destruktor nincs!**
  - Ha szükség lenne rá (pl. nyitott fájlt kezelünk, amit szeretnénk mindenképp lezárni), akkor ne használjunk statikus osztályt, más megoldást kell választani (pl. singleton pattern)

Objektumorientált programozás X.

Osztályszintű tagok

Névterek

Gyakorlás

Névterek

- **A névterek az elnevezések tetszőleges logikai csoportosítását teszik lehetővé**
  - **Nincs közük 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();
```

```
namespace A.B
{
   class Másik {…}
}
namespace C
{
   class Harmadik {…}
}
…
A.B.Másik példa2 = new A.B.Másik();
C.Harmadik példa3 = new C.Harmadik();
```

  - A névterek nevének Microsoft által javasolt formátuma:
    <Company>.(<Product>|<Technology>)[.<Feature>][.<Subnamespace>]

Névterek

- **Minden névre a saját névterével együtt kell hivatkozni**

```
System.Console.WriteLine();
System.Threading.Thread.Sleep(100);
pi = System.Math.PI;
```

  - Ez az ún. ***teljesen minősített név*** (fully qualified name), formája: névtér.alnévtér.alnévtér(...).elnevezés

- **A névterek 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 álnév is adható**
  - Célja a hosszú névterek egyértelmű rövidítése

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

Objektumorientált programozás X.

Osztályszintű tagok

Névterek

Gyakorlás

Objektumtömb

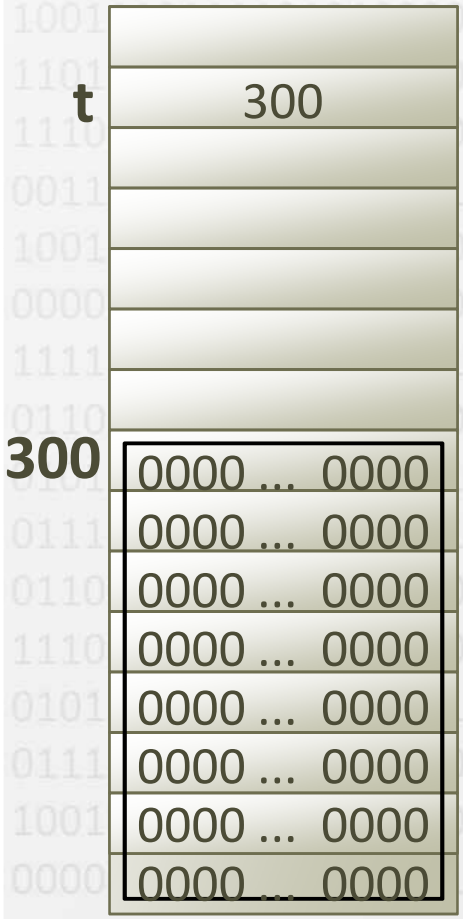
- **Ismétlés: tömblétrehozás**

