PROGRAMOZÁS II
2. ÓRA: OOP

OOP: OBJEKTUM ORIENTÁLT PROGRAMOZÁS

OOP alapfogalmai

- Objektum
  - Összefogja az egybe tartozó adatokat az őket kezelő kódokkal
  - Tagváltozó vagy mező: az objektum által tárolt adatok
  - Metódus vagy tagfüggvény: az objektum által tárolt kódok
- Osztály
  - Definiál egy újra használható mintát objektumok létrehozásához
- A kódban
  - Az osztály egy típus
  - Az objektum egy változó
  - Az objektum az osztály egy példánya (az objektum létrehozása a példányosítás)

OOP alapelvei

- Egységbe zárás
  - Az adatok és a függvények egybe fogása
  - Az adatokhoz csak a megengedett módon, jellemzően a függvényeken keresztül, lehet hozzáférni
- Származtatás (öröklés)
  - Az osztály adatainak és függvényeinek újra használása
  - Specializált osztályokat készít
- Polimorfizmus (többalakúság)
  - Ugyanaz az objektum több típusként is használható

OOP alapelvei: példák

- Egységbe zárás
  - Egy eltárolt képnek nem írhatom csak úgy át a felbontását a tartalom módosítása nélkül
  - Erre van az átméretezés függvény
- Származtatás (öröklés)
  - Ha képfájlokat akarok kezelni, akkor nagyon sok funkció megegyezik általában a fájlok funkcióival
  - A képet nem kezdem elölről, hanem egy speciális fájlként tekintek rá
- Polimorfizmus (többalakúság)
  - Bizonyos műveletekhez teljesen mindegy, hogy képfájlt kezelek, hangfájlt vagy szöveget
  - Bármelyik kezelhető egyszerűen fájlként

OSZTÁLY LÉTREHOZÁSA

Osztály: adattagok

- A tagváltozóknak meg kell adni az elérhetőséget:
  - private: csak az osztály éri el (alapértelmezett)
  - public: bármi eléri
  - protected: csak az osztály és a gyerekosztályok érik el

Ezek a fontosabbak

```
class Esemeny
{
    public string _nev;
    protected int _vendegekSzama;
    private int _jegyAr;
}
```

```
class Esemeny
{
    string _nev;
    int _vendegekSzama;
    int _jegyAr;
}
```

Osztály: metódusok

- A metódust ugyanúgy az osztályon belülre írjuk
- Ezeknek is van elérhetőségük

```
class Esemeny
{
    string _nev = "pelda";
    int _vendegekSzama = 40;
    int _jegyAr = 1500;

    public int OsszesBevetel()
    {
        return _vendegekSzama * _jegyAr;
    }

    public void UjVendegekFeljegyzese(int ujVendegekSzama)
    {
        _vendegekSzama += ujVendegekSzama;
    }
}
```

Osztály: metódus meghívása

- A metódust olyan szintaxissal hívjuk meg, ahogy az adattagokat is elértük eddig
  - Ezek is az objektum részei

```
static void Main(string[] args)
{
    Esemeny peldaEsemeny = new Esemeny();

    Console.WriteLine($"Osszes bevetel: {peldaEsemeny.OsszesBevetel()}");
    //Osszes bevetel: 60000

    peldaEsemeny.UjVendegekFeljegyzese(15);

    Console.WriteLine($"Osszes bevetel: {peldaEsemeny.OsszesBevetel()}");
    //Osszes bevetel: 82500
}
```

Hogy lesz a metódus az objektum része: this

- Az osztály metódusain belül a this kulcsszóval tudunk az aktuális objektumra hivatkozni, amin belül a metódust meghívtuk
  - Ha nincsenek ütköző nevek, akkor nem kötelező

```
class Esemeny
{
    string _nev = "pelda";
    int _vendegekSzama = 40;
    int _jegyAr = 1500;

    public int OsszesBevetel()
    {
        return this._vendegekSzama * this._jegyAr;
    }
    public void UjVendegekFeljegyzese(int ujVendegekSzama)
    {
        this._vendegekSzama += ujVendegekSzama;
    }
}
```

Konstruktor

- Speciális metódus, feladata az adattagok értékeinek megfelelő beállítása, amikor egy objektum létrejön
- C#-ban a neve az osztály nevével egyezik meg, visszatérési típusa nincs

```
class Esemeny
{
    string _nev;
    int _vendegekSzama;
    int _jegyAr;

    public Esemeny(string nev, int jegyAr)
    {
        _nev = nev;
        _jegyAr = jegyAr;
        _vendegekSzama = 0;
    }
}
```

