Objektumorientált programozás IX.

Osztályok, objektumok

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

- **Belső adatok és a rajtuk végezhető műveletek által alkotott egységes adattípus**
  - OOP paradigma alapeleme
- **Az osztályok deklarálása a class kulcsszó segítségével történik**

```
class ComplexNumber
{
    double real;
    double imaginary;

    double Abs()
    {
        return Math.Sqrt(Math.Pow(real, 2) +
                         Math.Pow(imaginary, 2));
    }
}
```

Osztályok alkotóelemei I.

```
class ComplexNumber
{
    public double real;
    public double imaginary;

    public double Abs()
    {
        return Math.Sqrt(Math.Pow(real, 2) +
                         Math.Pow(imaginary, 2));
    }
    ...
}
```

Adattagok

Metódusok (függvények)

Adattagok

- **Az adattagok értéke a deklarációval egyszerre, helyben is megadható (inicializálás)**

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

- **Az adattagok lehetnek**
  - Olvasható-írható adattagok
    - Értékük tetszés szerint olvasható és módosítható az osztály függvényei által
    - Kívülről a láthatóság függvényében
  - Csak olvasható adattagok
    - Értékük futásidőben áll be, konstruktorban vagy helybeni inicializációval
    - Utána nem módosíthatók

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

  - Konstans adattagok
    - Értéküket a fordítóprogram előre letárolja, futásidőben nem módosíthatók

```
const double π = 3.14159265;
const int összeg = 23 * (45 + 67);
```

Metódusok

- **Minden metódus tagja egy osztálynak (tagfüggvény)**
- **A metódusok rendelkeznek:**
  - Megadott, nulla vagy változó darabszámú paraméterrel (params kulcsszó). A paraméterátadás módja in (jelöletlen), out vagy ref.

```
void EgyparaméteresMetódus(bool feltétel) { ... }
void TöbbparaméteresMetódus(int a, float b, out string c) { ... }
void MindenbőlSokatElfogadóMetódus(params object[] paraméterTömb) { ... }
```

  - Visszatérési értékkel (ha nincs: void)

```
void NincsVisszatérésiÉrtékem() { ... }
int EgészSzámotAdokVissza(float paraméter) { ... }
string VálaszomEgyKaraktersorozat(string bemenőAdat) { ... }
SajátTípus Átalakító(SajátTípus forrásObjektum, int egyikAdattagÚjÉrtéke,
  string másikAdattagÚjÉrtéke) { ... }
```

- **A paraméterek és a visszatérési értékek határozzák meg azt a protokollt, amelyet a metódus használatához be kell tartani**
  - A visszatérési értéket és a paramétereket magában foglaló, a kifejtést nem tartalmazó megadási formát szokták a függvény szignatúrájának nevezni

```
void TöbbparaméteresMetódus(int a, float b, string c);
```

Példányosítás

- **A class kulcsszó után megadott felépítés csak a típus leírása.**
- **Ahhoz, hogy egy ilyen típusú adatot használni tudjunk a programunkban, létre kell hozni a típus egy példányát (példányosítás/instantiation):**
  - Gyakran nevezzük a példányt objektumnak is

```
ComplexNumber c = new ComplexNumber();
c.real = 20;
c.imaginary = 2;
Console.WriteLine(c.Abs());
```

- **Tagok elérése: „ . ” operátorral**
- **A new operátor elvégzi az objektum számára a memóriafoglalást, és meghívja a megfelelő konstruktort**

Osztályok alkotóelemei II.

```
class ComplexNumber
{
    public double real;
    public double imaginary;

    public ComplexNumber(double r, double i)
    {
        real = r;
        imaginary = i;
    }

    public double Abs()
    {
        return Math.Sqrt(Math.Pow(real, 2) + Math.Pow(imaginary, 2));
    }
}
```

Adattagok

Konstruktorok

Metódusok

```
ComplexNumber c = new ComplexNumber(20, 2);
Console.WriteLine(c.Abs());
```

Konstruktorok I.