```
int[] t = new int[8];
```

```
string[] t = new string[8];
```

```
char[] t = new char[8];
```

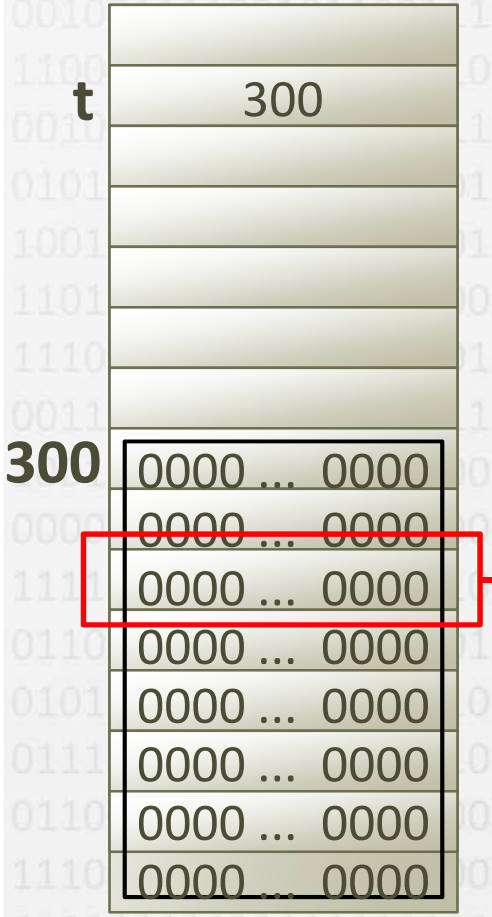
```
Osztaly[] t = new Osztaly[8];
```



- A tömb referenciatípus, így a változóban nem közvetlenül az adat helyezkedik el, hanem egy memóriacím (referencia)
- Az adott memóriacímen helyezkedik el az adat maga
- **Tömb létrehozásakor a tömb által lefoglalt adatterület kinullázódik a memóriában**

Objektumtömb

- **Ismétlés: alapértelmezett érték**



– A tömbelemben a nullázás után a típushoz tartozó alapértelmezett érték van:

– **Ha *érték típusú* változók tömbje volt, akkor közvetlenül az adat van az elemben:**
  - Ha a tömb int[] volt: 0
  - Ha a tömb char[] volt: \0 (a 0 kódú karakter)
  - Ha a tömb bool[] volt: false

– **Ha viszont *referencia típusú* változók tömbje volt, akkor ez a csupa 0 valamilyen memóriacím (referencia)**
  - Nem érvényes memóriacím
  - **null**-nak nevezzük
  - **Null elemre nem lehet hivatkozni**

Objektumtömb

- **Ismétlés: alapértelmezett érték**
  - Érték típusú változókat tartalmazó tömbök:

```
int[] t = new int[8];
```

```
char[] t = new char[8];
```

    - A tömbelemek értéke a létrehozás után érvényes, azonnal használható

```
int[] t = new int[8];
t[2]++;
Console.WriteLine(t[2]); //1
```

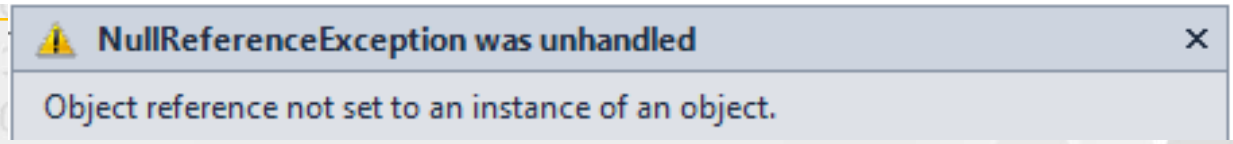
  - Referencia típusú változókat tartalmazó tömbök:

```
string[] t = new string[8];
```

```
Osztaly[] t = new Osztaly[8];
```

    - **A tömbelemek értéke a létrehozás után null, érvénytelen**
    - Ezért értéket kell adni minden tömbelemnek, mielőtt használatba vesszük

```
Osztaly[] t = new Osztaly[8];
Console.WriteLine(t[2].Tulajdonsag);  //NEM JÓ!!!
```

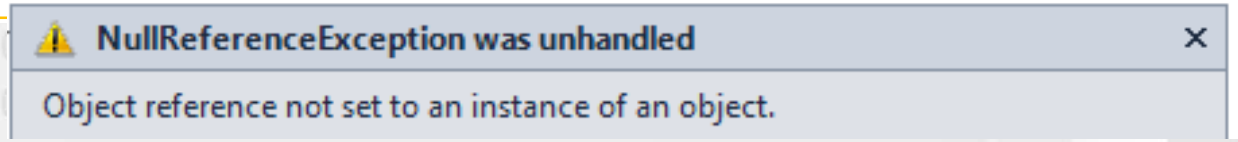


Objektumtömb

- **Objektumtömbben példányosítani kell minden elemet**

