

# Opjektum Orientált Programozás

## KONSTRUKTOROK

Készítette: Vastag Atila

2017

Minden esetben amikor példányosítunk egy osztályt egy speciális metódus a **konstruktor** fut le, melynek feladata, hogy „beállítsa” az osztály értékeit. Bár az eddigiekben bemutatott osztályunkban, **Negyzet**, nem definiáltunk semmi ilyesmit ettől függetlenül rendelkezik alapértelmezett (azaz paraméter nélküli) konstruktorral amelyet a fordító automatikusan hoz létre.

A konstruktor feladata, hogy olyan adattal lássa el az osztályt amely nélkül nem tud létezni.

(létrehozható e a négyzet oldal nélkül?!)

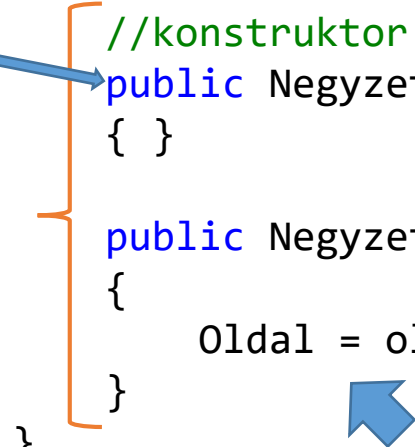
igen – da van értelme 0 egységnyi nagyságú négyzetnek ?!

A konstruktor neve meg kell egyezzen az osztály nevével és semmilyen visszatérési értéke nem lehet.

```
public class Negyzet
{
    public double Oldal { get; set; }

    //konstruktor
    public Negyzet()
    { }

    public Negyzet(double oldal)
    {
        Oldal = oldal;
    }
}
```



beállítódik az osztály  
valamelyik adattagja

Egy osztálynak paraméterlistától függően bármennyi konstruktora lehet és egy konstruktorból hívhatunk egy másikat a *this*-szel:

```
Negyzet negyzet = new Negyzet();
```

```
Negyzet negyzet = new Negyzet(5);
```

```
public class Negyzet
{
    public double Oldal { get; set; }

    //konstruktor
    public Negyzet(): this(0)
    { }

    public Negyzet(double oldal)
    {
        Oldal = oldal;
    }
}
```

Ha több konstruktor is van, akkor a paraméter típusához leginkább illeszkedő fut le.

```
Negyzet negyzet = new Negyzet(5);
Negyzet alakzat = new Negyzet(5, 10);

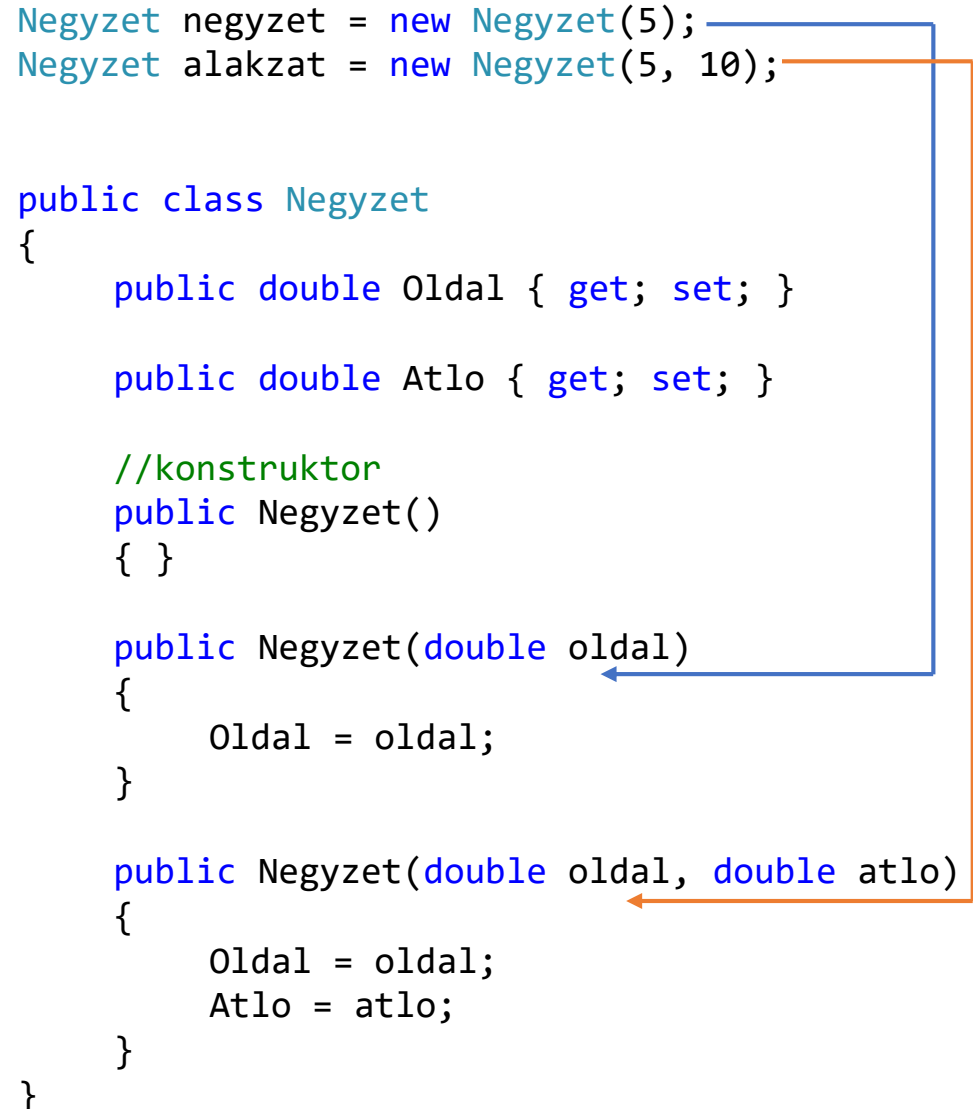
public class Negyzet
{
    public double Oldal { get; set; }

    public double Atlo { get; set; }

    //konstruktor
    public Negyzet()
    { }

    public Negyzet(double oldal)
    {
        Oldal = oldal;
    }

    public Negyzet(double oldal, double atlo)
    {
        Oldal = oldal;
        Atlo = atlo;
    }
}
```



