Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

1. előadás

A .NET keretrendszer és a C# programozási nyelv

Giachetta Roberto

A jegyzet az ELTE Informatikai Karának
2014. évi Jegyzetpályázatának támogatásával készült

A .NET keretrendszer

Eredete

- 1980-ban jelent meg a *SmallTalk*, amely az első *tisztán objektumorientált* programozási nyelv
  - minden elem objektum, és egyben egy osztály példánya, az osztályok egymástól öröklődnek, és egy *teljes származtatási hierarchiá*ban helyezkednek el
  - teljesen hordozható kód, *virtuális gép*en történő futtatás, *dinamikus programozás* támogatása
  - memóriafelügyelet és *szemétgyűjtés*
- 1991-ben jelent meg a *Java*, amely a C++-ra alapozva valósította meg a tisztán objektumorientált koncepciót
  - jelentős méretű osztálykönyvtár minden lehetséges feladatra

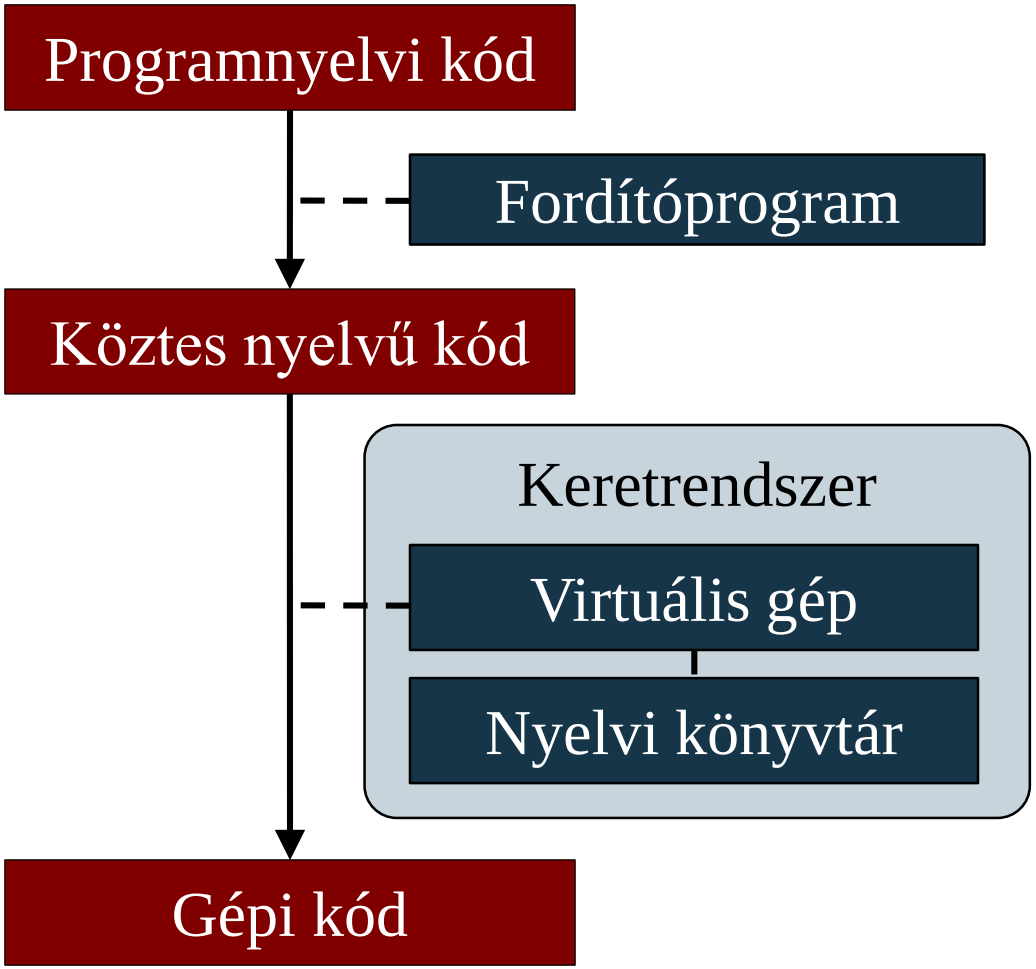
A .NET keretrendszer

A szoftver keretrendszer

- A hordozhatóságot, gyors futást és hibamentességet *futásidejű fordítás* (*Just In Time Compilation*) biztosítja
  - az eredeti kód egy alacsony szintű, de gépfüggetlen kódra fordul, az a *köztes nyelvű kód* (*Intermediate Language*)
  - a köztes nyelvű kódot külön kell értelmezni, erre szolgál a *virtuális gép* (*Virtual Machine*), ami egyúttal lehetőséget ad a memóriafelügyeletre
- *Szoftver keretrendszer*nek nevezzük a kiemelt programkönyvtár és a virtuális gép együttesét
  - tartalmazza az API gépfüggetlen, absztrakt lefedését
  - felügyeli a programok futásának folyamatát

A .NET keretrendszer

A szoftver keretrendszer



A .NET keretrendszer

Megvalósítása

- A Microsoft egy saját Java környezetet (Visual J++) készített, majd ebből kiindulva egy egységes platformot fejlesztett nyelvei (Visual C++, Visual Basic) számára, közös virtuális géppel és nyelvi könyvtárral
- 2001-re készült el a *.NET Framework*, és a dedikáltan ráépülő nyelv, a C#
