HALADÓ PROGRAMOZÁS

Előadások/Gyakorlatok:	
Eseménykezelés alapjai, gyakorlati megvalósítása	2+2
Delegáltak készítése, a .NET beépített delegáltjainak használata	2+2
Fájlok és könyvtárak kezelése	2+2
XML és JSON fájlok létrehozása, feldolgozása	2+2
Nyelvbe ágyazott lekérdezések (LINQ) alapjai	2+2
LINQ lekérdezések objektumgyűjteményeken, XML és JSON fájlokon	2+2
DLL-ek készítése, felhasználása	2+2
Attribútum alapú programozás, reflexió alapjai	2+2
Adatbázis elérési lehetőségek Adatbázisok felhasználása	2+2
Adatbázisok készítése DbFirst módszerrel	2+2
Adatbázisok készítése CodeFirst módszerrel	2+2
ZH	2+2
Féléves feladat bemutatása	2+2
Féléves feladat pót bemutatása	2+2

Tematika

- 1 db elméleti ZH és a gyakorlat teljesítése

Könyv

Troelsen, Andrew, and Phil Japikse. *Pro C# 8 with. NET Core 3: Foundational Principles and Practices in Programming*. Apress, 2020.

Richter, Jeffrey. *CLR via C*. Pearson Education, 2012.

Bevezető (Miért tanuljuk ezt?)

A .Net platform és a C# nyelv eredete 2002-ben indult és azóta egy fontos eszköze a modern szoftverfejlesztésnek

Nagyszámú prgramnyelvet lehet vele használni(C#, VB, F#)

2016-ban indul a .NET Core – összekötni az MS és Linux világot

2021 C#10 és .NET 6 (külön verzió és egymástól függetlenül telepíthető)

Felülről kompatibilitás

.NET jellemzői

Programnyelvek támogatása

Közös futtatókörnyezet

Nyelvi integráció

Osztálykönyvtárak

Egyszerű telepítési modell

Parancssori támogatás - CLI

:NET és :NET Framework (keretrendszer)

- A .NET-keretrendszerben található alaposztály-könyvtárak száma messze meghaladja a .NET-ben található könyvtárakat. Ez érthető is, hiszen a .NET-keretrendszer 14 éves előnyt jelent a .NET-en.
- Ez az eltérés problémákat okozott, amikor a .NET-keretrendszer-kódot .NET-kóddal próbálták használni.
- A .NET Framework/.NET Core 3.1 együttműködés megoldása (és követelménye) a .NET Standard
- A .NET Standard egy olyan specifikáció, amely meghatározza a .NET API-k és alaposztályú könyvtárak elérhetőségét, amelyeknek minden megvalósításban elérhetőnek kell lenniük.

C# nyelv jellemzői

- Nincs szükség mutatókra! A C# programoknak jellemzően nincs szükségük közvetlen mutatómanipulációra (bár erre a szintre szabadon leereszkedhet, ha feltétlenül szükséges).
- Automatikus memóriakezelés a szemétgyűjtésen keresztül.
- Formális szintaktikai konstrukciók osztályokhoz, interfészekhez, struktúrákhoz, felsorolásokhoz és delegáltakhoz.
- A C++-szerű képesség az operátorok túlterhelésére.
- Attribútum alapú programozás támogatása. Ez a fejlesztési márka lehetővé teszi a típusok és tagjaik megjegyzéseit, hogy tovább minősítse viselkedésüket. Például, ha megjelöl egy metódust az [Obsolete] attribútummal, a programozók látni fogják az egyéni figyelmeztető üzenetet, ha megpróbálják használni a díszített tagot.

