Szoftvertervezés és -fejlesztés II. labor

1. rész

Programozási tételek

Öröklődés

Interfészek

Eseménykezelés

Kivételkezelés

Visszalépéses keresés

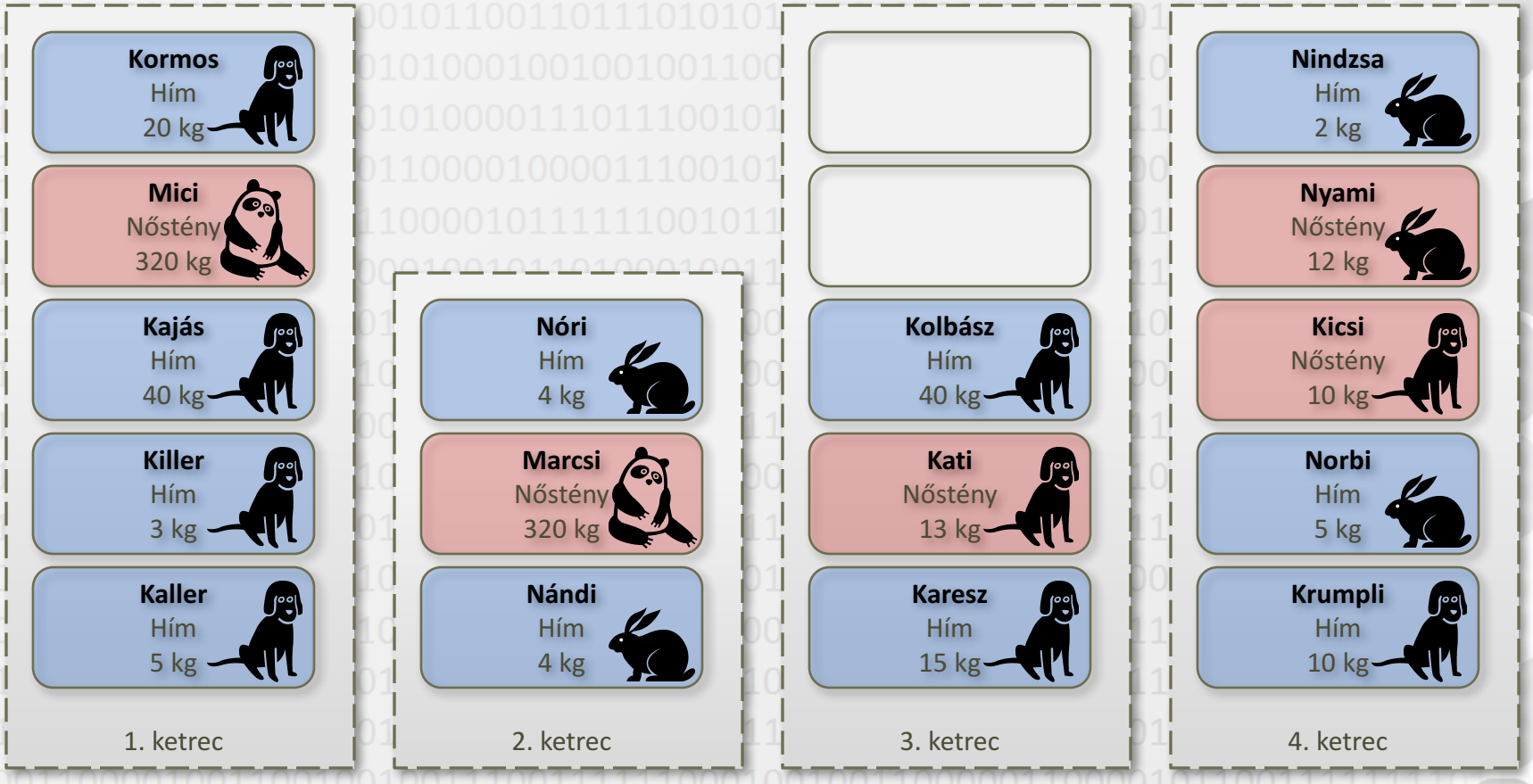
Szoftvertervezés és -fejlesztés II.

- Programozási tételek
- **OOP alapok ismétlése**
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

Programozási tételek összeépítése

- **Feladat - mintaadatok**

  Állatkertünkben háromféle állat tárolunk különböző méretű ketrecekben az alábbi mintaadatok szerint:



szenasi.sandor@nik.uni-obuda.hu

Programozási tételek összeépítése

- **Készítsünk egy *Állat* nevű osztályt, ami**
  - Tárolja az alábbi adatokat:
    - *név* – szöveg
    - *nem* – logikai érték (*igaz* ha hím, *hamis* ha nőstény)
    - *súly* – egész szám
    - *faj* – felsorolás típus (kutya, panda, nyúl)
  - A mezők értékét a konstruktorban lehessen beállítani és tulajdonságokkal lehessen lekérdezni
- **Készítsünk egy *Ketrec* osztályt, ami állatokat tárol**
  - Legyen egy belső tömbje, ami az állatokat tárolja. Ennek a méretét a konstruktorban lehessen megadni
  - Legyenek ilyen metódusai:
    - *Felvétel*, ami paraméterként kap egy állatot, és ezt elhelyezi a tömbbe (ha elfér)
    - *Töröl,* ami törli a paraméterként megadott nevű állatot
- **Készítsünk egy főprogramot**
  - Itt legyen egy 4 *Ketrec* objektumot tartalmazó tömb, és azt töltsük fel a megadott állatokkal

szenasi.sandor@nik.uni-obuda.hu

Programozási tételek összeépítése

- **Egyszerű programozási tételek (N → 1)**
  - Megadott ketrecben hány darab megadott fajú állat található?
    int FajDarab(AllatFaj faj)
  - Megadott ketrecben van-e megadott fajú és nemű állat?
    bool FajEsNemVanE(AllatFaj faj, bool himnemu)
- **Egyszerű programozási tételek (N → N)**
  - Megadott ketrecben melyek a megadott fajú állatok?
    Allat[ ] FajAllatok(AllatFaj faj)
- **Programozási tételek összeépítése**
  - Megadott ketrecben mennyi a megadott fajú állatok átlagos tömege?
    float AtlagFajTomeg(AllatFaj faj)
  - Megadott ketrecben van-e legalább egy azonos fajú, de ellenkező nemű egyedekből álló páros?
    bool AzonosFajEllenkezoNemVanE()
  - Melyik ketrecben található a legtöbb megadott fajú állat?
    Ketrec LegtobbFaj(Ketrec[ ] A, AllatFaj faj)

szenasi.sandor@nik.uni-obuda.hu

Szoftvertervezés és -fejlesztés II. labor

1. rész

Programozási tételek

Öröklődés

Interfészek

Eseménykezelés

Kivételkezelés

Visszalépéses keresés

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - **Öröklődés a C# nyelvben**
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