  - virtuális gép: *Common Language Runtime* (*CLR*)
  - köztes nyelv: *Common Intermediate Language* (*CIL*)
  - egységes típusrendszer: *Common Type System* (*CTS*)
  - teljes körű programkönyvtár: *Base Class Library* (*BCL*), *Framework Class Library* (*FCL*)

A .NET keretrendszer

Fejlődése

- 1.1 (2003): kompakt keretrendszer, kódbiztonság (CAS)
- 2.0 (2005): eléri a Java funkcionalitását (sablonok, parciális osztályok, névtelen metódusok), új adatkezelés (ADO.NET)
- 3.0 (2007): technológiák egységesítése (WCF, WPF, WF, CardSpace)
- 3.5 (2008): funkcionális programozás, nyelvbe ágyazott lekérdezések (LINQ), AJAX támogatás
- 4.0 (2010): párhuzamosítás támogatása (PLINQ, TPL), szerződés alapú programozás (DbC)
- 4.5 (2012): nyelvi szintű párhuzamosítás, Modern UI

A .NET keretrendszer

Jellemzői

- Összesen 37 nyelv támogatott, az alapvető nyelvek a *C#*, *Visual C++*, *Visual Basic.NET*, *F#*
- Hátrányai:
  - megnövelt erőforrásigény, lassabb programindítás és futtatás, mint fordított nyelveknél
  - kód visszafejthetőség
- Kapcsolódó keretrendszerek:
  - *Mono*, *Moonlight*, *Xamarin*: Linux, OS X, iOS, Android
  - *Silverlight*, *XNA*: multimédia, web, Windows Phone, XBox
  - *Windows Runtime*: Windows RT és 8, Windows Phone 8

A .NET keretrendszer

A fejlesztőkörnyezet

- A szoftverfejlesztéshez szükséges egy *integrált fejlesztőkörnyezet* (IDE), amely lehető legjobb módon támogatja a programok tervezését, implementálását, és tesztelését
- A *Visual Studio* az egyik legtöbb funkcionalitást nyújtó fejlesztőkörnyezet is számos téren elősegíti a szoftverfejlesztést, úgymint:
  - csapatmunka támogatás (verziókövetés, feladatkövetés)
  - tervezés (architektúra, végrehajtás, terv-kód szinkronizálás)
  - kód-kiegészítés, kód-újratervezés, kód-stílus követés
  - hibakeresés, tesztelés, kódelemzés