Nem csak a konstruktorban adhatunk értéket az adattagoknak, hanem használhatunk ún. inicializálókat is:

```
public class Negyzet
{
    public double Oldal { get; set; } = 0;

    public double Atlo { get; set; } = 0;

    //konstruktor
    public Negyzet()
    { }

    public Negyzet(double oldal)
    {
        Oldal = oldal;
    }

    public Negyzet(double oldal, double atlo)
    {
        Oldal = oldal;
        Atlo = atlo;
    }
}
```

```
Negyzet negyzet = new Negyzet(5);
negyzet.Oldal = 5;
```

Az inicializálás mindig a konstruktor előtt fut le, ez egyben azt is jelenti, hogy az utóbbi felülbíráhatja. Ha a **Negyzet** osztálynak ezt a módosított változatát használtuk volna fentebb, akkor a példányosítás során minden esetben felülírnánk az alapértelmezettnek megadott kort.

Az inicializálás sorrendje megegyezik a deklaráció sorrendjével (felülről lefelé halad).

A konstruktorok egy speciális változata az ún. másoló- vagy *copy-konstruktor*.

Ez paramétereként egy saját magával megegyező típusú objektumot kap és annak értékeivel inicializálja magát, *de lehet más objektum is paraméter.*

```
public class Negyzet
{
    public double Oldal { get; set; } = 0;

    public double Atlo { get; set; } = 0;

    //konstruktorok
    public Negyzet()
    { }

    public Negyzet(double oldal)
    {
        Oldal = oldal;
    }

    public Negyzet(double oldal, double atlo)
    {
        Oldal = oldal;
        Atlo = atlo;
    }

    public Negyzet(Negyzet negyzet)
    {
        Oldal = negyzet.oldal;
        Atlo = negyzet.atlo;
    }
}
```

Írj osztályt, ami egy Macska objektumot valósít meg.

- A macska adattagjai a következők legyenek: név (string), súly (double), éhes -e (boolean).
- Két konstruktort is készíts az osztályhoz. Az egyik általános legyen, ami minden adattagot a konstruktor paraméterlistájából állít be, illetve egy másik, ami az első két adattagot a konstruktor paraméterlistájából kapja, és alapértelmezetten legyen éhes a macska.
- Az osztálynak legyen egy **Eszik** metódusa, ami egy *double* értéket vár (étel mennyisége), és egy *bool*-al tér vissza (sikeres volt -e az etetés). Ha a macska éhes, az etetés sikeres, és a súly nőjön az étel mennyiségével. A macska ezután ne legyen éhes. Ha a macska nem éhes, az etetés nem sikeres.
- Az osztálynak legyen egy *void* **Futkos** metódusa, ami nem vár paramétert. A macska súlya csökkenjen 0.1-el, és ha nem volt éhes, akkor éhezzen meg.
- A main metódusban hozz létre két macskát a két különböző konstruktorral, és próbáld meg megetetni őket. Az etetés sikerességéről írd ki információt konzolra.
- Mindkét macska futkosson, és utána írd ki szövegesen az objektumokat

Írj osztályt, ami egy *Szamitogep* objektumot valósít meg.