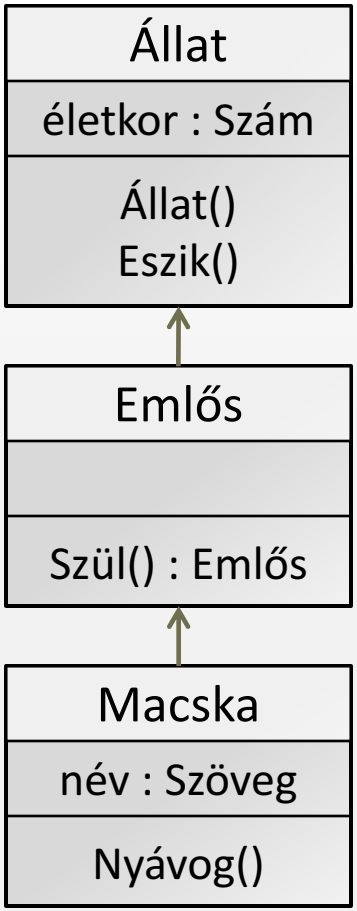
Öröklődés a C# nyelvben

- **Leszármazott osztályok deklarációjánál „:” karakterrel elválasztva lehet megadni az ősosztály nevét (csak egyszeres öröklődés van)**
- **Ezt követően csak az új mezőket/metódusokat kell felsorolni**

```
class Állat
{
    int életkor;
    public Állat( ) { ... }
    public void Eszik( ) { ... }
}

class Emlős : Állat
{
    public Emlős Szül( ) { ... }
}

class Macska : Emlős
{
    string név;
    public void Nyávog( ) { ... }
}
```



Konstruktorok öröklődése

- **A konstruktorok nem öröklődnek**
  - Van lehetőség meghívni az ősosztály konstruktorát a „base” kulcsszó segítségével (több ős konstruktor esetén a paraméterlista alapján dönt)
- **Kötelező konstruktorhívás**
  - A leszármazottban kötelező meghívni az ős valamelyik konstruktorát
  - Amennyiben nincs ilyen hívás, akkor az ős paraméter nélküli konstruktora automatikusan meghívódik
    (ha az ősnek nincs paraméter nélküli konstruktora, a fordító hibát jelez)

```
class A
{
    public A( ) { ... }
}
```

```
class B : A
{
    public B( ) { ... }
}

class C : A
 {
    public C(int x) { ... }
}
```

```
class E : C
{
    public E(int y) : base (y) { ... }
}

class F : C
{
    public F( ) : base (5) { ... }
}
```

Virtuális és nemvirtuális metódusok

- **Nemvirtuális metódusok**
  - Korai kötés jellemzi őket
  - Alapértelmezetten minden metódus nemvirtuális
- **Virtuális metódusok**
  - Késői kötés jellemzi őket
  - Külön szintaktikai megjelölést igényelnek
    - A virtuális metódusokat az ősosztályban a „virtual” kulcsszóval kell megjelölni
    - A leszármazottakban a felülbírált virtuális metódusokat az „override” kulcsszóval kell megjelölni (egyébként metódus elrejtés történik, lásd következő dia)
    - Az ős metódus elérhető a „base” kulcsszó segítségével

```
class A
{
    public virtual int Metodus( ) { ... }
}

class B : A
{
    public override int Metodus( ) { ... }
}
```

Metódusok elrejtése

- **Elrejtés: leszármazott osztályban azonos néven létrehozunk egy másik metódust**
  - A leszármazott osztályban az új metódust a „new” kulcsszóval célszerű megjelölni (bár nem kötelező, a fordító figyelmeztet ha elmarad)
  - Az ősosztály azonos nevű metódusa elérhető a „base” kulcsszó segítségével
- **Elrejtés - virtualitás**
  - Mind virtuális, mind pedig nemvirtuális metódusok esetében használható (virtuális metódus esetén egy új virtuális hívási láncot indít)
  - Az elrejtésnek nincs köze a virtualitáshoz, valójában egymástól független metódusokat jelent, akiknek „véletlenül” azonos a nevük

```
class A
{
    public int MetodusX( ) { ... }
    public virtual int MetodusY( ) { ... }
}
```

```
class B : A
{
    public new int MetodusX( ) { ... }
    public new virtual int MetodusY( ) { ... }
}
```

Típuskényszerítés („casting”)

- **Típuskényszerítéssel egy objektumot úgy kezelhetünk, mintha egy másik típusú lenne**
  - Implicit: automatikus típusátalakítás
    - Pl. számok közötti automatikus konverzió (egész → lebegőpontos), nincs szükség jelölésre
  - Explicit: átalakítás a programozó kérésére
    - Jelölése: az átalakítandó típus elé zárójelbe írjuk a kívánt típust
    - Pl. Állat x; Macska y = (Macska)x; ((Macska)x).Nyávog();
- **„is” operátor**
  - Használata: „x is Állat”
  - Igaz értékkel tér vissza, ha az ellenőrizendő objektum a megadott osztályhoz, vagy annak valamely leszármazottjához tartozik
- **„as” operátor**
  - Használata: „x as Állat”
  - Ha az átalakítás sikerül, a kifejezés használható a megadott típusúként, egyébként a kifejezés értéke null lesz

szenasi.sandor@nik.uni-obuda.hu

Absztrakt osztály és metódus

- **Az absztrakt metódusokat és osztályokat az „abstract” kulcsszóval kell megjelölni**
  - Egy osztály kötelezően absztrakt, ha legalább egy absztrakt metódusa van
  - Absztrakt osztályból nem lehet példányosítani
  - Absztrakt metódusokat a leszármazottban kötelező implementálni (vagy absztraktként jelölni)

```
abstract class Síkidom
{
    public abstract double Terület( );
    public abstract double Kerület( );
}

class Téglalap : Síkidom
{
    int a;
    int b;
    public override double Terület( ) { return a * b; }
    public override double Kerület( ) { return 2 * (a + b); }
}
```