```
class Vektor
{
    private double x;
    public double X { get { return x; } set { x = value; } }
    private double y;
    public double Y { get { return y; } set { y = value; } }
    ...
    public Vektor(double x, double y)
    {
        this.x = x; this.y = y;
    }
}
```

```
Vektor[] tomb = new Vektor[8];
Console.WriteLine(tomb[2].X + " " + tomb[2].Y); //NEM JÓ!!!
```

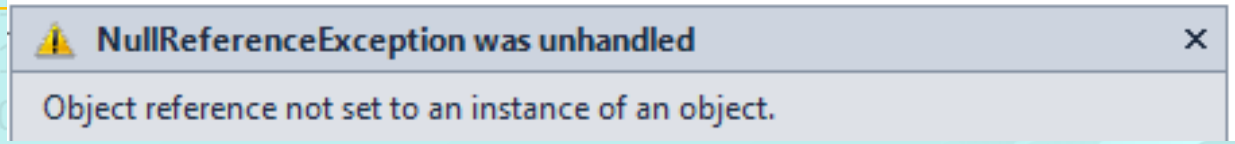



```
Vektor[] tomb = new Vektor[8];
for (int i = 0; i < tomb.Length; i++)
{
    tomb[i] = new Vektor(5, 5);
}
Console.WriteLine(tomb[2].X + " " + tomb[2].Y); //JÓ: 5 5-öt ír ki
```

Objektumtömb

- **A string is referenciatípus**
  - Speciális tulajdonsága miatt számos esetben érték típusként viselkedik
- **Ezért az alapértelmezett érték ott is null**
- **A stringtömbben is értéket kell adni minden elemnek:**

```
string[] t = new string[8];
Console.WriteLine(t[2].ToUpper()); // NEM JÓ!!!
```




```
string[] t = new string[8];
for (int i = 0; i < tomb.Length; i++)
{
     t[i] = "ffff";
}
Console.WriteLine(t[2].ToUpper()); // FFFF
```

Gyakorló feladat

Csevegőprogramot készítünk, amelyben Személy típusú elemekkel reprezentáljuk a kontaktjainkat. Egy Személynek van neve, születési éve és neme. Hozzunk létre 5 elemű tömböt, amelyet feltöltünk Személyekkel! A Személy osztályban legyen olyan paraméteres konstruktor, amelynek segítségével név és nem ismeretében, de véletlenszerű születési évvel létrehozhatjuk a személyt.

Vannak olyan emberek, akiknek ugyanaz a vezeték- vagy keresztnevük, mint a felhasználónak? Ha vannak, akkor listázzuk ki mindet.

Listázzuk ki az összes olyan Személyt, aki a felhasználóval ellenkező nemű!

Számoljuk meg, hány olyan Személy van, aki a felhasználóval azonos korosztályba tartozik (azaz maximum 5 év van közöttük)!

Hozzunk létre egy 20 elemű tömböt, amelyben Zh típusú elemek vannak. Egy ilyen elemmel egy zh-eredményt reprezentálunk. Egy Zh-hoz egy 6 jegyű Neptun-kód és egy 0-100-ig terjedő pontszám tartozik. Egy konstruktor segítségével véletlenszerű Neptun-kóddal és pontszámmal hozzuk létre a tömbben lévő példányokat.

A Neptun-kód egy string, amely kezdetben üres. Mind a 6 karakterét legeneráljuk egyenként, a következőképpen:

- Véletlenszám sorsolásával eldöntjük, szám jön-e vagy karakter.
- Ha szám, akkor sorsolunk egy számot 0-9-ig, és hozzáadjuk a stringhez.
- Ha karakter, akkor sorsolunk egy karaktert, és azt hozzáadjuk a stringhez.

Karakter sorsolása:

