A C# nyelv újdonságai és jövője

Neuwirth István, 2010
pitta2@gmail.com

C# verziók

- 1.0 – menedzselt kód, a .Net platform saját nyelve
- 2.0 – generikusok bevezetése
- 3.0 – LINQ
- 4.0 – dinamikus programozás
- 5.0 – a fordító, mint szolgáltatás

Tartalom

- Előző előadásban: lexikális elemek, vezérlési szerkezetek, típuskonstrukciók
- Következik:
  - Kivételkezelés
  - yield return
  - Delegate-k
  - Generikusok
  - Attribútumok
  - Unsafe mód
  - Nullable típusok
  - Extension methods
  - LINQ
  - C# 4.0: dynamic, named & optional arguments, variance…

Kivételkezelés

- try – catch – finally
- A catch ágak sorrendje kötött! A speciálisabbtól az általánosabbig, a sorrendet a fordító ellenőrzi.
- Kivételek az Exception osztály leszármazottai.
- Nincs throws záradék, mint például Java-ban.
- Kivétel dobása: throw new MyException(…);

Yield return

- yield return <expression>; vagy yield break;
- Lehet rekurzív is a függvény!
- Késleltetett kiértékelés
- A háttérben a fordító készít egy IEnumerator<T>-t (is) implementáló osztályt

```
public IEnumerator<int> GetEnumerator()
{
  for (int i = 0; i < 10; i++)
  {
    yield return i;
  }
}
```

Delegate-k

- Típusos metódusreferenciák
- Hasonló, mint a C/C++-beli függvénypointerek
- public delegate int Calc(int x, int y);
  - public int add(int x, int y) { return x + y; }
    Calc calc = new Calc(this.add);
    int x = calc(3, 4);
- A delegate is objektum, így paraméterként átadható.

Delegate-k

- Multicast delegate
  - Egy delegate valójában egy időben több függvényre is mutathat.
  - A delegate-re feliratkozó függvények a delegate invoke-álásakor sorban meghívásra kerülnek.

```
delegate void Del(string s);
void leftChar(string s) { Console.WriteLine(s[0]); }
void rightChar(string s) { Console.WriteLine(s[s.length-1]); }

…

Del del = new Del(leftChar);
del += rightChar;
del("Teszt");
// Eredmény: T\nt
```

Delegate-k

- Események (event)
  - Ugyanúgy MulticastDelegate-k
  - public delegate void SomeEventHandler(object o,
    SomeEventArgs e);
    public event SomeEventHandler SomeEvent;

    …

    obj.SomeEvent += new SomeEventHandler(obj_Some);

    …

    void obj_Some(object sender, SomeEventArgs e) { … }

Generikus lehetőségek

- Típusbiztos, dobozolás nélküli collection-ök
- class Osztaly<X, Y> {
  X x;
  Y y;
  public X Method(Y y1, Y y2) { ... }
  }
- Metódus, delegate is lehet generikus...
- Constraint-ek: class Osztaly<T> where T:
  - struct
  - class
  - new()
  - <base class>
  - <interface>

Generikus lehetőségek

- default érték
  - T temp = default(T);
  - T temp = null; nem működne értéktípusoknál
  - T temp = 0; nem működne referenciatípusoknál
- Hasonlít a C++-os template-khez, de nem ugyanaz!
  - Nem adható át érték, csak típus paraméterként
  - Kevésbé rugalmas, mint a C++-nál: nem használhatók például aritmetikai műveletek, csakis olyanok, amelyek megléte következik a megszorításokból
  - …

Attribútumok

- Metainformációk
- Hasonló, mint a Java annotációi
- Osztályokhoz, metódusokhoz, tagokhoz, stb.
  - [Serializable]
  - class Osztaly { … }
- Attribute osztály leszármazottai, konvenció szerint Attribute-ra végződik a nevük, azonban az elhagyható.