Lezárt osztály és metódus

- **A lezárt metódusokat és osztályokat a „sealed” kulcsszóval kell megjelölni**
- **Megjelölhető vele egyetlen metódus vagy egy teljes osztály is**
  - Osztály esetén nem engedi a származtatást
  - Metódus esetén nem engedi a felülírást

```
sealed class Téglalap : Síkidom
{
    int a;
    int b;
    public override double Terület( )
    {
        return a * b;
    }
    public override double Kerület( )
    {
        return 2 * (a * b);
    }
}
```

Object ősosztály

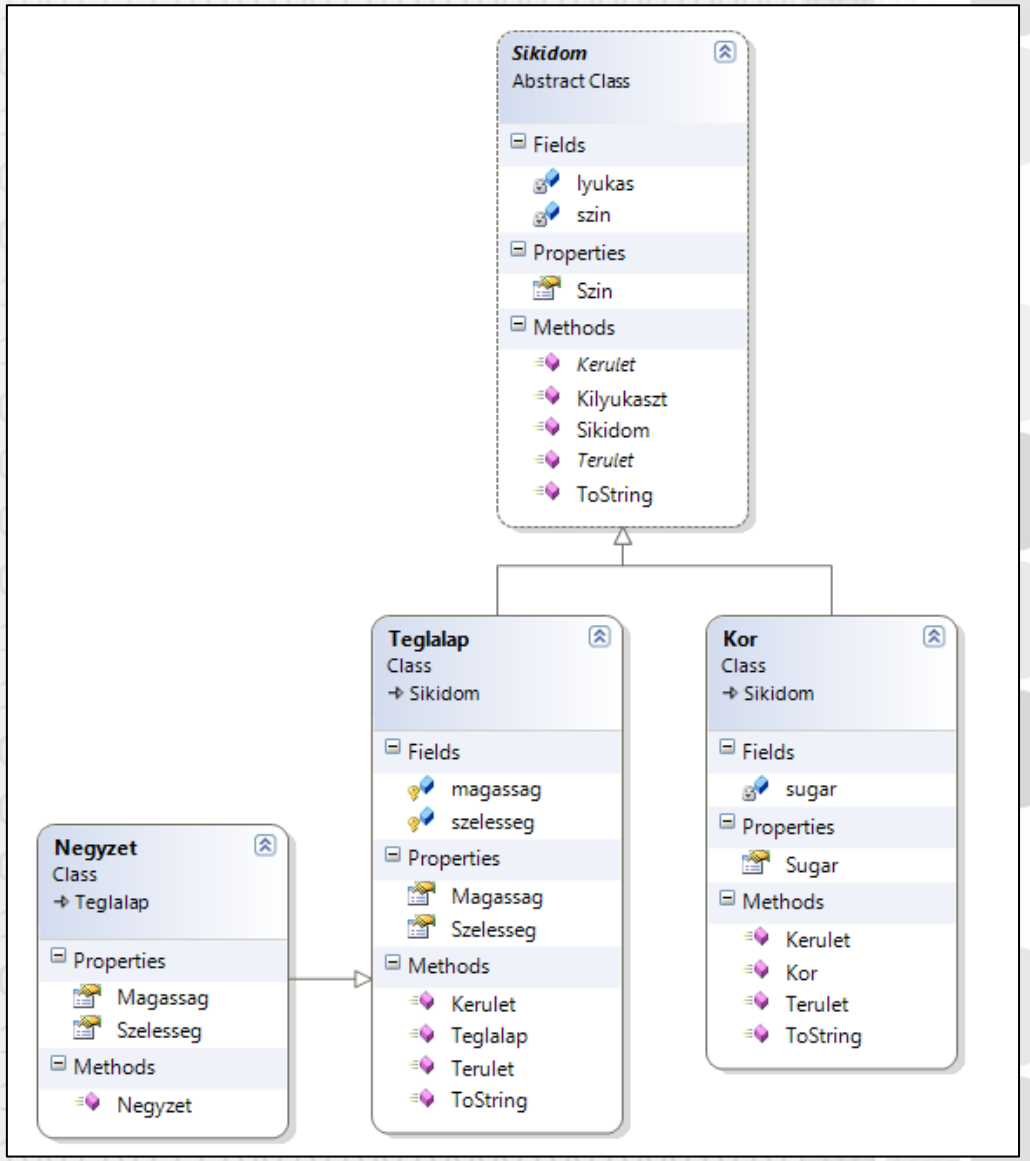
- **Minden osztály közös őse a „System.Object” osztály**
- **Amennyiben külön nem adunk meg egy osztálynak őst, akkor az automatikusan az Object leszármazottja lesz**
- **Néhány fontosabb metódusa:**
  - **public Type GetType()**
    Visszaadja a példány típusát reprezentáló objektumot
  - **public virtual bool Equals(object obj)**
    Egyenlőség vizsgálat, saját osztály esetén célszerű felülírni
  - **public virtual int GetHashCode()**
    Visszaad egy hash értéket, saját osztály esetén célszerű felülírni
  - **public virtual string ToString()**
    Tetszőleges szöveget ad vissza, a gyakorlatban gyakran jól használható
  - **public static bool ReferenceEquals(object objA, object objB)**
    Statikus metódus a referencia szerinti egyenlőségvizsgálathoz
  - **public static bool Equals(object objA, object objB)**
    Statikus metódus a tartalom szerinti egyenlőségvizsgálathoz

Szoftvertervezés és -fejlesztés II. labor

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - **Öröklődés feladatok**
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

Öröklődés példa

- **Feladat**
  Valósítsuk meg az oldalt látható osztályhierarchiát
- **Oldjuk meg az alábbiakat:**
  - Tároljunk el 5 db síkidomot egy tömbben
  - Készítsünk egy metódust, ami egy síkidomot kilyukaszt, ha annak nagyobb a területe mint a kerülete
  - Készítsünk egy metódust, ami megadott oldalhosszak alapján létrehoz egy *Téglalap* vagy egy *Négyzet* objektumot
  - Készítsünk egy metódust, ami *Síkidom*ok tömbjéből megadja a legnagyobb területű elemet



szenasi.sandor@nik.uni-obuda.hu

Szoftvertervezés és -fejlesztés II. labor

1. rész

Programozási tételek
Öröklődés
Interfészek
Eseménykezelés
Kivételkezelés
Visszalépéses keresés