CIL (NSIL, IL)

```
//Calc.cs
Calc c = new Calc();
int ans = c.Add(10, 84);
Console.WriteLine("10 + 84 is {0}.", ans);
//Wait for user to press the Enter key
Console.ReadLine();

// The C# calculator.
class Calc
{
  public int Add(int addend1, int addend2)
  {
    return addend1 + addend2;
  }
}
```

```
.method public hidebysig instance int32 Add(int32 addend1,
            int32 addend2) cil managed
{
  // Method begins at RVA 0x2090
  // Code size       9 (0x9)
  .maxstack  2
  .locals /*11000002*/ init (int32 V_0)
  IL_0000:  /* 00   |                  */ nop
  IL_0001:  /* 03   |                  */ ldarg.1
  IL_0002:  /* 04   |                  */ ldarg.2
  IL_0003:  /* 58   |                  */ add
  IL_0004:  /* 0A   |                  */ stloc.0
  IL_0005:  /* 2B   | 00               */ br.s       IL_0007
  IL_0007:  /* 06   |                  */ ldloc.0
  IL_0008:  /* 2A   |                  */ ret
} // end of method Calc::Add
```

Delegáltak, események

- Ismétlés –delegált fogalma
- Használata
- Esemény – fogalma
- Használata

Delegate

- Olyan típus, aminek változóiban metódusokat tudunk elhelyezni
  - A **delegált típusa** meghatározza, hogy milyen szignatúrájú függvényt tehetünk bele
  - A konkrét **delegált**ban tárolhatjuk el a függvényeket
  - **A delegáltnak null értéke van, amíg nincs benne függvény**

```
delegate double MyDelegate(int       , string
```

```
double funct(int elso, string masodik)
{
    return elso + masodik.Length;
}
```

```
MyDelegate del = new MyDelegate(funct); //hosszú megadás
MyDelegate del = funct;                 //rövid megadás
```

Delegate használata

- A C#-os delegáltak multicast tulajdonságúak, több függvény is lehet bennük – függvény hozzáadása, eltávolítása:

```
del += new MyDelegate(Function1);  //hosszú megadás
del += Function1;                  //rövid megadás
```

```
del -= new MyDelegate(Function1);  //hosszú megadás
del -= Function1;                  //rövid megadás
```

- Delegáltban lévő függvények hívása:
  - **A hívási sorrend a keretrendszer által nincs garantálva, ne építsünk rá!** (.NET 4.5-ös állapottól: abban a sorrendben hívja, amiben beleraktuk)
  - Visszatérési érték használata esetén az utoljára hívódó metódus visszatérési értékét kapjuk meg (szinte sohasem használjuk így a fenti miatt, inkább: GetInvocationList())

```
MyDelegate temp = del;  //Az ideiglenes változó
if (temp != null)       //szálbiztonság (thread safety)
    temp(0, "alma");    //miatt kell.
```

```
delegate double MyDelegate(int, string)
```

Saját vs. beépített delegált típus

Saját delegált megadás

- Egyeztetni kell a delegált és az átadott függvény típusát és visszatérési értékét
- Pl. „Olyan függvényt képes fogadni, aminek double visszatérési értéke van, és egy int és egy string paramétere”
- Covariancia: a visszatérési érték típusa nem pontosan egyeztetett, de adatvesztés nélkül konvertálható
- Contravariancia: A paraméter típusa nem pontosan egyeztetett, de adatvesztés nélkül konvertálható

Szinte soha nincs rá szükség, a keretrendszerben rengeteg a beépített delegált típus, használjuk inkább ezeket!

Beépített delegált típusok

Neve	**Visszatérési típus, paraméter**	**példa**
Predicate<T>	bool(T)	List<T>.Find(), .Exists(), RemoveAll()…
Comparison<T>	int(T1,T2)	List<T>.Sort(), Array.Sort()
MethodInvoker	void()	
EventHandler	void(object,EventArgs)	
EventHandler<T>	void(object,T) (T EventArgs utód)	
Action	**void()**	
Action<T>	**void(T)**	
Action<T1,T2>	**void(T1,T2)**	
Action<T1,T2,…,T16>	**void(T1,T2,…,T16)**	
Func<TRes>	**TRes()**	
Func<TRes, T>	**TRes(T)**	
Func<TRes, T1, T2>	**TRes(T1,T2)**	
Func<TRes, T1, T2, … T16>	**TRes(T1,T2,…,T16)**	