Amikor létrehozunk egy eseményt, adjuk meg a nevét, és a jegyek árát

A vendégek száma mindig 0-ról indul

Konstruktor

- Objektum létrehozásakor a konstruktornak át kell adni a paramétereket

```
static void Main(string[] args)
{
    Esemeny peldaEsemeny1 = new Esemeny("Focimeccs", 25000);
    peldaEsemeny1.UjVendegekFeljegyzese(150);
    Console.WriteLine($"Osszes bevetel: {peldaEsemeny1.OsszesBevetel()}");
    //Osszes bevetel: 3750000

    Esemeny peldaEsemeny2 = new Esemeny("Koncert", 65000);
    peldaEsemeny2.UjVendegekFeljegyzese(35);
    peldaEsemeny2.UjVendegekFeljegyzese(50);
    Console.WriteLine($"Osszes bevetel: {peldaEsemeny2.OsszesBevetel()}");
    //Osszes bevetel: 5525000
}
```

A nevet és a jegy árat megadjuk, a vendégek száma 0

Konstruktor

- A paraméterek tekintetében ugyanaz igaz rá, mint bármely más függvényre
  - Függvénytúlterhelés, opcionális paraméterek
- Ha nem készítünk saját konstruktort, akkor is létezik egy
  - Alapértelmezett konstruktor
  - Nem vár paramétert
  - Minden adattagot az alapértelmezett értékével hoz létre
    - Szám esetén 0
    - Objektum esetén null referencia

Példa: kutyasétáltatás

- Rozi sétáltatja Blöki-t
- Blöki sétál

- Szereplők és viselkedések
  - Rozi (ember)
    - Sétáltatja Blökit (kutya)
  - Blöki (kutya)
    - Sétál

Példa: kutyasétáltatás (Imperatív)

```
class Kutya
{
    public string nev;
    public bool setalE;
}
```

```
class Ember
{
    public string nev;
}
```

```
static void Main(string[] args)
{
    Kutya bloki = new Kutya();
    bloki.nev = "Blöki";
    bloki.setalE = false;
    Ember rozi = new Ember();
    rozi.nev = "Rozi";

    Console.WriteLine($"{rozi.nev} sétáltatja {bloki.nev}-t");
    bloki.setalE = true;
    Console.WriteLine($"{bloki.nev} sétál");
    //Rozi sétáltatja Blöki-t
    //Blöki sétál
}
```

Mennyi helyen lehet itt hibázni?

Példa: kutyasétáltatás (Strukturált)

```
class Kutya
{
    public string nev;
    public bool setalE;
}
```

```
class Ember
{
    public string nev;
}
```

```
public static void KutyaSetal(Kutya kutya)
{
    kutya.setalE = true;
    Console.WriteLine($"{kutya.nev} sétál");
}
public static void EmberKutyatSetaltat(Ember ember, Kutya kutya)
{
    Console.WriteLine($"{ember.nev} sétáltatja {kutya.nev}-t");
    KutyaSetal(kutya);
}
```

Példa: kutyasétáltatás (Strukturált)

```
class Kutya
{
    public string nev;
    public bool setalE;
}
```

```
class Ember
{
    public string nev;
}
```

```
static void Main(string[] args)
{
    Kutya bloki = new Kutya();
    bloki.nev = "Blöki";
    bloki.setalE = false;
    Ember rozi = new Ember();
    rozi.nev = "Rozi";

    EmberKutyatSetaltat(rozi, bloki);
    //Rozi sétáltatja Blöki-t
    //Blöki sétál
}
```

Mennyi helyen lehet itt hibázni?

Példa: kutyasétáltatás (Objektum orientált)

- Átalakítás
  - Publikus adattagok → privát adattagok
  - Függvények → metódusok
  - Változók → Objektumok
- Egységbe zárás elve
  - Adattagok privátok
  - Csak tagfüggvényekkel érhetőek el és módosíthatóak
- Adatok megfelelő inicializálása
  - Konstruktor

Példa: kutyasétáltatás (Objektum orientált)

- Ember: osztály
  - Tudása (tagváltozó):
    - Neve
  - Felelőssége (metódus):
    - Kutyát sétáltat
- Kutya: osztály
  - Tudása (tagváltozó):
    - Neve
    - Sétál-e
  - Felelőssége (metódus):
    - Sétál