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - **Interfész a C# nyelvben**
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

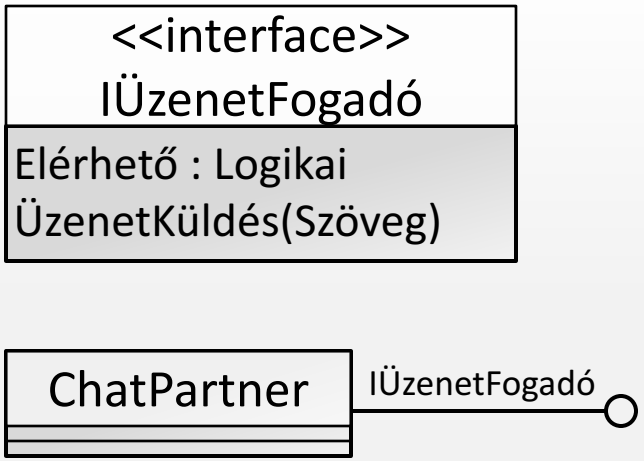
Interfész C# nyelvben

- **Interfész létrehozása az „interface” kulcsszóval lehetséges**
  - Fel kell sorolni a kötelezővé teendő metódusokat, tulajdonságokat
- **Az interfész implementálása az ősosztály megadásához hasonló**
  - Ha meg van adva ősosztály, akkor vesző után következik az interfész
  - Egyszerre több interfész is megvalósítható
- **Interfész típus nem példányosítható, de referenciaváltozó lehet**

```
interface IÜzenetFogadó
{
    bool Elérhető { get; set; }
    void ÜzenetKüldés(string üzenet);
}

class ChatPartner : Személy, IÜzenetFogadó
{
    public bool Elérhető { get {...} set {...} }
    public void ÜzenetKüldés(string üzenet) { ... }
}

IÜzenetFogadó x = new ChatPartner();
```



Absztrakt osztály és interfész

- **Absztrakt osztály is megvalósíthat interfészt**
  - Ilyenkor a metódust nem szükséges implementálni, azonban kötelezően absztraktként kell megjelölni
  - Ugyanez igaz az interfészben szereplő tulajdonságokra

```
interface IEladható
{
    bool Ár { get; set; }
    void Elad( );
}

abstract class Termék : IEladható
{
    public abstract bool Ár { get; set; }
    public abstract void Elad( );
}
```

- **Az absztrakt osztály leszármazottainak implementálniuk kell a metódusokat és tulajdonságokat**

szenasi.sandor@nik.uni-obuda.hu

Interfészek kiterjesztése

- **Interfészek kiterjesztése formailag hasonló az osztályok származtatásához („:” jel után kell megadni az ősöket)**
  - Egy interfész egyszerre több másik interfészt is kiterjeszthet
  - Egy interfészt megvalósító osztálynak implementálnia kell az interfész ősei által előírt követelményeket is (metódusok, tulajdonságok, események stb.)

```
interface IEladható
{
    bool Ár { get; set; }
    void Elad( );
}
interface IAkciózható : IEladható
{
    void Akció(double kedvezmény);
}
public class Termék : IAkciózható
{
    public bool Ár { get; set; }
    public void Elad( ) { ... }
    public void Akció(double kedvezmény) { ... }
}
```

Implicit interfész megvalósítás

- **Implicit interfész megvalósítás**
  - Ha több megvalósított interfész is tartalmaz ugyanolyan metódus szignatúrát (+maga a megvalósító osztály is tartalmazhat ilyen metódust), akkor azok egy metódussal is megvalósíthatók
  - Ebben az esetben mindegy, hogy melyik referenciával hivatkozunk az objektumra, mindig ez a metódus fog lefutni

```
interface IFileKezelő
{
    void Töröl( );
}
interface IKorrektúra
{
    void Töröl( );
}
public class SzövegFile : IFileKezelő, IKorrektúra
{
    public void Töröl( ) { … }
}
```

Explicit interfész megvalósítás

- **Explicit interfész megvalósítás**
  - Ha több megvalósított interfész is tartalmaz ugyanolyan metódus szignatúrát (+ maga a megvalósító osztály is tartalmazhat ilyen metódust), akkor azok különböző metódussal is megvalósíthatók
  - Ebben az esetben a hívást végző referencia típusától függ, hogy melyik metódus fut le

```
interface IFileKezelő
{
    void Töröl( );
}
interface IKorrektúra
{
    void Töröl( );
}
public class SzövegFile : IFileKezelő, IKorrektúra
{
    public void Töröl( ) { … }
    void IFileKezelő.Töröl( ) { … }
    void IKorrektúra.Töröl( ) { … }
}
```

Szoftvertervezés és -fejlesztés II.

