

C-Sharp nyelv alapjai 3.

Attribútumok

Nyelvi elemekhez jelzések, tulajdonságok a fordítóprogram számára, hogy valamit másképp kell csinálni, vagy magamnak jelezhetek valamit. Olyan dolgokhoz lehet csak attribútumot írni, amihez assembly kódban meta adatok tartozhatnak (típus,névtér,objektum,)

PL: enum típushoz lehet FLAGS attribútumot beállítani:

```
[Flags]
enum Abcd {A = 1, B = 2, C = 4, D = 8}
```

(Flags azért jó mert myvar = Abcd.A | Abcd.C felvehetünk és erre .ToString eredménye A,B lesz)

Példa: Sql tárolt eljárást hívjon meg egy osztályt, ezeket a paramétereket programból felhúzni nehéz. V eljárásához függvényt írok paramétereibe tárolt eljárás paramétereinek feleljen meg függvény név = tárolt eljárás neve. Saját attribútummal jelezzük, hogy tárolt eljárásról van szó, valamint a paraméterek SQL-típusát is jelezhetjük attribútumokkal.

Függvény törzse semmi... Aspect-eket lehet csinálni .Net-ben. Meg tudom mondani függvénynek adott kontextusba kerülve adott utasítást csináljon. Nézzem meg ki hívta, az attribútumokból kérdezzem le a tárolt eljárás nevét és paramétereit és állítsa elő a tárolt eljárást.

Tárol el és adathozzáférési rétegen melyiket hívja.

2 fajta:

*Beépített attribútumok(pl Flags) - fordítóprogram fele jelez pl: felsorolás típus enum FelsTip(F1, F2...)

*Definiált / Saját attribútum létrehozása

+++

```
[AttributeUsage(AttributeTargets.Method|AttributeTargets.Class)]
class ElteIkNetAttribute : System.Attribute //System.Attribute-ból kell származnia!
{
    public ElteIkNetAttribute(string text)
    {
        this.text = text;
    }

    public int AdditionalParameter //property neve (set get)
    {
        get { return additionalParameter; }
        set { additionalParameter = value; }
    }

    private string text;
    private int additionalParameter;
}

[ElteIkNet("ez egy osztaly előtt van")]//tulajdonságokat irtunk
```

```
class AttrExampleClass
{
    [ElteIkNet("ez egy fuggveny előtt van", AdditionalParameter = 1)]
    public void AttrExampleFunction()
    {
    }

    [ElteIkNet("ez egy fuggveny előtt van", AdditionalParameter = 2)]
    public void AttrExampleFunction2()
    {
    }
}
+++
```

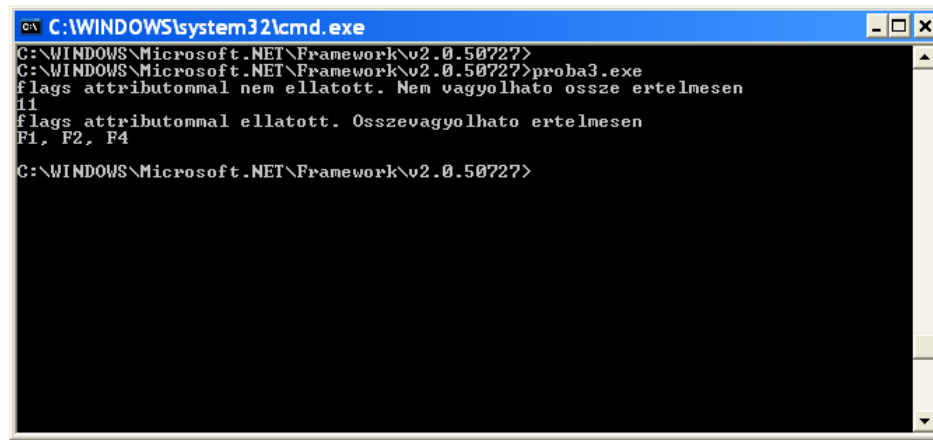
[Flags] attribútum jelölése, nyelvi elem elé írható // pl: enum a kettő hatványai
Az enumoknál meg kell adni, hogy a különböző tagoknak az 1,2,4,8 értéket, csak az értékek összevagyolásánál van különbség.

```
+++
enum FelsorTipus
{
    F1=1,
    F2=2,
    F3=4,
    F4=8
}

[Flags]
enum FelsorTipusBitfield
{
    F1=1,
    F2=2,
    F3=4,
    F4=8
}

static void Main(string[] args)
{
    Console.WriteLine("flags attributommal nem ellatott. Nem
        vagyolható össze értelmesen");
    FelsorTipus f = FelsorTipus.F1 | FelsorTipus.F2 | FelsorTipus.F4;
    Console.WriteLine(f);
    Console.WriteLine("flags attributommal ellatott. Osszevagyolható
        értelmesen");
    FelsorTipusBitfield f2 = FelsorTipusBitfield.F1 |
        FelsorTipusBitfield.F2 | FelsorTipusBitfield.F4;
    Console.WriteLine(f2);
}
+++
```