- **Speciális függvény, amely a példány létrehozásakor fut le, főként inicializációs feladatokat végez**
  - A konstruktorok neve mindig megegyezik az osztály nevével
  - Visszatérési értéke nincs (void sem!)
  - A new kulcsszón kívül máshogyan kívülről nem hívható
    - De egy konstruktor a this vagy base kulcsszavak segítségével meghívhat másik konstruktort
- **Ha mi magunk nem deklarálunk konstruktort, a C# fordító automatikusan létrehoz egy paraméter nélkülit (alapértelmezettet):**
  - Az első példában a fordító által létrehozott alapértelmezett konstruktort hívtuk:

```
ComplexNumber c = new ComplexNumber();
```

  - Az előző példában egy általunk definiáltat:

```
ComplexNumber c = new ComplexNumber(20, 2);
```

Konstruktorok II.

- Lehet több konstruktora is egy osztálynak – eltérő paraméterlistával

```
class ComplexNumber
{
    public double real;
    public double imaginary;

    public ComplexNumber(double r, double i)
    {
        real = r;
        imaginary = i;
    }

    public ComplexNumber(double r)
    {
        real = r;
        imaginary = 0;
    }
    ...
}
```

```
ComplexNumber c = new ComplexNumber(20);      //JÓ
ComplexNumber c = new ComplexNumber(2, 20);   //JÓ
ComplexNumber c = new ComplexNumber();        //NEM JÓ
```

2 'OOP_eleje.ComplexNumber' does not contain a constructor that takes 0 arguments

Példány élettartama, destruktor

- **Addig él, amíg van rá referencia**
- **Nem kell (nem is lehet) felszabadítanunk az általa elfoglalt helyet a memóriában – ezt az automatikus szemétgyűjtő (GC) teszi meg a már nem „élő” objektumoknál**
  - Destruktor:

```
~ComplexNumber()
{
    ...
}
```

    - Akkor fut le, ha a példány megsemmisül
    - Fő feladata: erőforrás-felszabadítás
    - Nem hívható, a GC hívja *ismeretlen időben* - azonnali felszabadítást (pl. hálózati vagy adatbázis-kapcsolatok, fájlok lezárása...) más módon kell végezni (Dispose pattern)
    - C#-ban a felesleges destruktorok overhead-et okoznak a GC működése miatt – kerüljük, és üres destruktort sohasem írunk

This

```
public ComplexNumber(double real, double imaginary)
{
      real = real;
      imaginary = imaginary;
}
```

⚠	1	Assignment made to same variable; did you mean to assign something else?
⚠	2	Assignment made to same variable; did you mean to assign something else?

- **A helyi változók elrejtik az osztály saját, azonos nevű adattagjait**
- **this: referencia az aktuális objektumra**

```
public ComplexNumber(double real, double imaginary)
{
    this.real = real;
    this.imaginary = imaginary;
}
```

- **Használjuk még pl.:**
  - Elrejtett adattagok elérésére
  - Az aktuális objektum paraméterként való átadásakor
  - Másik saját konstruktor hívásakor

Gyakorló feladat

Készítsünk Háromszög osztályt, amely egy háromszöghöz tárolja a három oldal hosszúságát! Legyen képes kerület, terület számítására.

```
class Haromszog
{
    public double a; //egyelőre mindenhol public – mindjárt lesz róla szó
    public double b;
    public double c;

    public Haromszog(double a, double b, double c)
    {
        this.a = a; this.b = b; this.c = c;
    }

    public double Kerulet()
    {
            return a + b + c;
    }

    public double Terulet() //Hérón-képlet
    {
        double s = Kerulet() / 2;
        return Math.Sqrt(s * (s - a) * (s - b) * (s - c));
    }
}
```