A .NET keretrendszer

Támogató eszközök

- Ezen felül számos kiegészítő eszköz adott, amely növeli a .NET alapú fejlesztés hatékonyságát:
  - natív kódelemzés (*IL Disassembler*, *.NET Reflector*)
  - statikus kód elemzés (*FxCop*, *NDepend*, *StyleCop*)
  - kódrészletek előállítása (*Snipper Compiler*, *Regulator*)
  - kód-újratervezés (*Resharper*)
  - automatizált tesztelés (*NUnit*, *TestDriven.NET* *CruiseControl.NET*, *Moq*)
  - teljesítménytesztelés (*dotTrace*)
  - automatikus dokumentálás (*GhostDoc*, *NDoc*, *Sandcastle*)

A .NET keretrendszer

Fordítás és futtatás

- A Visual Studio lehetőséget ad a fordítás módjának konfigurálására
  - alapvetően két üzemmód támogatott:
    - *Debug*: nyomkövetéshez optimalizált, így a kód számos kiegészítést tartalmaz
    - *Release*: végleges használatra optimalizált, így a kód jelentős optimalizáláson megy keresztül
  - a fordításhoz megadhatjuk a célplatformot (x86, x64, vagy tetszőleges)
- A lefordított (köztes nyelvű) program tartalmazza az eredeti kódot, és visszafejthető (pl. *.NET Reflector* segítségével)

A .NET keretrendszer

Hibakeresés

- *Hibakeresés* (*debugging*) során a programot futás közben elemezzük, követjük a változók állapotait, a hívás helyét, felfedjük a lehetséges hibaforrásokat
- A fontosabb hibakövetési funkciók:
  - *megállási pontok* (*breakpoint*) elhelyezése, ahol a program futása megáll, és várakozó állapotba lép
  - *változókövetés* (*watch*), amely automatikus a lokális változókra, szabható rá feltétel
  - *hívási lánc* (*call stack*) kezelése, a felsőbb szintek változóinak nyilvántartásával
  - *léptetés* (*step-in*): az utasítások lépésenkénti végrehajtása

A C# programozási nyelv

Lehetőségei

- A *C#* tisztán objektumorientált programozási nyelv, amely teljes mértékben a *.NET Framework*re támaszkodik
  - szintaktikailag nagyrészt C++, megvalósításában Java
  - egyszerűsített szerkezet, strukturált felépülés névterekkel
  - tisztán objektumorientált, minden típus egy .NET keretrendszerbeli osztály, vagy leszármazottja
  - támogatja a sablon-, eseményvezérelt, és funkcionális programozást
  - a forrásfájl kiterjesztése: **`.cs`**
  - kódolás: Unicode 3.0

A C# programozási nyelv

A „Hello, World!” program

```
using System; // névtér használatba vétele

namespace Hello // névtér
{
   class HelloWorld // osztály
   {
      static void Main() // statikus főprogram
      {
         Console.WriteLine("Hello, World!");
         // kiírás konzol képernyőre (a Console
         // osztály statikus WriteLine metódusa)
      }
   }
}
```

A C# programozási nyelv

Névterek

- A névterek biztosítják a kód logikai felbontását, minden osztálynak névtérben kell elhelyezkednie
  - hierarchikusan egymásba ágyazhatóak (ponttal jelölve)
  - nincs globális, névtelen névtér
- Névtereket használatba venni a **`using <névtér>`** utasítással lehet (az utasítás hatóköre a fájl)
  - pl.:
```
using System;
using System.Collections.Generic;
```
  - az osztálynév előtt is megadhatjuk a névteret (így nem kell **`using`**), pl.:
```
System.Console.WriteLine("Hello, World!");
```