Az első esetben számoknak veszi és összeadja – Az output 11 lesz (1+2+8)
A második esetben pedig szépen kiírja a tagokat : F1, F2, F3



```

C:\WINDOWS\system32\cmd.exe
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>proba3.exe
flags attributommal nem ellatott. Nem vagyolható össze ertelmesen
11
flags attributommal ellatott. Osszevagyolható ertelmesen
F1, F2, F4
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
    
```

Becsukható és átláthatóbb a kód, ha régiókra bontjuk.

#region Interfaxe example

#end region

A + vagy – al becsukni illetve kinyitni lehet.

Genericek

Generic - típussal paraméterezhető programkód.

Generikus osztályok: osztály paraméterezése típussal. `System.Collection.Generic` névtér.

Pl. `Dictionary`, `List`, `Queue`. Fordítási időben történik meg a típusellenőrzés. Csak akkor fordul, ha az összes behelyettesíthető típusra értelmes a kód. Azért jobb mint a futási idejű behelyettesítés, mert hatékonyabb lesz így a kód.

A típusra megkötések adhatók meg:

- interfész: `where T: MyInterface,`
- ősosztály: `where T: BaseClass,`
- default konstruktor megléte: `where T: new(),`
- érték vagy referencia típus.
- többféle megkötés: `where T: BaseClass, new(),`

Non-Generic

ArrayList
 Hashtable
 SortedList
 Queue
 Stack
 IEnumerable
 ICollection
 N/A
 IList
 CollectionBase
 ReadOnlyCollectionBase
 DictionaryBase
 IDictionary<TKey, TValue>
 N/A
 N/A
 N/A

Similar Generic Type

List<T>
 Dictionary<TKey, TValue>
 SortedList<TKey, TValue>
 Queue<T>
 Stack<T>
 IEnumerable<T>
 N/A (use IEnumerable<T> anything that extends it)
 ICollection<T>
 IList<T>
 Collection<T>
 ReadOnlyCollection<T>
 N/A (just implement
 SortedDictionary<TKey, TValue>
 KeyedCollection<TKey, TItem>
 LinkedList<T>

Kő-papír-olló játék

Alappéldát mutat interface, property, öröklődés használatára MS online játék központi webservice

Interface-nél meg tudom mondani mit várok.

Egymással játszanak 2 játékos 2 szer egymás után mutat valamit -> 3.-ra mást kell mutatni (25 meccs)

Tudni akarja előtte miket tett ellenfél: Step-be

Hf. megnézni

Reflection

(Reflexió)

Arra jó hogy futási időben típus információkat tudjak kinyerni beleértve, hogy egy osztály milyen interface-t implementál, miből származik, milyen metódusai vannak, sőt milyen attribútumai vannak.

Valamint ha külső modulból betölteni akarok valamit, és nem adtam hozzá referencia szerint csak behúzom (new típus) fordításkor nem ismerem. Reflection-el felderítem a külső modul-t és típusait, abból a típusból, ami nekem megfelel abból tudok létrehozni egy példányt futás időben névvel megadott módon műveleteket végrehajtani, Java-ban is van ilyen api. Alap C++-ban nincs, de van kiterjesztésekben (pl. managed c++).

Pl: reflection-t nem kell hozzáadni.