Delegate használata a gyakorlatban

- Rengeteg keretrendszeri példa!

```
private bool ParosE(int i)
{
    return i % 2 == 0;
}
private int ParosakatElore(int i1, int i2)
{
    bool i1Paros = ParosE(i1);
    bool i2Paros = ParosE(i2);
    if (i1Paros && !i2Paros) return -1;
       else if (!i1Paros && i2Paros) return 1;
       else return 0;
}
```

```
int[] tomb; List<int> lista;
// ...
int elsoParos = lista.Find(ParosE);
List<int> osszesParos = lista.FindAll(ParosE);
bool vanParos = lista.Exists(ParosE);
Array.Sort(tomb, ParosakatElore);
```

Array.Sort-szerű példa

```
delegate bool Osszehasonlito(object bal, object jobb);
class EgyszeruCseresRendezo
{    public static void Rendez(object[] tomb, Osszehasonlito nagyobb)
    {
        for (int i = 0; i < tomb.Length; i++)
            for (int j = i + 1; j < tomb.Length; j++)
                if (nagyobb(tomb[j], tomb[i]))
                {
                    object ideiglenes = tomb[i];
                    tomb[i] = tomb[j];
                    tomb[j] = ideiglenes; }
    }
}
```

Array.Sort-szerű példa – példaosztály

```
class Diak
{
    public string Nev { get; set; }
    public int Kreditek { get; set; }
    public Diak(string nev, int kreditek)
    {
        this.Nev = nev; this.Kreditek =
 kreditek;
    }
}
```

```
Diak[] csoport = new Diak[] {
    new Diak("Első Egon", 52),
    new Diak("Második Miksa", 97),
    new Diak("Harmadik Huba", 10),
    new Diak("Negyedik Néró", 89),
    new Diak("Ötödik Ödön", 69)
};
```

Array.Sort-szerű példa – használat

```
bool KreditSzerint(object elso, object masodik)
{
    return ((elso as diak).Kreditek <
              (masodik as diak).Kreditek);
}

Osszehasonlito del =
  new Osszehasonlito(KreditSzerint);
EgyszeruCseresRendezo.Rendez(csoport, del);
VAGY
EgyszeruCseresRendezo.Rendez(csoport,
  KreditSzerint);
```

Feladat: P01_Delegate

Készítsünk programot, amely:

1. deklarál egy void visszatérési értékű, string paraméterű delegate-et
2. megvalósít egy olyan metódust, melynek szignatúrája megfelel a delegate-nek és kiírja a Console-ra a paraméter értékét
3. meghívja a metódust a delegate-en keresztül
4. megvalósít egy másik metódust, melynek szignatúrája nem felel meg
5. próbáljuk meg meghívni a delegate-en keresztül

Névtelen függvények

- Delegáltak fő használati módjai:
  - Események
  - Metódussal való paraméterezés
- Probléma: az egyszer használatos függvények „elszennyezik” a kódot (pl: IntelliSense-t is)
- Megoldás: névtelen függvények

Anonymous functions
- anonymous methods ☺
- lambda expressions

Anonim függvények

```
int elsoParos = lista.Find(delegate(int i) { return i % 2 == 0; });
List<int> osszesParos = lista.FindAll(delegate(int i) { return i % 2 == 0; });
bool vanParos = lista.Exists(delegate(int i) { return i % 2 == 0; });

Array.Sort(tomb,  delegate(int i1, int i2)
    {
        bool i1Paros = i1 % 2 == 0;
        bool i2Paros = i2 % 2 == 0;
        if (i1Paros && !i2Paros)
            return -1;
        else if (!i1Paros && i2Paros)
            return 1;
        else return 0;
    });
```

- Ma már nem használjuk (-> lambdák)

Feladat

- Készítsünk x,y koordinátával megadott pont osztályt
- Készítsünk és inicializáljunk pontokból álló listát
- Delegált segítségével rendezzük a listát az x, azután az y koordináta szerint, majd egy bekért koordinátájú ponttól való távolság szerint.