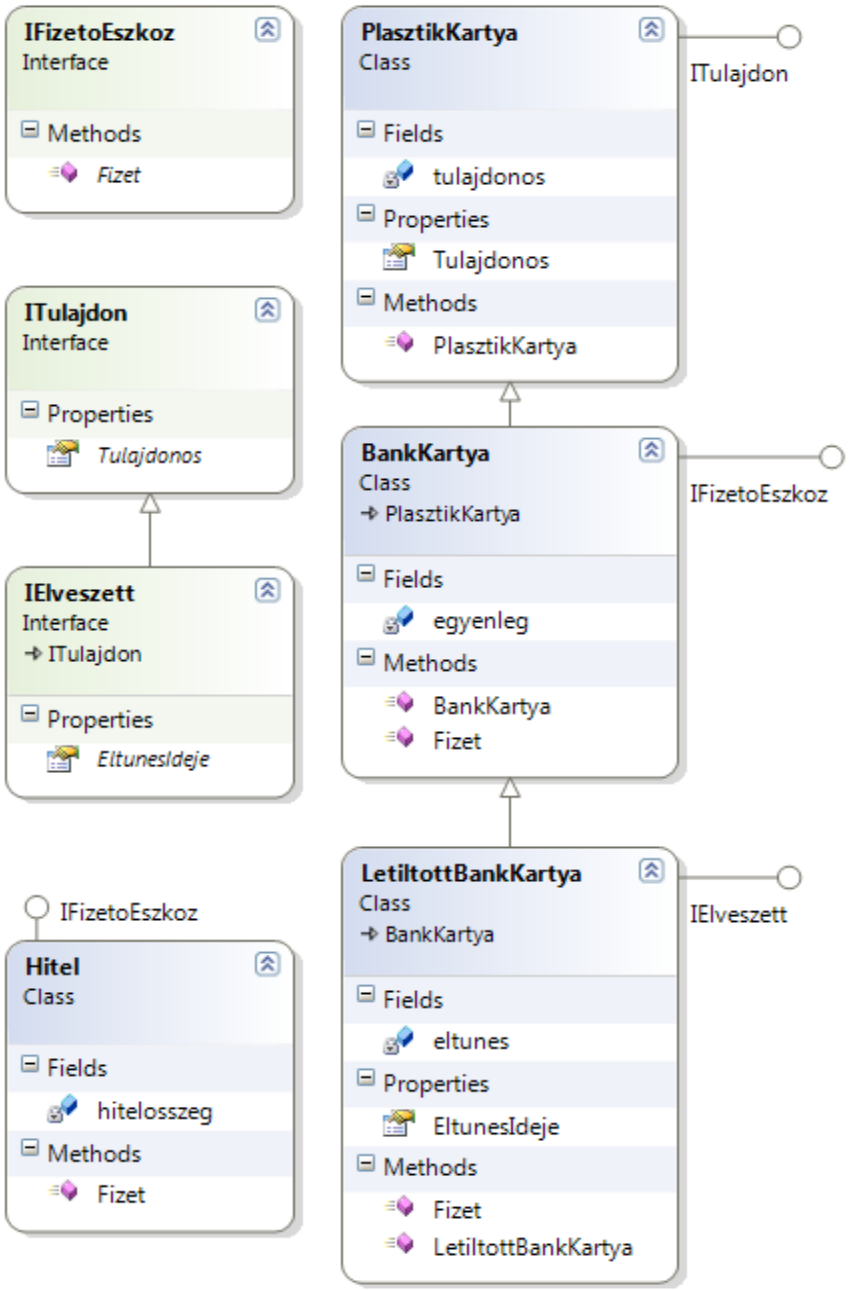
- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - **Interfész feladatok**
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

Meglévő interfész megvalósítása

- **Feladat**
  Készítsen egy tetszőleges osztályt, amely megvalósítja az IComparable interfészt
  - A .NET osztálykönyvtár tartalmaz egy *IComparable* nevű interfészt, amelyet megvalósítva egy objektum össze tudja hasonlítani önmagát egy másikkal
  - Az interfész csak egyetlen metódust határoz meg:
    int CompareTo(Object obj)
    A leírás alapján ennek lehetséges visszatérési értékei:
    - kisebb mint 0 – a példány megelőzi a paraméterként átadottat a sorrendben
    - 0 – a példány és a paraméterként átadott azonos helyen szerepelnek a sorrendben
    - nagyobb mint 0 – a példány követi a paraméterként átadottat a sorrendben
- **Próbálja ki az így megvalósított objektumokat, egy belőlük létrehozott tömböt adjon át a beépített rendező metódusnak**
  - A .NET osztálykönyvtár rendelkezik egy *Array.Sort(Array)* statikus metódussal
  - Ez a metódus rendezi a paraméterként átadott *IComparable* interfészt megvalósító objektumokat

Saját interfészek

- **Feladat**
  Valósítsuk meg az ábrán látható interfész és osztály hierarchiát
- **Készítsünk főprogramot ami egy közös tömbben tárol IFizetoEszkoz interfészt meg-valósító objektumokat**
- **Legyen egy fizetés metódusa, ami egy ilyen tömböt kap paraméter-ként, és akkor ad vissza igazat, ha bármelyik elem végre tudta hajtani a fizetést**
- **Legyen egy ellenőrzött fizetés metódus, ami hasonló, de a fizető nevét is ellenőrzi**



Szoftvertervezés és -fejlesztés II. labor

1. rész

Programozási tételek
Öröklődés
Interfészek
Eseménykezelés
Kivételkezelés
Visszalépéses keresés

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - **Eseménykezelés interfészekkel**
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

Eseménykezelés interfészekkel

- **Feladat - Készítsük el a felsorolt osztályokat és interfészeket**
- **IHívásFigyelő interfész az alábbi követelményekkel**
  - *BejövőhívásTörtént(Telefon küldő, String forrás_telefonszám)*
  - *KimenőhívásTörtént(Telefon küldő, String cél_telefonszám)*
- **Telefon oszály**
  - Legyen egy *telefonszám* nevű mezője, amit a konstruktorban lehet beállítani
  - Legyen egy *egyenleg* nevű mezője, amit az *EgyenlegFeltöltés(int összeg)* metódus hívásával lehet növelni
  - Legyen egy *hívásfigyelő* nevű, *IHívásFigyelő* típusú mezője, ami a regisztrált eseménykezelőt tárolja. Egy *FigyelőRegisztrál(IHívásFigyelő figyelő)* metódussal tudjon egy objektum regisztrálni az eseményekre
  - Legyen egy *HívásFogadás(Telefon forrás)* metódusa, amely meghívja az eseménykezelő objektum (ha az létezik) *BejövőHívásTörtént* metódusát
  - Legyen egy *HívásKezdeményezés(Telefon cél)* metódus, amely meghívja az eseménykezelő *KimenőHívásTörtént* metódusát, és ha az egyenleg engedi, meghívja a cél *HívásFogadás* metódusát, majd csökkenti az egyenleget

Eseménykezelő megvalósítása

- **HívásNapló oszály**
  - Valósítsa meg az *IHívásFigyelő* interfészt
  - Mindkét szükséges metódus írja ki a képernyőre a paraméterként kapott adatokat
- **A fenti osztályok és interfészek implementálását követően hozzon létre minta objektumokat, majd próbálja ki a fenti funkciók működését**
  - Hozzon létre legalább két *Telefon* objektumot
  - Hozzon létre egy *HívásNapló* objektumot
  - Regisztrálja a *HívásNaplót* a két telefonnál eseménykezelőként
  - Töltse fel valamelyik *Telefon* egyenlegét
  - Indítson néhány hívást a telefonokról