Példa: kutyasétáltatás (Objektum orientált)

```
class Kutya
{
    string _nev;
    bool _setalE;

    public Kutya(string nev)
    {
        _nev = nev;
        _setalE = false;
    }
    public void Setal()
    {
        _setalE = true;
        Console.WriteLine($"{_nev} sétál");
    }
    public string NevLekeres()
    {
        return _nev;
    }
}
```

Konstruktor: sosem lesz név nélkül a kutya, és a bool változó értéke mindig false lesz

A névhez nem lehet kívülről hozzáférni, de egy metódus visszaadhatja

Példa: kutyasétáltatás (Objektum orientált)

```
class Ember
{
    string _nev;

    public Ember(string nev)
    {
        _nev = nev;
    }
    public void KutyatSetaltat(Kutya kutya)
    {
        Console.WriteLine($"{_nev} sétáltatja {kutya.NevLekeres()}-t");
        kutya.Setal();
    }
}
```

Konstruktor: sosem lesz név nélkül az ember

A kutya nevét csak lekérni tudjuk a metódussal, átállítani nem

Példa: kutyasétáltatás (Objektum orientált)

```
static void Main(string[] args)
{
    Kutya bloki = new Kutya("Blöki");
    Ember rozi = new Ember("Rozi");

    rozi.KutyatSetaltat(bloki);
    //Rozi sétáltatja Blöki-t
    //Blöki sétál
}
```

Mennyi helyen lehet itt hibázni?

PROPERTY-K

Property

- A property-k olyan elemei az osztályoknak, amelyek a privát adattagokhoz biztosítanak hozzáférést
- Használat során úgy néznek ki, mintha publikus adattagok lennének, de valójában köztes függvények a meghívó kód és a privát adat között
- Lehetővé teszik a privát adat elérésének korlátozását és validálását
- Egy property kezelhet több adattagot is, és egy adattaghoz lehet több property is

Property példa

```
class Ertekeles
{
    string _nev;
    int _ertek; // százalékban

    public Ertekeles(string nev, int ertek)
    {
        _nev = nev;
        _ertek = ertek;
    }

    public string Nev
    {
        get { return _nev; }
    }

    …
}
```

A Nev a property, ami hozzáférést ad a _nev adattaghoz

Csak a lekérdezést engedjük meg

Property példa

```
class Ertekeles
{
    string _nev;
    int _ertek; // százalékban

    …

    public int Ertek
    {
        get { return _ertek; }
        set
        {
            if (value < 0) { _ertek = 0; }
            else if (value > 100) { _ertek = 100; }
            else { _ertek = value; }
        }
    }
}
```

Az Ertek a property, ami hozzáférést ad az _ertek adattaghoz

A lekérdezés értelemszerűen megy

Megengedjük a beállítást is, de korlátozottan
Csak 0 és 100 között lehet az érték

Property példa

```
static void Main(string[] args)
{
    Ertekeles ertekeles1 = new Ertekeles("Dávid", 76);
    Ertekeles ertekeles2 = new Ertekeles("Sára", 82);
    Console.WriteLine($"{ertekeles1.Nev} értékelése: {ertekeles1.Ertek}");
    //Dávid értékelése: 76
    Console.WriteLine($"{ertekeles2.Nev} értékelése: {ertekeles2.Ertek}");
    //Sára értékelése: 82

    ertekeles1.Ertek = 86;
    Console.WriteLine($"{ertekeles1.Nev} értékelése: {ertekeles1.Ertek}");
    //Dávid értékelése: 86
    ertekeles2.Ertek = 120;
    Console.WriteLine($"{ertekeles2.Nev} értékelése: {ertekeles2.Ertek}");
    //Sára értékelése: 100
    ertekeles1.Ertek = -30;
    Console.WriteLine($"{ertekeles1.Nev} értékelése: {ertekeles1.Ertek}");
    //Dávid értékelése: 0
}
```

Property példa

```
static void Main(string[] args)
{
    Ertekeles ertekeles1 = new Ertekeles("Dávid", -3);
    Ertekeles ertekeles2 = new Ertekeles("Sára", 120);
}
```

Mi van, ha itt adunk meg rossz értéket?
A konstruktor nem ellenőrizte ...