Láthatóságok

```
class ComplexNumber
{
    public double real;
    public double imaginary;

    public ComplexNumber(double r, double i)
    {
        real = r;
        imaginary = i;
    }

    public double Abs()
    {
        return Math.Sqrt(Math.Pow(real, 2)
                         + Math.Pow(imaginary, 2));
    }
}
```

- Lehetnek olyan függvények, adattagok stb., amiket csak osztályon belül használunk
- Lehetnek olyanok, amelyeket kívülről is szabadon el lehet érni

Láthatóságok

- **Alapelv:**
  - A belső reprezentációt nem akarjuk a hívó kód felé megmutatni – **adatrejtés**
  - pontosan szabályozni akarjuk, mely tagokhoz férhetnek hozzá kívülről. Az osztály belső reprezentációját így bármikor módosíthatjuk, csupán a kívülről való elérés módját kell változatlan formában meghagynunk – **egységbezárás; újrafelhasználhatóság**

public	Korlátlan
protected	Csak az adott osztályban és az utódosztályokban érhető el (később)
private	Csak az adott osztályban érhető el
internal	Csak az adott assemblyn belül lehet használni (később)
protected internal	Csak az adott assemblyn belül, vagy utódokban (később)

- **A tagok alapértelmezett láthatósága private**

Gyakorló feladat

Egészítse ki a Háromszög osztályt egy olyan konstruktorral, amely véletlenszerű (0-100 közötti) oldalhosszúsággal rendelkező háromszöget generál. A keletkező háromszögnek szerkeszthetőnek kell lennie!

```
public Haromszog()
{
    Random rand = new Random();
    do
    {
        a = rand.Next(0, 101);
        b = rand.Next(0, 101);
        c = rand.Next(0, 101);
    }
    while (!SzerkeszthetoEzAHaromszog());
}
...
private bool SzerkeszthetoEzAHaromszog()
{
    return (a + b > c) && (a + c > b) && (b + c > a);
}
```

Tulajdonságok

- **Eddig public-ká tettük az adattagokat is, hogy hivatkozni tudjunk rájuk kívülről**

```
class ComplexNumber
{
    ...
    public double real;
    public double imaginary;

    public ComplexNumber(double r, double i)
    {
        real = r;
        imaginary = i;
    }
    ...
}
```

- **Problémák:**
  - Public láthatóságú adat írható és olvasható is
    - Esetleg nem szeretnénk, hogy írni lehessen
  - Az értékadás kontrollálatlan
  - Nem akarjuk kifelé megmutatni a reprezentációt – még azt sem, hogy az adatok milyen formában vannak tárolva
    - Lehet, hogy egy adat tárolási módja összetett
    - Lehet, hogy egy adott adat nincs is tárolva, hanem on-the-fly kiszámoljuk?

Tulajdonságok

- **Kontrollált hozzáférést biztosítanak speciális formában megadott *getter* és *setter* metódusok segítségével**

```
class ComplexNumber
{
    ...
    private double real;

    public double Real
    {
        get { return real; }
        set { real = value; }
    }
    ...
}
```

  - Kívülről az elérés hasonló a változókéhoz
  - Valójában azonban a getter függvény fut le, ha lekérik a „változó” értékét
  - A setter fut le, ha beállítják
    - A value speciális paraméter ebben a függvényben, az épp hozzárendelt értéket jelzi

```
ComplexNumber c = new ComplexNumber(20, 2);
c.real = 10;
```

1 'OOP_eleje.ComplexNumber.real' is inaccessible due to its protection level

```
c.Real = 10;                    //JÓ
Console.WriteLine(c.Real); //JÓ
```

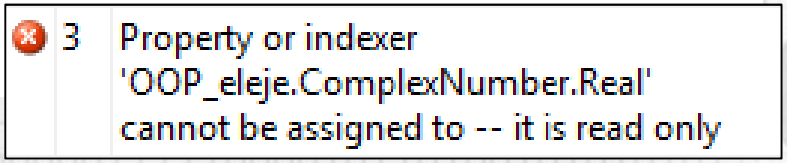
Tulajdonságok

- **Akár a getter, akár a setter elhagyható**

```
private double real;
public double Real
{
    get { return real; }
}
```