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - **Eseménykezelés delegáltakkal**
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

A képviselő („delegate”)

- **A képviselő segítségével egy osztályhoz külső osztályok kapcsolódhatnak**
  - Más néven „delegált” vagy „metódusreferencia”
  - Tetszőleges osztály kapcsolódhat a képviselőn keresztül egy másik osztályhoz, ennek feltétele az előírt formátum betartása
  - Ez a formátum az üzenetváltáshoz hívandó metódus szignatúrája
- **A képviselő segítségével egy osztály metódusai külső osztályok metódusait ismeretlenül is meg tudják hívni**
  - Üzenetküldési lehetőség külső osztályok részére
  - Visszahívási, értesítési funkció
- **A képviselők a delegate kulcsszó segítségével saját névvel és az általuk képviselendő metódusok szignatúrájával deklarálhatók**

```
public delegate void EventHandler(object sender, EventArgs e);
```

  - Neve: EventHandler
  - Visszatérési érték típusa: void
  - Paraméterek típusai: object és EventArgs

szenasi.sandor@nik.uni-obuda.hu

Metódushívás képviselőkön keresztül

- **Egyszerű metódushívás lépései**
  - Példányosítani kell a képviselő osztályt, ehhez a konstruktorának át kell adni a későbbiekben hívandó metódust (megfelelő szignatúrával)
  - Ezt követően a létrejött képviselő objektumon keresztül meg lehet hívni az előzőleg a konstruktornak átadott metódust

```
delegate double Közvetítő(double szám);

static class Műveletek
{
    public static double Kétszerezés(double szám) { return szám + szám; }
}

class Program
{
    static void Main()
    {
        Közvetítő teszt = new Közvetítő(Műveletek.Kétszerezés);
        System.Console.WriteLine(teszt(5));
    }
}
```


szenasi.sandor@nik.uni-obuda.hu

Több metódus felfűzése (multicast)

- **Egy képviselő egyszerre több metódust is „tartalmazhat”**
  - A += operátor segítségével lehet hozzáadni új metódust
  - A -= operátor segítségével lehet elvenni egy metódust

```
delegate void SzövegFeldolgozó(string szöveg);

static class Műveletek
{
    public static void KiírNagy(string s) { System.Console.WriteLine(s.ToUpper()); }
    public static void KiírKicsi(string s) { System.Console.WriteLine(s.ToLower()); }
}

class Program
{
    static void Main()
    {
        SzövegFeldolgozó teszt = new SzövegFeldolgozó (Műveletek.KiírNagy);
        teszt += new SzövegFeldolgozó (Műveletek.KiírKicsi);
        teszt(”Teszt Üzenet”);
    }
}
```


szenasi.sandor@nik.uni-obuda.hu

Néhány tipikus használati eset

- **Képviselő változóként**
  - Az adatokhoz hasonlóan eltárolhat valamilyen funkciót
    - Bármikor meghívható
    - Bármikor megváltoztatható
  - Tömb esetén funkciók egy teljes listáját tárolhatja
- **Képviselő paraméterként**
  - Paraméterként várhat valamilyen funkciót
    - Rendezés esetén a rendezési szempontot (< operátor megvalósítása)
    - Visszahívási lehetőség (a függvény kap egy funkciót és azt hívja meg bizonyos feltételek teljesülése esetén)
- **Képviselő visszatérési értékként**
  - Visszatérési értékként visszaadhat valamilyen funkciót
    - A hívónak nem is kell ismernie a rendelkezésre álló funkciót körét, a paraméterek alapján a metódus dönthet, hogy melyiket célszerű használni
- **Eseménykezelés megvalósítására**

Az esemény („event”)

- **Az esemény egy kifejezetten értesítési célú nyelvi elem**
  - Valójában egy korlátozott, biztonságosabbá tett képviselő
- **Az eseménykezelés általános menete**
  - Szükséges képviselő típusok létrehozása
  - Az osztály közzéteszi az eseményeit
  - Az esemény iránt érdeklődő osztályok saját metódusaik átadásával feliratkoznak az eseményre
    - Értelemszerűen csak megfelelő szignatúrájú metódussal
    - Feliratkozás a +=, leiratkozás a -= operátorral történik
  - Amikor a közzétevő osztályban kiváltódik az esemény, a képviselők segítségével értesíti a feliratkozott osztályokat
- **Esemény közzététele**
  - Az események speciális mezők, amelyeket az event kulcsszó jelöl, rendelkeznek saját névvel, illetve egy képviselő által megadott típussal

```
public event EventHandler Esemény;
```


szenasi.sandor@nik.uni-obuda.hu

Eseménykezelés mintaprogram

```
public delegate void SzamValtozott(int szamErtek);

class Szamolo {
    public event SzamValtozott figyelo;

    int szam;
    public int Szam {
        set {
            szam = value;
            if (figyelo != null) figyelo(szam);
        }
    }
}

class Program {
    static void Figyelo(int szam) { Console.WriteLine("Változás:" + szam); }

    static void Main(string[ ] args) {
        Szamolo teszt = new Szamolo( );
        teszt.figyelo += Figyelo;
        teszt.Szam = 10;
    }
}
```