Lambda függvények

- Új operátor: => (Lambda operátor)
  - Bemenet és a kimenet összekötésére
- Szintaxis: paraméter[ek] => kimenetet meghatározó kifejezés

- Használata:
  - delegate típus (saját v. keretrendszeri)

```
delegate double SingleParamMathOp(double x);
```

  - delegate változó elkészítése és függvény megadása lambda kifejezés formájában, metódus meghívása

```
SingleParamMathOp muvelet = x => x * x;
double j = muvelet(5);
```

Lambda kifejezések

```
delegate double TwoParamMathOp(double x, double y);
```

```
TwoParamMathOp myFunc = (x, y) => x + y;
double j = myFunc(5, 10);   //j = 15
```

- Beépített delegált típussal:

```
Func<int, int> myFunc = (x) => x * x;
int j = myFunc(5); //j = 25
Func<int, int, int> myFunc2 = (x, y) => x + y;
int j2 = myFunc2(5, 10); //j = 15
```

- Több paramétert zárójelezni kell
- A paraméterek típusozása nem kötelező, csak speciális esetekben

Lambda kifejezések

```
int elsoParos =
    lista.Find(i => i % 2 == 0);
List<int> osszesParos =
    lista.FindAll(i => i % 2 == 0);
bool vanParos =
    lista.Exists(i => i % 2 == 0);
```

```
Array.Sort(tomb,
      (i1, i2) =>
      {
            bool i1Paros = i1 % 2 == 0;
            bool i2Paros = i2 % 2 == 0;
            if (i1Paros && !i2Paros)
                  return -1;
            else if (!i1Paros && i2Paros)
                  return 1;
            else return 0;
      });
```

Lambda kifejezések

- Altípusok:
  - **Kifejezéslambda (Expression Lambda)**
    - Szigorúan egyetlen kifejezés a kimenetet meghatározó oldalon

```
x => x * x
```

  - **Kijelentéslambda (Statement Lambda)**
    - Akár több sorból álló kód a kimenetet meghatározó oldalon, változólétrehozás, .NET függvény hívása, return is megengedett

```
x => { Console.WriteLine(x); }
```

  - **Különbség:**
    - **Az kifejezéslambda adott helyeken (pl. adatbázis felé való kommunikáció) nem delegáltra fordul, hanem kifejezésfára (Expression Tree) – pl. adatbázisszerver tudja az SQL dialektusára való fordítás után végrehajtani**

Névtelen függvények és lambdák

- Előny:
  - A függvény azonnal olvasható a felhasználás helyén
  - Kevesebb „szemét” tagfüggvény az osztályban
- Csak az egyszer használatos, és lehetőleg rövid függvényeket írjuk így meg:
  - A hosszú kód olvashatatlan
  - Mivel „beágyazott kód”, ezért nem újrafelhasználható
- Lehetőleg ne ágyazzunk egymásba névtelen metódusokat

Feladat: P02_DelegateOperations

Készítsünk programot, amely:

1. deklarál egy int visszatérési értékű, string paraméterű delegate-et
2. megvalósít legalább 3 metódust, melyek szignatúrája egyezik a delegate-tel
3. ezen metódusok különböztessék meg magukat a Console-ra való kiíráskor
4. konstruáljunk egy híváslistát, melyet hívások között módosítunk metódusok hozzáadásával és eltávolításával

Feladat: P03_ArrayClass

Készítsünk programot, amely:

1. Tartalmaz egy ArrayClass osztályt, ami egy int[] tömb wrapper-je.
2. Valósítsa meg az index operátort.
3. Legyen egy Length property-je.
4. Legyen egy Transform nevű metódusa, mely egy delegate-et vár paraméterként és mely delegate meghívásával módosítja a tömb összes elemét. Ennek megfelelően deklaráljon egy delegate típust ehhez a művelethez.
5. BÓNUSZ: Legyen az ArrayClass bejárható foreach-csel.

P07_MathOperations

Készítsünk programot, amely:

1. Deklarál egy delegate-et két egész számmal paraméterként.
2. Megvalósítja az egész számokon értelmezett négy alapműveletet elvégző metódusokat és paramétereikkel együtt kiíratja az eredményt a konzolra.
3. Deklarál egy változót a delegate típusával és feltölti a listáját a négy alapművelet metódusaival.
4. Meghívja a delegate-et két tetszőleges számmal.

Összefoglalás

- Megismerkedtünk a .Net fő jellemzőivel
- C# nyelv fő jellemzőivel
- Delegáltakkal
- Névtelen metódusokkal
- Lambda kifejezésekkel