- A számítógép adattagjai a következők legyenek: szabad memória MB-ban (double), be van -e kapcsolva (boolean).
- Készíts két konstruktort is az osztályhoz. Az egyik általános legyen, ami minden adattagot a paraméterlistából állít be, a másik egy alapértelmezett konstruktor legyen, ami 1024 MB memóriával, kikapcsolva hozza létre a gépet.
- Az osztálynak legyen egy *void* **Kapcsol** metódusa, ami nem vár paramétert. Ha a gép ki van kapcsolva, akkor kapcsolja be, egyébként kapcsolja ki.
- Az osztálynak legyen egy *bool* **ProgramMasol** metódusa, ami egy program méretét várja paraméternek MB-ban (double). Ha a program ráfér még a gépre, és a gép be van kapcsolva, úgy csökkenjen a szabad memória a program méretével. A metódus térjen vissza *bool* változóval, hogy sikeres volt -e a másolás.
- Készíts **ToString** metódust az osztályhoz.
- A *main* metódusban hozz létre két számítógépet a fenti konstruktorokkal. Mindkét gép kikapcsolt állapotban kezdjen. Az alapértelmezett gépet kapcsold be, és másold rá először 800 MB, aztán 400 MB programot. A másik gépre másolj 1 MB programot. A másolások eredményeit írd ki.
- Mindkét objektumot írd ki szövegesen.

Írj osztályt, ami egy Hallgato objektumot valósít meg.

- A hallgató adattagjai a következők legyenek: azonosító (string), évfolyam (int), kreditszám (int).
- Két konstruktort is készíts az osztályhoz. Az egyik általános legyen, ami minden adattagot a konstruktor paraméterlistájából állít be, illetve egy másik, ami az első adattagot a konstruktor paraméterlistájából kapja, évfolyama 1 és kreditszáma 0 legyen.
- Az osztálynak legyen egy *void* **TargyFelvesz** metódusa, amivel egy int paramétert (tárgy kreditértéke) kér. A hallgató kreditszáma nőjön a kapott értékkel.
- Az osztálynak legyen egy *bool* **Vizsgazik** metódusa, ami nem vár paramétert. Ha a hallgatónak 0-nál több kreditje van, akkor a sikeres a vizsga: a következő évfolyamba lép, és nullázódik a kreditszáma. Egyébként a vizsga sikertelen.
- Készíts **ToString** metódust az osztályhoz.
- A main metódusban hozz létre két hallgatót a két különböző konstruktorral. Az egyikük vegyen fel tárgyat, majd vizsgáztasd őket. A vizsga sikerességéről írd ki információt konzolra.
- Ezután mindkét hallgatót írd ki szövegesen.



Írj osztályt, ami egy *Torta* objektumot valósít meg.

- A torta adattagjai a következők legyenek: emeletek száma (int), meg van-e kenve krémmel (boolean).
- Készíts két konstruktort is az osztályhoz. Az egyik általános legyen, ami minden adattagot paraméterlistából állít be, a másik egy alapértelmezett konstruktor legyen, ami 1 emeletes, krém nélküli tortát hoz létre.
- Az osztálynak legyen egy void **UjEmelet** metódusa, ami nem vár paramétert, és egy új emeletet rak a tortára.
- Az osztálynak legyen egy bool **KremmelMegken** metódusa, ami nem vár paramétert. Ha a torta még nincs megkenve krémmel, úgy a metódus tegye ezt meg. Térjen vissza logikai értékkel attól függően, hogy sikerült-e.
- Készíts egy int típusú **MennyiKaloria** metódust az osztályhoz. A torta minden emelete 1000 kalória értékű, ha még krémmel is meg van kenve, akkor ennek a kétszerese.
- Készíts **ToString** metódust az osztályhoz.
- A *main* metódusban hozz létre két tortát a két konstruktorral. Az alapértelmezett tortát kétszer is kend meg krémmel, ennek eredményét mindig írd konzolra. A másik tortára rakj egy emeletet.
- Mindkét objektumot írd ki szövegesen.