Unsafe mód

- unsafe kulcsszóval megjelölhetünk blokkot, metódust, de akár osztályt is!
- C++ módba váltás
- Teljesítménykritikus helyeken
- Pointeraritmetika hasonló, mint C++-ban (++, --, &, void*, stb.)
- Veremre foglalhatunk tömböt
  - char* charPointer = stackalloc char[123];
  - Veszélyes, mert nincs tömbhatárellenőrzés

Unsafe mód

```
static unsafe void Copy(byte[] src, int srcIndex, byte[] dst, int dstIndex, int
count) {
  if (src == null || srcIndex < 0 || dst == null || dstIndex < 0 || count < 0) {
    throw new System.ArgumentException();
  }
  int srcLen = src.Length; int dstLen = dst.Length;
  if (srcLen - srcIndex < count || dstLen - dstIndex < count) {
    throw new System.ArgumentException();
  }

  fixed (byte* pSrc = src, pDst = dst) {
    byte* ps = pSrc; byte* pd = pDst;

    for (int i = 0 ; i < count / 4 ; i++) {
      *((int*)pd) = *((int*)ps);
      pd += 4; ps += 4;
    }

  for (int i = 0; i < count % 4 ; i++) {
    *pd = *ps; pd++; ps++;
  }
}
```

Nullable típusok

- Például adatbázisokban…
- int? a;
  - a = 5; // ok
  - a = null; // ok
  - if (a.HasValue) { int b = a.Value; }
- Valójában a System.Nullable<T> osztályra épül, szintaktikus egyszerűsítés

Nullable típusok

- Mi történik?
  - int? a = 5; int? b = null;
  - a += b; // ?
  - if (a<b) { … } // ?
- Ha az egyik operandus null, a végeredmény null! Illetve logikai kifejezésnél false.
- bool?-ok között & és | műveletek a háromértékű logikának megfelelően, az SQL-lel kompatibilis módon

Nullable típusok

- bool? operator &(bool? x, bool? y)
- bool? operator |(bool? x, bool? y)

x	y	x&y	x|y
true	true	true	true
true	false	false	true
true	null	null	true
false	true	false	true
false	false	false	false
false	null	false	null
null	true	null	true
null	false	false	null
null	null	null	null

Nullable típusok

- ?? null coalescing operator
  - Feltételes operátorral is megoldható (?:), de így egyszerűbb…

```
string str1 = "...";
string str2 = str1 ?? "null!";
// str2 == "..."

string str1 = null;
string str2 = str1 ?? "null!";
// str2 == "null!"
```

Extension methods

- Látszólag osztályok működésének kibővítésére.
- Gyakorlatilag syntax sugar.
- Vigyázni kell velük, mert könnyen ellepnek (legalábbis az IntelliSense listát biztosan)

```
namespace ExtensionMethods {
  public static class MyExtensions {
    public static int WordCount(this String str) {
      return str.Split(new char[] { ' ', '.', '?' },
      StringSplitOptions.RemoveEmptyEntries).Length;
    }
  }
}

...

int i = "Hello Extension Methods".WordCount();
```

Extension methods

- Statikus osztályba statikus függvényként, első paraméter: this <osztály> <név>
- Rengeteg megvalósított extension method: System.Linq névtérbeli Enumerable és Queryable osztályokban (Average, Sum, Take, Where, Union, stb.)

Még néhány 3.0-s apróság…

- Tömörebb kód
  - Auto a = new Auto(); a.Rendszam = "XXX-001"; // helyett
    Auto a = new Auto{ Rendszam = "XXX-001" };
  - List list = new List(); list.Add(0); list.Add(1); // helyett
    List list = new List{ 0, 1 }
  - public string Name{ get; private set; }
  - List<List<KeyValuePair<string, int>>> list = new
    List<List<KeyValuePair<string, int>>>(); // helyett
    var list = new List<List<KeyValuePair<string, int>>>();
  - delegate int del(int i);
    del d = delegate(int x) { return x * x; }; // helyett
    del d = x => x * x;