Property példa

```
static void Main(string[] args)
{
    Ertekeles ertekeles1 = new Ertekeles("Dávid", -3);
    Ertekeles ertekeles2 = new Ertekeles("Sára", 120);
}
```

Mi van, ha itt adunk meg rossz értéket?
A konstruktor nem ellenőrizte …

```
class Ertekeles
{
    string _nev;
    int _ertek; // százalékban

    public Ertekeles(string nev, int ertek)
    {
        _nev = nev;
        Ertek = ertek;
    }

    …
}
```

De át lehet írni, hogy ez a property-n keresztül végezze a beállítást

Automatikusan implementált property

- Ha a property-t csak a triviális módon használjuk (egyszerű beállítás és lekérés), akkor van egyszerűbb szintaxis is

- Ilyen esetben a privát adattagot nem kötelező kézzel létrehoznunk
  - A fordító a háttérben generálja magának, de a programozó csak a property-t kezeli

Automatikusan implementált property

```
class Ertekeles
{
    int _ertek; // százalékban

    public string Nev { get; }

    public Ertekeles(string nev, int ertek)
    {
        Nev = nev;
        Ertek = ertek;
    }

    …
}
```

A Nev egy automatikus property
Jelen esetben csak olvasható
Beállításra egyedül a konstruktor képes

Különböző verziók

```
public string Nev { get; }
```

Ez esetben csak olvasható, de az publikusan
A konstruktor tudja csak beállítani

```
public string Nev { get; set; }
```

Ez esetben publikusan lekérhető és beállítható

```
public string Nev { get; private set; }
```

Ez esetben publikusan lekérhető
Beállítani csak a saját metódusok tudják

Vannak még opciók, de ezek a legjellemzőbbek

MÉG EGY PÉLDA

A példa: kártya pakli és laphúzás

- Egy egyszerű kártyajáték első lépéseit valósítjuk meg
- Legyen egy francia kártya pakli
- A játékban valamennyi játékos mindegyike kap valamennyi lapot, majd dönthet, hogy eldob 1-et vagy 2-t
  - A végén lenne egy kiértékelés, de annak a részleteibe most nem megyünk bele
- Hogy épül fel a program?

Szereplők és szerepek

- Kártya
  - Csak adatot tárol
- Pakli
  - Tárolja a még ki nem húzott lapokat
  - Lehetőséget ad a laphúzásra
  - Meg lehet keverni
- Játékos
  - Tárolja saját lapjait
  - Tud lapot húzni
- Játék
  - Lebonyolítja a teljes folyamatot

Pakli

- Tudása (tagváltozó):
  - A pakliban lévő lapok
- Felelőssége (metódus):
  - Inicializálás az összes lappal
  - Keverés
  - Laphúzás a tetejéről
  - Mennyi lap van még benne?

Játékos

- Tudása (tagváltozó):
  - A kezében lévő lapok
- Felelőssége (metódus):
  - Lap húzása a pakliból
  - Lap eldobása kézből
  - Lapok lekérhetőek

Játék

- Tudása (tagváltozó):
  - Játékosok
  - Pakli
- Felelőssége (metódus):
  - Lapot oszt minden játékosnak
  - Felhasználói interfész a játékosi döntésekhez (lapeldobás)
  - Végső kiértékelés

Kártya

```
class Card
{
    public char Suit { get; }
    public string Rank { get; }

    public Card(char suit, string rank)
    {
        Suit = suit;
        Rank = rank;
    }

    public string Value
    {
        get { return Suit + Rank; }
    }
}
```

A színek egyszerűen karakterrel: '♠', '♥', '♦', '♣'

Az értékeknek a „10” miatt szöveg kell

Extra property, hogy egyben elérhessük

Pakli

```
class CardDeck
{
    Card[] _cards;
    public int RemainingCards { get; private set; }

    public CardDeck(bool shuffle = true)
    {}

    public void ShuffleDeck()
    {}

    public Card DrawTopCard()
    {}

    public bool IsEmpty()
    {}
}
```

A tömbhöz semmiképp sem lehet kívülről hozzáférni

Játékos

```
class Player
{
    Card[] _cardsInHand;
    public int CardCount { get; private set; }

    public Player(int maxCardCount)
    {}

    public void DrawCardFromDeck(CardDeck deck)
    {}

    public Card GetCard(int index)
    {}

    public void DiscardCardByIndex(int index)
    {}
}
```

A tömbhöz semmiképp sem lehet kívülről hozzáférni

Az egyes kártyák lekérhetőek

Játék

```
class GameManager
{
    Player[] _players;
    CardDeck _deck;
    int _numberOfCardsToDeal;
    int _maxCardsToDiscard;

    public GameManager(int playerCount, int numberOfCardsToDeal, int
maxCardsToDiscard)
    {}
    public void PlayGame()
    {}
    private void PlayerInteraction(Player player)
    {}
    private bool ReadIndexAndDiscard(Player player)
    {}
    private void DetermineAndAnnounceResult()
    {}
}
```

A játék logikához vannak segéd metódusok, de mind privát