Assembly a. Load(...)... - betölteni vmit

a.GetTypes - ezel be lehet húzni a típusokat vagy direkt egyet név szerint

```
+++
```

```

    static void Main(string[] args)
    {

//          Assembly a = Assembly.Load(...);
//          a.GetType
        Console.WriteLine("Reflection example");
    }

```

```
// itt csaltunk a load... plaginek kellenének (ld. később)
Type opType = typeof(Operate);
//" ..." idézőjel között kell a nevet megadni, mint string
MethodInfo miRegOp = opType.GetMethod("RegisterOperation");
MethodInfo miDoOp = opType.GetMethod("DoCalculation");

//Osztályból egy példányt akarok, mintha nem ismerném ezért nem a new-t, hanem:
//Activator.CreateInstance(optype)-ot használjuk
object obj = Activator.CreateInstance(opType);

//melyik példány, milyen paraméterrel
miRegOp.Invoke(obj, new object[] { new Addition() });
miDoOp.Invoke(obj, new object[] { 3, 4 });

//attrEx, milyen attribútumok vannak az osztálynak HF megnézni
Type attrEx = typeof(AttrExampleClass);
attrEx.GetCustomAttributes
//
}
+++
```

Példa2:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Reflection;

namespace AttrGenRefl
{
    [AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]
    class SajjatAttribute : Attribute //saját attribútum osztály
    {
        private string _msg;
        private int _code;
        public int Code
        {
            get { return _code; }
            set { _code = value; }
        }

        public string Msg
        {
            get { return _msg; }
        }

        public SajjatAttribute(string msg)
        {
            _msg = msg;
        }
    }

    [Serializable]
    [Sajjat("Üzenet")]
    class PeldaOsztaly
    {
    }

    [Sajjat("Üzenet", Code=123)]
    class PeldaOsztaly2
    {
        private int myVar;

        //[Sajjat("ÜzenetProp", Code = 123134)]
        public int MyProperty
        {
        }
    }
}
```

```

        get { return myVar; }
    }

    [Sajat("UzenetFv", Code = 123134)]
    public void Fv()
    {
        Console.WriteLine("aaa");
    }
}

class Program
{
    static void Main(string[] args)
    {
        MethodInfo[] methods = typeof(PeldaOsztaly2).GetMethods();
        foreach (MethodInfo m in methods)
        {
            Console.WriteLine(m.Name);
        }

        MethodInfo mi = typeof(PeldaOsztaly2).GetMethod("Fv", BindingFlags.Instance |
BindingFlags.Public);

        Type type = typeof(PeldaOsztaly2);

        object obj = Activator.CreateInstance(type);

        mi.Invoke(obj, null);
        object []attrs = mi.GetCustomAttributes(typeof(SajatAttribute), false);
        foreach (SajatAttribute sa in attrs)
        {
            Console.WriteLine(sa.Msg);
        }
    }
}

```

