Delegált, generikus delegált, névtelen függvény, lambda kifejezés

Delegate

- Olyan típus, aminek változóiban metódusokat tudunk elhelyezni
  - A **delegált típusa** meghatározza, hogy milyen szignatúrájú függvényt tehetünk bele

```
delegate double MyDelegate(int param1, string param2);
```

  - A konkrét **delegált**ban tárolhatjuk el a függvényeket

```
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
```

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

- **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())

Saját vs. beépített delegált típus

- Saját delegált megadás
  - Egyeztetni kell a delegált és az átadott függvény típusát és visszatérési értékét

```
delegate double MyDelegate(int param1, string param2);
```

  - 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-1; 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: Delegate1

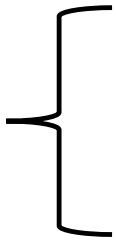
Készítsünk programot, amely:

1. deklarál egy int visszatérési értékű, int paraméterű delegate-et
2. megvalósít két olyan metódust, amelyek szignatúrája megfelel a delegate-nek; nevük Negyzet, ill. Ketszeres legyen. Ezek egy Proba nevű osztályban legyenek, és a Negyzet legyen statikus. A Negyzet négyzetre emeli a kapott számot, a Ketszeres megszorozza 2-vel
3. meghívja a metódusokat egy-egy delegate-en keresztül
4. *Érdemes kipróbálni:* megvalósít egy harmadik metódust, melynek szignatúrája nem felel meg, és
5. próbáljuk meg meghívni 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
- Az angol dokumentáció szerint: (http://msdn.microsoft.com/en-us/library/bb882516.aspx)

Anonymous functions



anonymous methods ☺

lambda expressions

```
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 otthoni gyakorláshoz:

- 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

```
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 otthoni gyakorláshoz:

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: Delegate3

Készítsünk programot, amely:

1. Tartalmaz egy Tomb osztályt, ami egy int[] tömb wrapper-je.
2. Valósítsa meg az index operátort.
3. Legyen egy Hossz tulajdonsága.
4. Legyen paraméteres konstruktora.
5. Legyen egy Atalakitas 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.

Teszteljünk a Main metódusban.

Beadandó házi feladat: Delegate

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.