A C# programozási nyelv

Típusok

- A nyelv három típuskategóriát különböztet meg:
  - *érték*: érték szerint kezelendő típusok, mindig másolódnak a memóriában, és a blokk végén törlődnek
  - *referencia*: biztonságos mutatókon keresztül kezelt típusok, a virtuális gép és a szemétgyűjtő felügyeli és törli őket
  - *mutató*: nem biztonságos mutatók, amelyek csak felügyeletmentes (`unsafe`) kódrészben használhatóak
- Minden típus objektumorientáltan van megvalósítva, és része a teljes származtatási hierarchiának
- A *primitív típusok* két névvel rendelkeznek: C# programozási nyelvi név és .NET könyvtárbeli megfelelő típusnév

A C# programozási nyelv

Primitív típusok

- Primitív típusok:
  - logikai: **`bool`** (**`Boolean`**)
  - egész: **`sbyte`** (**`SByte`**), **`byte`** (**`Byte`**), **`short`** (**`Int16`**), **`ushort`** (**`UInt16`**), **`int`** (**`Int32`**), **`uint`** (**`UInt32`**), **`long`** (**`Int64`**), **`ulong`** (**`UInt64`**)
  - lebegőpontos: **`float`** (**`Single`**), **`double`** (**`Double`**)
  - tizedestört: **`decimal`** (**`Decimal`**)
  - karakter: **`char`** (**`Char`**)
  - objektum (ősosztály): **`object`** (**`Object`**)
  - szöveg: **`string`** (**`String`**)

A C# programozási nyelv

Primitív típusok

- A primitív típusok is intelligensek, azaz támogatnak számos műveletet és speciális értéklekérdezést, pl.:
  - speciális értékek: **`Int32.MaxValue`**, **`Double.NaN`**, **`Double.PositiveInfinity`**, **`String.Empty`**
  - konverziós műveletek: **`Double.Parse(…)`**
  - karakterműveletek: **`Char.ToLower(…)`**,
  - szöveg műveletek: **`str.Length`**, **`str.Find(…)`**, **`str.Replace(…)`**
- A szimpla konstansok is intelligens objektumok, pl. **`10.ToString()`**, **`"Hello World".SubString(0, 5)`**

A C# programozási nyelv

Típuskezelés

- A nyelv *szigorúan típusos*, tehát minden értéknek fordítási időben ismert a típusa, és nem enged meg értékvesztést
  - nagyobb halmazra implicit típuskonverzió, kompatibilis halmazra explicit típuskonverzió használható, pl.:

```
int x = 1; double y = 2, string z;
Y = x; // implicit típuskonverzió
x = (int)y; // explicit típuskonverzió
z = (string)y; // hiba, nem kompatibilisek
```

  - primitív típusok konverziójához a **`Convert`** osztály, illetve egyéb metódusok is rendelkezésre állnak, pl.:

```
x = Convert.ToInt32(y);
z = Convert.ToString(y); // vagy y.ToString();
x = Convert.ToInt32(z); // vagy Int32.Parse(z);
```

A C# programozási nyelv

Példányosítás

- Változókat bármely (nem névtér) blokkon belül létrehozhatunk a programkódban típus, név és kezdőérték megadásával
  - pl.: `Int32 myInt = 10;`
  - felhasználás előtt mindenképpen kell kezdőértéket kapnia
  - összetett típusok esetén a **`new`** operátort használjuk, pl.: `Stack<Int32> s = new Stack<Int32>();`
- Típusnév helyett használható a **`var`** kulcsszó, ekkor a típus behelyettesítődik fordítási időben, pl.: `var myInt = 10;`
- Konstansokat a **`const`** kulcsszóval, konstruktorban értékül adható mezőket a **`readonly`** kulcsszóval adhatunk meg

A C# programozási nyelv

Vezérlési szerkezetek

- *Szekvencia*: a `;` tagolja az utasításokat
- *Programblokk*: **`{ <utasítások> }`**
- *Elágazás*: lehet kétágú (**`if`**), illetve többágú (**`switch`**), utóbbinál az ágakat le kell zárni (**`break`**, **`goto`**, **`return`**)
- *Ciklus*:
  - számláló (**`for`**), előtesztelő (**`while`**), utántesztelő (**`do … while`**)
  - bejáró (egy **`IEnumerable`** gyűjtemény elemein halad végig):
    `foreach (<deklaráció> in <gyűjtemény>)`
    `  <utasítás>;`