Kívülről csak olvasható reális rész! – Az osztályon belül azonban elérjük a real változót közvetlenül is

```
c.real = 10;                    //NEM JÓ
c.Real = 10;                    //NEM JÓ
Console.WriteLine(c.Real); //JÓ
```



- **Akár komplex kód is írható a getter és setter törzsébe**
  - Általában ellenjavallt!

```
public double Real
{
    get { return real; }
    set
    {
        Trace.WriteLine("real: " + real + " changed to " + value +
            " from " + new StackTrace().GetFrames()[0].GetMethod().Name);
        real = value;
    }
}
```

Tulajdonságok

- Példa on-the-fly kiszámított tulajdonságra

```
private double imaginary;
private double real;
...
//z = a + bi komplex szám felírható z = r*(cosφ + i*sinφ) alakban
...
public double Radius
{
    get { return Math.Sqrt(Math.Pow(real, 2) + Math.Pow(imaginary, 2)); }
}

public double Angle
{
    get { return Math.Atan2(imaginary, real); }
}
```

```
ComplexNumber z = new ComplexNumber(10, 0);
Console.WriteLine("z trigonometrikus alakja: " + z.Radius +
                  "*(cos(" + z.Angle + ") + i*sin(" + z.Angle + "))");
Console.ReadLine();
```

```
z trigonometrikus alakja: 10*(cos(0) + i*sin(0))
```

Osztályok alkotóelemei III.

```
class ComplexNumber
{
    private double real;
    private double imaginary;

    public double Real
    {
        get { return real; }
        set { real = value;}
    }

    public double Imaginary
    {
        get { return imaginary; }
        set { imaginary = value;}
    }

    ...

    ...
}
```

Adattagok

Tulajdonságok

Metódusok (függvények)

Konstruktorok

Gyakorló feladat

Egészítse ki a Háromszög osztályt úgy, hogy módosíthatóak legyenek az oldalhosszak, de csak úgy, hogy a háromszög szerkeszthető maradjon! (Rossz oldalhossz megadása esetén maradjon meg az eredeti hossz.)

```
private double a;
public double A
{
    get { return a; }
    set
    {
        double regiA = a;
        a = value;
        if (!SzerkeszthetoEzAHaromszog()) //Ebben a megoldásban egyszerűen
csak felhasználtuk ezt a régebben meglévő függvényt.
            a = regiA;
    }
}

...
```

Gyakorló feladatok

Készítsen Kör osztályt, amely egy kört a középpontja koordinátáival és a sugárral reprezentál! Az osztályban legyen metódus, ami megállapítja, hogy egy adott pont benne van-e a körben vagy sem.

Készítsen példányt a tesztelésre: kérje be a felhasználótól egy kör adatait, majd kérjen be pontokat, és mondja meg, ezek benne vannak-e a körben vagy sem.

Készítsen a Körhöz egy új konstruktort, amely megadott sugárral, de véletlenszerű középponttal hozza létre a példányt.

Egészítse ki a Kör osztályt úgy, hogy legyen metódusa, amely megállapítja, hogy egy adott pont a kör középpontjától pontosan mekkora távolságra van.

Módosítsa a Kör osztályt Céltábla osztállyá úgy, hogy legyen benne metódus, amely egy adott pontnak a középponttól való távolsága szerint különböző pontszámokat ad! - Ezután készítsen céllövő játékot: hozzon létre egy véletlenszerű középpontú céltáblát, majd kérjen be a felhasználótól 15 lövést (pontot), és számolja, hány pontnyi találata van a felhasználónak a 15 lövés után. Minden lövés után segítségül közölje, mekkora volt a lövés távolsága a céltáblától.