LINQ

- Language Integrated Query
- Tömör, SQL-szerű forma felsorolható adatszerkezetekkel való műveletvégzésekhez

```
List<int> numbers = new List<int>() { 5, 4, 1, 3, 7, 2, 0 };
IEnumerable<int> orderingQuery =
    from num in numbers
    where num < 3 || num > 7
    orderby num ascending
    select num;

string[] groupingQuery = { "carrots", "cabbage", "broccoli",
"beans", "barley" };
IEnumerable<IGrouping<char, string>> queryFoodGroups =
    from item in groupingQuery
    group item by item[0];
```

LINQ

- Bármilyen, IEnumerable<T>-t megvalósító adatokon
- Lehet az adatbázistábla, tömb, láncolt lista…
- Syntax sugar:

```
string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };

IEnumerable query = from n in names
                    where n.Contains("a")
                    orderby n.Length
                    select n.ToUpper();

IEnumerable query = names.Where<string>(delegate (string n) { return
n.Contains("a"); }).OrderBy<string, int>(delegate (string n) { return
n.Length; }).Select<string, string>(delegate (string n) { return
n.ToUpper(); });

IEnumerable query =
  names.Where(n => n.Contains("a")).OrderBy(n =>
  n.Length).Select(n => n.ToUpper());
```

C# 4.0

- Paramétereknek alapértelmezett érték (lásd C++)
  - public void M(int x, int y = 5, int z = 7);
    M(1, 2, 3); // normál hívás
    M(1, 2); // ⇔ M(1, 2, 7)
    M(1); // ⇔ M(1, 5, 7)
- Mi legyen ha x-et és z-t szeretnénk csak beállítani?
  - M(1,,3) alakban nem hívható
  - Név szerinti paraméter-megfeleltetés
  - Volt ilyen lehetőség Ada-ban is
  - M(1, z: 3); // ⇔ M(x: 1, z: 3) ⇔ M(z: 3, x: 1)

C# 4.0

- Kovariancia és kontravariancia generikusoknál

C# 4.0

- Parallel Extensions
  - Nem nyelvi elemek (még)
  - Parallel LINQ – PLINQ
  - Task Parallel Library – TLP

```
IEnumerable<T> data = ...;
var q = data.Where(x => p(x)).Orderby(x => k(x)).Select(x => f(x)); foreach
(var e in q) a(e);

// helyett, AsParallel<T>() metódus IParallelEnumerable<T>-rel tér vissza

IEnumerable<T> data = ...;
var q = data.AsParallel().Where(x => p(x)).Orderby(x => k(x)).Select(x =>
f(x)); foreach (var e in q) a(e);
```

C# 4.0

- Parallel.For, Parallel.Foreach
- A lambda-kifejezéseknek köszönhetően szintaktikailag hasonló formában, mint az eredeti for, foreach

```
for (int row=0; row < pixelData.GetUpperBound(0); ++row) {
  for (int col=0; col < pixelData.GetUpperBound(1); ++col) {
    pixelData[row, col] = AdjustContrast(pixelData[row, col],
    minPixel, maxPixel);
  }
}

Parallel.For(0, pixelData.GetUpperBound(0), row => {
  for (int col=0; col < pixelData.GetUpperBound(1); ++col) {
    pixelData[row, col] = AdjustContrast(pixelData[row, col],
    minPixel, maxPixel);
  }
});
```

C# 4.0

- Dinamikus programozás támogatása (CLR + DLR)
- Nagyobb rugalmasság – futási időben több hibalehetőség
- dynamic d = GetDynamicObject(…);
  d.M(7);
  d statikus típusa dynamic!
- Új .Net-es nyelvek: IronPython, IronRuby

C# 5.0

- Még csak concept…
- Fordító, mint szolgáltatás
  - Metaprogramozás, programmal generálunk programot, ami…

```
CSharpEvaluator cE = new CSharpEvaluator();
cE.Usings.Add("System");
cE.Eval("for (int i=0; i<10; ++i) Console.WriteLine(i*i);");
```

Köszönöm a figyelmet!