A C# programozási nyelv

Osztályok

- A C# programozási nyelv *tisztán objektumorientált*, ezért minden érték benne objektum, és minden típus egy osztály
  - az osztály lehet érték szerint (*struct*), vagy referencia szerint kezelt (*class*), utóbbi élettartama független a blokktól
  - az osztály tagjai lehetnek mezők, metódusok, események, tulajdonságok (*property*), illetve más (beágyazott) osztályok
    - a tulajdonság lényegében a lekérdező (*get*) és beállító műveletek (*set*) absztrakciója
  - minden tagnak, és az osztályt is jelöljük a láthatóságát (**`public`**, **`private`**, **`protected`**, **`internal`**)
  - a nyílt rekurziót a **`this`** kulcsszó biztosítja

A C# programozási nyelv

Osztályok

```
<láthatóság> class/struct <osztálynév> {
   <láthatóság> <típus> <mezőnév>; // mező
   …
   <láthatóság> <típus> <metódusnév>
     ([ <paraméterek> ]) { <törzs> } // metódus
   …
   <láthatóság> <típus> <tulajdonságnév> {
      [ get { <törzs> } ]
      [ set { <törzs> } ]
   } // tulajdonság
   …
   <láthatóság> event <delegált> <eseménynév>;
     // esemény
}
```

A C# programozási nyelv

Osztályok felépítése

- A *mezők* típusból és névből állnak, illetve kaphatnak kezdőértéket (csak referencia szerinti osztályban)
  - a mezők alapértelmezett értéket kapnak, amennyiben nem inicializáljuk őket
- A *metódusok* visszatérési típussal (amennyiben nincs, akkor **`void`**), névvel és paraméterekkel rendelkeznek
  - a konstruktor neve megegyezik a típussal, a destruktort általában nem valósítjuk meg (szemétgyűjtés miatt)
  - lehetnek cím szerinti (**`ref`**), kimenő (**`out`**), alapértelmezett, tetszőleges számú (**`params`**) paraméterek
  - a paraméterek átadhatóak név szerint

A C# programozási nyelv

Osztályok felépítése

- Pl. C++:

```
class Rational {
   private:
      int num;
      int denom;
   …
   public:
      Rational(int, int);
   …
};
…
Rational::Rational(int n, int d) {
   num = n; denom = d;
}
```

A C# programozási nyelv

Osztályok felépítése

- Pl. C#:

```
struct Rational {
    private Int32 num; // mező
    private Int32 denom;
        // mindenhol jelöljük a láthatóságot
    …
    public Rational(Int32 n, Int32 d) { // metódus
        num = n;
        denom = d;
        // a deklaráció és a definíció nem
        // választható el
    }
    …
} // nem kell a végén ; ☺
```

- A tulajdonság egy könnyítés a programozónak a lekérdező és író műveletek absztrakciójára
  - a beállító tulajdonságban a **`value`** pszeudováltozó veszi át az értéket
  - pl. C++:

```
class Rational {
   …
   int getDenominator() { return denom; }
   void setDenominator(int value) {
      denom = (value == 0) ? 1 : value;
   }
   // publikus lekérdező és beállító művelet
}
```

A C# programozási nyelv

Osztályok felépítése

- Pl. C#:

```
struct Rational {
   …
   public Int32 Denominator {
      get { return denom; }
      set { denom = (value == 0) ? 1 : value; }
   } // változóhoz tartozó publikus tulajdonság
}
…
Rational r = new Rational(10, 5);
r.Denominator = 10; // a 10 kerül a value-ba
```

- külön definiálható csak lekérdező, csak beállító művelet
- tulajdonsággal lehet automatikusan mezőt is létrehozni