```
char c = (char)rand.Next(65, 91);
```

Sorolja fel, kik mentek át a zh-n!

Írja ki a legjobb eredményű hallgató Neptun-kódját! Ha több embernek is ez az eredménye, akkor mindegyikükét.

Listázza ki az eredményeket úgy, hogy a nevek mellett a jegyek szerepelnek! A jegyeket ponthatártömb alapján állapítsa meg.

Bölényvadász játékot készítünk. A bölénycsorda 10 bölényből áll, amelyek egy 5x5-ös játéktéren a 0,0 koordinátából indulnak, és az 5,5-be akarnak menni. Minden körben minden bölény 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 bölényt, az meghalt. A bölények győznek, ha bármelyik elér a célba, és a felhasználó győz, ha minden bölény meghalt.

Egy bölényt egy Bölény típusú objektummal reprezentáljon, amelyben legalább a következő tagok legyenek:

- **X,Y – a bölény aktuális x és y koordinátája**
- **Lep() – hatására a bölény lépjen egyet véletlenszerűen**
- **Egy konstruktor, amely létrehozza a bölényt, és egyszer lépteti, hogy ne a 0,0-n kezdjen.**
- **Tavolsag() – megadja az adott bölénynek a céltól való távolságát (légvonalban).**

A 10 darab bölényt tömbben helyezze el.

Minden körben számolja meg és írja ki, hogy a játékos lövése hány bölényt talált el (ezeket a tömbből vegye ki), aztán léptesse a megmaradt bölényeket, majd írja ki a célhoz legközelebb lévő bölénynek a céltól való távolságát.

Segítség otthoni gyakorláshoz

- **A randomgenerátor egy kiinduló egész szám, ún. Seed alapján generál egymás utáni számokat. Azonos Seeddel létrehozott randomgenerátorok azonos számsorozatot generálnak.**
- **A C#-os randomgenerátor Seedje a létrehozási idő.**
- **Ez azt jelenti, hogy ha „túlságosan egyszerre” hozunk létre randomgenerátorokat, akkor azok ugyanazokat a számokat generálják.**
- **Ha eddig tömegesen hoztunk létre példányokat úgy, hogy a konstruktorban volt a randomgenerátor, akkor pl. tegyük inkább át (a generátort) egy static adattagba.**
  - Így nem különböző generátorokat használunk, hanem egyetlen azonosat minden példányban. Ezzel nem lesz gond.

Tervezési gyakorlatok

Készítsen Pont2D osztályt, amely alkalmas egy 2D pont tárolására (x,y koordináták).

Készítsen Szakasz osztályt. A szakaszt két Pont2D határozza meg. Tudni kell meghatározni a szakasz hosszát és a felezőpontját.

Készítsen ElemN osztályt, amelyben pontosan N darab egész számot tárolhat úgy, hogy nem szerepelhet egynél többször ugyanaz a szám. Nem lehet belőle elemet kivenni, viszont (egy valahanyadik elemet) másra kicserélni igen. Lehessen egy valahanyadik elemet lekérni, illetve ellenőrizni, hogy egy adott elem benne van-e az ElemN-esben vagy sem.

Az ElemN-est mindig N különböző elemmel hozzuk létre (itt nem kell hibakezelés, feltételezhetjük, hogy az N darab kezdőelem tényleg különböző). Utána a kapacitása nem változik. Le kell tudni kérni, hogy mennyi elemből áll az ElemN-es.

Készítsen Mátrix osztályt, amely tetszőleges méretű lehessen. Lehessen ellenőrizni, hogy nullmátrix-e és szorozható-e egy másik mátrixszal, illetve legyen képes transzponálásra (a mátrix transzponáltjaként egy másik Mátrix jöjjön létre, tehát az eredeti megmarad). (A szorzást nem kell megvalósítania). Lehessen lekérni egy adott elemet.

Készítsen egyszerű Tároló osztályt, amelyben pozitív egész számokat tárolhat. A Tárolónak legyen meghatározott kapacitása. Elemeket lehet beletenni és kivenni belőle. Lehessen ellenőrizni, hogy egy adott elem benne van-e vagy sem. A Tárolóról bármikor meg lehet tudni, hogy tele van-e, illetve ki kell tudni üríteni.