Szoftvertervezés és -fejlesztés II. labor

1. rész

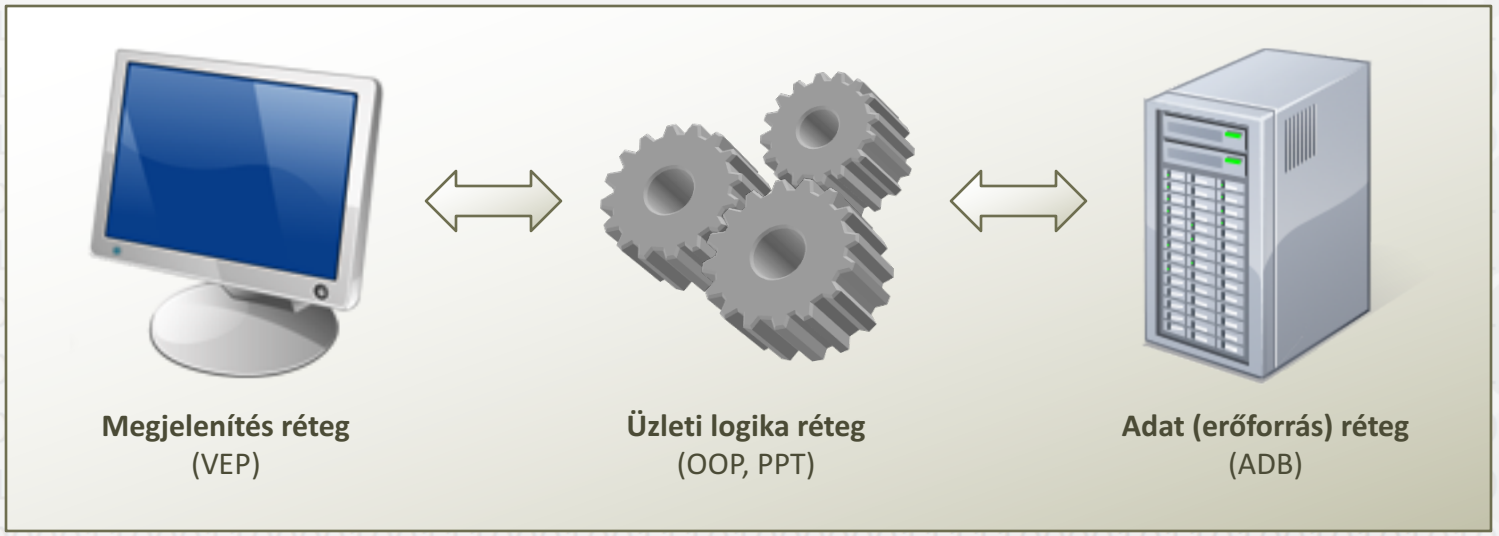
Programozási tételek
Öröklődés
Interfészek
Eseménykezelés
Kivételkezelés
Visszalépéses keresés

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - **Saját kivételek készítése**
- Rekurzió
  - Egyszerű rekurzív feladatok
  - Visszalépéses keresés

Többrétegű architektúrák

- **Háromrétegű architektúra**



- **Jellemzői**
  - Minden réteg egy jól definiálható feladatot lát el, közvetlenül az alatta lévő réteget felhasználva
  - Ha nincs is szükség az egyes rétegek fizikai függetlenségére (pl. külön alkalmazásszerver), akkor is célszerű laza kapcsolatot építeni a szintek között
  - Előnyei:
    - Kódújrafelhasználás, modularitás, team-munka egyszerűsödik, kód áttekinthetőbb
    - Egyszerűbb karbantartás, jó skálázhatóság

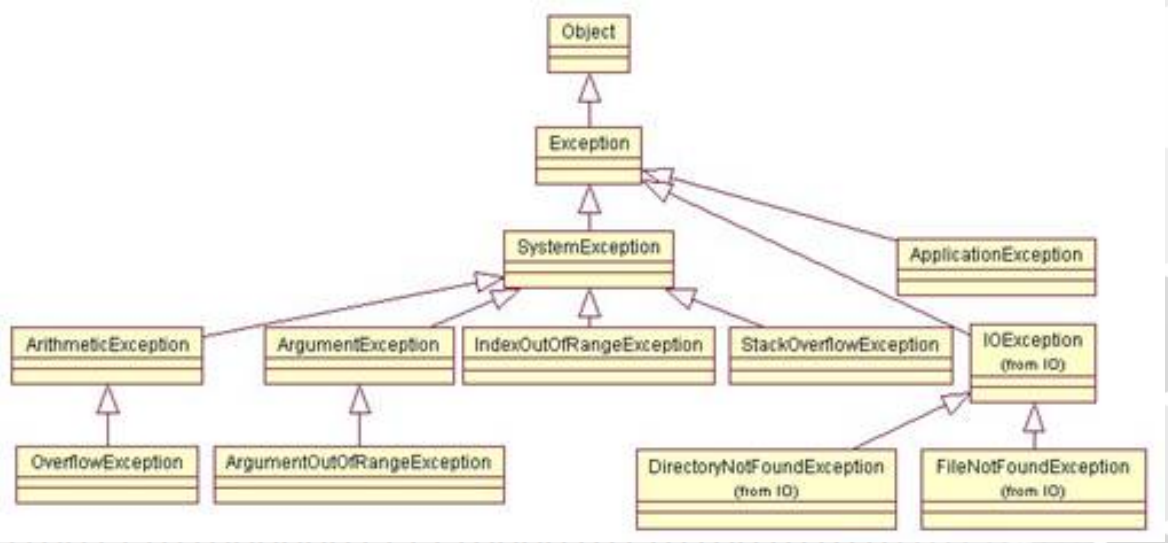
Kivételek elkapása

- **Példa kivételek kezelésére**

```
try
{
  FileStream sr = File.OpenRead(@"c:\temp\hibas.bin");
  BinaryReader br = new BinaryReader(sr);
  br.ReadDecimal();
  sr.Close();
}
catch (FileNotFoundException e)
{
  Console.WriteLine("Nincs ilyen file:" +e.FileName);
}
catch (EndOfStreamException e)
{
  Console.WriteLine("Nincs benne adat");
}
finally
{
  Console.WriteLine("Mindig lefut");
}
```

Saját kivétel

- **Saját kivétel osztály létrehozása**
  - Exception leszármazottja
  - ApplicationException
  - Lehet saját mező, tulajdonság, metódus, stb.



- **Kivétel eldobása**
  - Kivétel osztályból egy példány létrehozása
  - Ennek eldobása

```
throw new SajatKivetel(param1, param2);
```

- **Kivétel elkapása**
  - Megegyezik a beépített kivételeknél láthatóval
  - A kivétel típus utáni változó segítségével magára az előzőleg létrehozott kivétel objektumra hivatkozhatunk

Saját kivételek készítése

- **Feladat: készítsünk egy Verem osztályt, ami kivételekkel jelzi a nem megfelelő hívásokat**
- **Verem osztály**
  - Egy belső tömbben tárolja az elemeket (egész számokat tároljon)
  - Legyen egy konstruktora, ahol paraméterként meg lehet adni az adatszerkezet méretét
    public Verem(int meret)
  - Egy metódus segítségével lehessen új számot berakni a verembe. Amennyiben a fix méretű tömbbe már nem fér el az elem, dobjon egy **VeremMegteltKivetel** típusú kivételt (ami tartalmazzon egy referenciát a veremre, illetve magát a sikertelenül beszúrt elemet is)
    public void Push(int elem)
  - Legyen lehetőség kivenni az utolsóként berakott elemeket a veremből. Ha üres veremnek hívják meg ezt a metódusát, dobjon **VeremUresKivetel** típusú kivételt
    public int Pop()

Szoftvertervezés és -fejlesztés II. labor

1. rész

Programozási tételek
Öröklődés
Interfészek
Eseménykezelés
Kivételkezelés
Visszalépéses keresés

Szoftvertervezés és -fejlesztés II.

- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - **Egyszerű rekurzív feladatok**
  - Visszalépéses keresés

Programozási tételek rekurzív formája

- **Feladat**

  Valósítsuk meg rekurzívan (ciklus használata nélkül) az alábbi feladatokat:

- **Programozási tételek**
  - Mennyi egy N elemű A tömb elemeinek az összege?
    int Osszegzes(int[ ] A, int N)
  - Egy N elemű A tömbben van-e 15-el osztható szám?
    bool Eldontes(int[ ] A, int N)
  - Egy N elemű A tömbben hol van a(z egyik) maximális érték?
    int MaximumKivalasztas(int[ ] A, int N)
- **Logaritmikus keresés**
  - Valósítsuk meg a keresés rekurzív változatát az alábbi paraméterekkel:
    - *A – az elemeket tartalmazó tömb*
    - *keresett – a keresett elem értéke*
    - *alsoindex – a még vizsgálandó terület első elemének indexe*
    - *felsoindex – a még vizsgálandó terület utolsó elemének indexe*

  int LogaritmikusKereses(int[ ] A, int keresett, int alsoindex, int felsoindex)

Szoftvertervezés és -fejlesztés II.

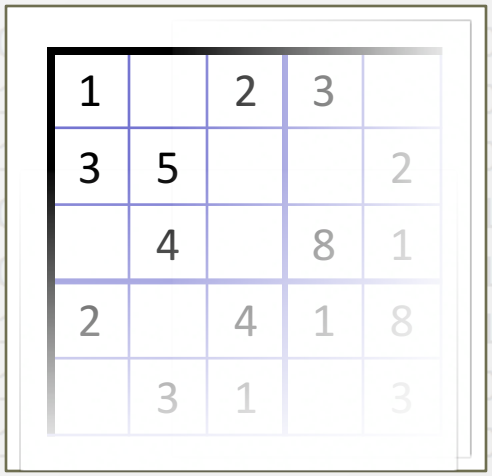
- Programozási tételek
  - OOP alapok ismétlése
- Öröklődés
  - Öröklődés a C# nyelvben
  - Öröklődés feladatok
- Interfészek
  - Interfész a C# nyelvben
  - Interfész feladatok
- Eseménykezelés
  - Eseménykezelés interfészekkel
  - Eseménykezelés delegáltakkal
- Kivételkezelés
  - Saját kivételek készítése
- Rekurzió
  - Egyszerű rekurzív feladatok
  - **Visszalépéses keresés**

Visszalépéses keresés (1)

- **6. Feladat**

  Készítsünk egy visszalépéses keresésen alapuló Sudoku megoldó programot, amely a tábla üres helyeit kitölti az alábbi szabályok szerint:
  - Minden üres helyre egy szám írható 1..9 között
  - Egy sorban, illetve egy oszlopban nem szerepelhet kétszer ugyanaz a szám
  - A teljes tábla 3x3-as blokkokra oszlik, egy blokkon belül nem szerepelhet kétszer ugyanaz a szám (a tábla mérete 9x9 tehát összesen 9 blokkot tartalmaz)

- **Visszalépéses keresés használatához javasolt átalakítások**
  - Kétdimenziós tábla adatainak átalakítása részfeladatok sorozatává
  - Fixen megadott számok és a kitöltendő üres helyek szétválogatása



Fix mezők: (0,0) (0,2) (0,3) (1,0) …

Üres mezők: (0,1) (0,4) (1,2) (1,3) …

Visszalépéses keresés (2)

- **Megvalósítandó függvények**
  - ft(szint, szám) függvény
    Visszatérési értéke igaz, ha az előre fixen beírt számok egyike sem zárja ki, hogy a **szint**edik részeredményhez tartozó mezőbe beírjuk a **szám** értéket
    bool ft(int szint, int szam)
  - fk(szint, szám, k, kszám) függvény
    Visszatérési igaz, ha a **k**. részeredményként választott **kszám** érték nem zárja ki, hogy a **szint**edik részeredményhez tartozó mezőbe beírjuk a **szám** értéket
    bool fk(int szint, int szam, int k, int kszam)
  - BackTrack(szint, címszerint VAN, E)
    Az előadáson megismert visszalépéses keresés algoritmus implementációja
    void BackTrack(int szint, ref bool VAN, int[ ] E)
  - Rekurziót indító metódus
    A fenti metódusok célszerűen nem publikusak, ezért készítsünk egy egyszerű, az algoritmus működésének (és bemenő paramétereinek) ismeretét nem feltételező indító metódust
    public bool MegoldasKereses()

Visszalépéses keresés (3)

- **Már megvalósított segéd metódusok**
  - A Pozicio osztály lenti metódusa segítségével egyszerűen eldönthető, hogy a paraméterként átadott két mező kizáró kapcsolatban áll-e egymással (nem tartalmazhatják ugyanazt a számot). Visszatérési értéke csak akkor igaz, ha a két mező egy sorban, egy oszlopban vagy egy blokkban van:
    static bool Kizaroak(Pozicio p1, Pozicio p2)
  - A Sudoku osztály alábbi metódusai elvégzik a tábla szétbontását fix és kitöltendő mezők listájára, illetve az eredmény betöltését az eredeti táblába:
    void MezoSzetvalogatas( )
    void MezoOsszefuzesMegoldassal(int[ ] E)
  - A Program osztály rendelkezik egy statikus metódussal, amely segítségével ki lehet listázni egy tábla tartalmát a képernyőre:
    static void TablaKirajzolas(int[,] tabla)
  - A Program osztály rendelkezik egyéb statikus metódusokkal, amelyek visszaadnak egy-egy Sudoku feladvány táblát (a 0. mintatábla csak tesztelési célokat szolgál, ugyanis nem a játék szabályainak megfelelő méretű):
    static int[,] MintaTablaFeltoltes_?( )