A C# programozási nyelv

Felsorolási típusok

- A *felsorolási típus* (**enum**) értékek egymásutánja, ahol az értékek egész számoknak feleltethetőek meg (automatikusan 0-tól sorszámozva, de ez felüldefiniálható), pl.:

```
enum Munkanap { Hétfő, Szerda = 2, Csütörtök }
```

  - a hivatkozás a típusnéven át történik, pl.:

```
Munkanap mn = Munkanap.Hétfő; …
if (mn == Munkanap.Szerda) { … }
```

  - egy változó több értéket is eltárolhat, pl.:

```
Munkanap mn = Munkanap.Hétfő | Munkanap.Szerda
```

  - ez is egy osztály a **System** névtérben:

```
public abstract class Enum : ValueType, …
```

A C# programozási nyelv

Elemi osztályok

- Az *elemi osztály* (*struct*) egy egyszerűsített osztály, amely:
  - mindig érték szerint kezelődik
  - nem szerepelhet öröklődésben, de implementálhat interfészt
  - alapértelmezett konstruktora mindig létezik
- Pl.:

```
struct Rational { … } // elemi osztály
…
Rational r = new Rational(10, 5);
Rational t = r; // r érték szerint másolódik
t.Denominator = 10; // itt r.Denominator == 5
```

Elemi és referencia osztályok

- A *referencia osztály* (*class*) a teljes értékű osztály, amely öröklődésben is szerepelhet
  - csak egy őse lehet, de bármennyi interfészt megvalósíthat
  - mezőit lehet közvetlenül inicializálni
  - az öröklődés miatt lehet absztrakt osztály, és szerepelhetnek benne absztrakt és virtuális elemek
- Pl.:

```
class Rational { … } // referencia osztály
…
Rational r = new Rational(10, 5);
Rational t = r; // r cím szerint másolódik
t.Denominator = 10; // itt r.Denominator == 10
```

A C# programozási nyelv

Statikus osztályok

- Lehetőségünk van *statikus osztályok*, *mezők*, *tulajdonságok és műveletek* létrehozására a **`static`** kulcsszó használatával, pl.:

```
static class NumClass { // statikus osztály
   private static Int32 nr = 10;
      // statikus mező 10 kezdőértékkel
   public static Int32 Nr { get { return nr; } }
      // statikus tulajdonság
   public static void Increase() { nr++; }
      // statikus metódus
}

Console.WriteLine(NumClass.Number) // eredmény: 10
NumClass.Increase();
Console.WriteLine(NumClass.Number) // eredmény: 11
```

A C# programozási nyelv

Öröklődés

- A .NET keretszerben az osztályok egy teljes származtatási hierarchiában vannak
  - minden osztály őse az **`Object`**, így megkapja annak műveleteit (pl.: **`Equals(…)`**, **`GetHashCode()`**, **`ToString()`**)
  - csak egyszeres öröklődés van, a konstruktor és destruktor automatikusan öröklődik
  - az ős tagjaira (beleértve a konstruktort) a **`base`** kulcsszón keresztül hivatkozhatunk
  - polimorfizmus során lehetőségünk van a típusazonosításra (**`is`**), valamint az explicit, illetve biztonságos típuskonverzióra (**`as`**)

A C# programozási nyelv

Öröklődés

- Pl.:

```
class BaseClass /* : Object */ { // ősosztály
   public Int32 Value;
   public BaseClass(Int32 v) { value = v; }
}
…
class DerivedClass : BaseClass { // leszármazott
   public BaseClass(Int32 v) : base(v) { }
      // ős konstruktorának meghívása
}
…
Object o = new DerivedClass(1); // polimorfizmus
if (o is BaseClass) // típusazonosítás, konverzió
   Console.WriteLine((o as BaseClass).Value)
```