```

C:\WINDOWS\system32\cmd.exe
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>proba4.exe
get_MyProperty
Fv
GetType
ToString
Equals
GetHashCode
aaa
UzenetFv
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>

```

Mi az az assembly?

Emlékeztetés: .Net-es értelemben félig lefordított .dll vagy .exe.

Ez IL kód, nem gépi kód. A különböző platformokhoz különböző JIT(Just In Time) fordítók vannak amik natív kódra fordítják le (futási időben).

Egy assambly(szerelvény) négy fő részből áll

1. Manifeszt: az assamblyhoz kapcsolódó metaadatokat írja le

Ebben van:

- assembly neve
 - verziószám
 - nyelv és kultúra
 - a hozzá tartozó állományok listája
 - táblázat amiben benne van, hogy melyik típus melyik modulban van
 - hivatkozások más szerelvényekre
2. Típusdefiníciók – típusok definíciója
 3. Implementáció – típusok megvalósítása
 4. Erőforrások – szükséges erőforrások (képek, konstansok,)

Megkülönböztetünk private és shared assemblyket.

Private: csak a mi alkalmazhatjuk, saját célra és nem publikáljuk

Shared: több alkalmazás használhatja és központi helyen tároljuk

(Lásd még : ennek jegyzetnek az első fejtében Intermediate Language - .NET assembly)

Strong name - erős név

Erős név = név + verzió szám + nyelv + titkos kulcs(hasonló RSA-hoz)

.Net előtt ez nem igazán volt. Így lehet ugyanazzal a fájlnevvvel több különböző assembly. Nagy előnye van, mert voltak problémák amiket kiküszöbölhetünk vele:

Példák: A dll elnevezése Unixban: libc-2-3-verzió... a dll elnevezésében a verzió volt. Mindig változott. Windowsban a dll msvcrt 6 msvcrt 7 kompatibilis-e? Bug az 1.0-ban volt és nem lehet kijavítani 2.0-ban, mert a ráépülő alkalmazások meghalhatnak.

Esetleg két gyártó ugyanolyan néven publikál két dll-t különböző funkcionalitással. És mindkettő a system32-be akarta rakni: ez a "dll hell". Az alkalmazások nem futnak, mert például egy install felül vágta.

WinSXS - eltérő verziójú DLL-eket tárolja az alkalmazások számára, és mindig a megfelelőt hajtja végre amikor hívás történik az assembly-be

* Side by Side execution: SxS: PI: A comctl32.dll egyik verzióját: Common Control alkalmazás akarja futtatni, akkor át van irányítva egy könyvtárba - aminek egyedi elnevezése van.

Kulcs létrehozása, assembly aláírása - sn.exe programmal

.Netben nem állnak neki a "dll hell"-el bohóckodni. Ugyanaz a név, de különböző legyen: Letehte öbb xy.dll nevű, de különböző tulajdonságai vannak a néven kívül: +verziószám +culture+kulcs = Strong name

Betöltő (Fusion) - GAC, natív image

- dll hell -> SxS *

- Mire jó a GAC

GAC – Global Assembly Cache

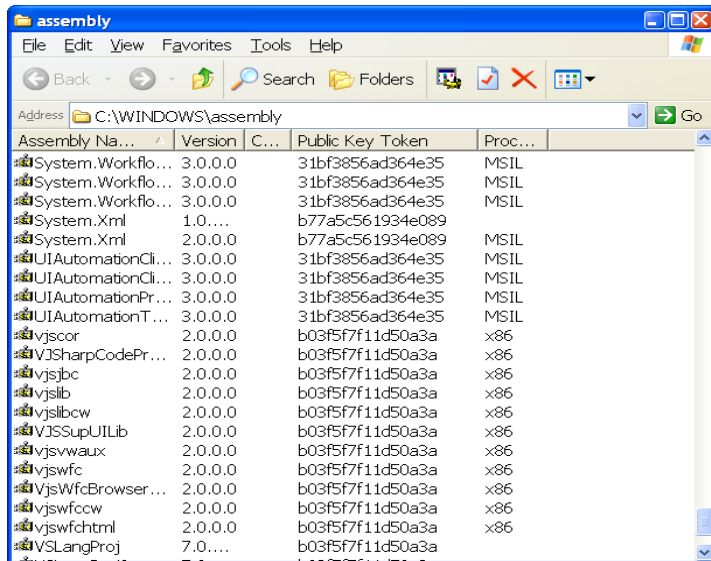
Rendszer szintű shared assemblyk csak erős névvel. (Egyedien azonosítja a nevet.)

C:\Windows\assembly –ben. Innen veszik a programok a szükséges assemblyket.

Ide alapon IL assemblyk kerülnek .NET alkalmazások esetén.

De lefordíthatjuk natív kódra "ngen" kulcsszóval (ingyenes program, .NET része).

Ez jóval gyorsabb és hatékonyabb kód, de platform függő és ekkor is kell az eredeti hozzá, mert abban vannak meta adatok.



Assembly segédprogrammal kilistázzhatjuk assemblyket: IL assembly-k MSIL, a natív assembly-k X86.
gacutil.exe – segédprogram GAC-hoz

Fontos : a programok először a GAC-ban keresik a megfelelő dll-t és csak utána az aktuális könyvtárban

Egyedi kulcsot generálni kell az "sn" paranccsal: sn -k ExternalModule.snk

C-Sharp nyelv alapjai 3	1
Attribútumok	1
Genericcek	3
Kő-papír-olló játék	4
Reflection	4
Mi az az assembly?	6
Strong name - erős név	7
Betöltő (Fusion) - GAC, natív image	7