A C# programozási nyelv

Öröklődés

- Öröklődés során a műveletek és tulajdonságok felüldefiniálhatóak, illetve elrejthetőek
  - felüldefiniálni csak a *virtuális* (**`virtual`**) és *absztrakt* (**`abstract`**) műveleteket, tulajdonságokat lehet, a felüldefiniálást is jelölnünk kell az **`override`** kulcsszóval
  - elrejteni a **`new`** kulcsszóval lehet (polimorfizmusnál nem érvényesül)
  - absztrakt metódusok törzs nélküliek, absztrakt tulajdonságoknál csak azt kell jelezni, hogy lekérdezésre, vagy értékadásra szolgálnak-e
  - absztrakt tagot tartalmazó osztály is absztrakt (szintén jelölnünk kell)

A C# programozási nyelv

Öröklődés

- Pl.:

```
class BaseClass { // ősosztály
    public void StandardMethod() {
        // lezárt (nem felüldefiniálható) művelet
        Console.WriteLine("BaseStandard");
    }
    public virtual void VirtualMethod() {
        // virtuális (felüldefiniálható) művelet
        Console.WriteLine("BaseVirtual");
    }
}
```

A C# programozási nyelv

Öröklődés

- Pl.:

```
class DerivedClass : BaseClass {
    public new void StandardMethod() {
        // művelet elrejtés
        Console.WriteLine("DerivedStandard");
    }
    public override void VirtualMethod() {
        // művelet felüldefiniálás
        base.VirtualMethod();
            // a felüldefiniált művelet meghívása
        Console.WriteLine("DerivedVirtual");
    }
}
```

A C# programozási nyelv

Öröklődés

- Pl.:

```
DerivedClass dc = new DerivedClass();
dc.StandardMethod(); // eredmény: DerivedStandard
dc.VirtualMethod();
   // eredmény:
   // BaseVirtual
   // DerivedVirtual

…
BaseClass bc = new DerivedClass();
bc.StandardMethod(); // eredmény: BaseStandard
bc.VirtualMethod();
   // eredmény:
   // BaseVirtual
   // DerivedVirtual
```

A C# programozási nyelv

Öröklődés

- Pl.:

```
abstract class BaseClass { // absztrakt ősosztály
   public abstract Int32 Value { get; }
      // absztrakt lekérdező tulajdonság,
      // felüldefiniálandó
   public abstract void AbstractMethod();
      // absztrakt metódus, felüldefiniálható
   public virtual void VirtualMethod() {
      Console.WriteLine(Value);
   }
}
…
BaseClass b = new BaseClass();
// hiba: absztrakt osztály nem példányosítható
```

A C# programozási nyelv

Öröklődés

```
class DerivedClass : BaseClass {
   public override Int32 Value {
      get { return 1; }
   } // tulajdonság felüldefiniálás
   public sealed override void AbstractMethod() {
      VirtualMethod();
      Console.WriteLine(2 * Value);
   }
}
…
BaseClass bc = new DerivedClass();
bc.AbstractMethod();
// eredménye:
// 1
// 2
```

A C# programozási nyelv

Interfészek

- Az *interfész* (*interface*) egy tisztán absztrakt osztály, deklarációk halmaza, amelyet az osztályok implementálnak
  - a többszörös öröklődés kiküszöbölésére szükséges
- Pl.:

```
interface IDoubleCompatible {
   Double ToDouble(); // láthatóság, törzs nélkül
}
…
struct Rational : IDoubleCompatible {
   …
   // interfész megvalósítása:
   public Double ToDouble(){ … }
}
```

A C# programozási nyelv

Attribútumok

- Az *attribútum*ok (*attribute*) olyan speciális osztályok, amely elsősorban a virtuális gépnek szolgálnak információkat (úgynevezett *metaadat*okat)
  - kiegészítik a kód deklarációit, és segítségre lehetnek a kód kezelésében, *reflexió* segítségével kezelhetőek
  - a deklaráció előtt adjuk meg őket, alkalmazhatóak osztályra, metódusra, paraméterre, …
- Pl.:

```
[Serializable] // attribútumok
[ComVisible]
class SomeClass { … }
```