INTERFACE

Készítette: Vastag Atila
2020

A C# nyelvben minden osztálynak csak egyetlen közvetlen őse lehet, ugyanakkor sok esetben hasznos lenne, ha egy osztály több helyről is örökölhetne tulajdonságokat, metódusokat.

Egy osztálynak csak egy közvetlen őse lehet, de több **interfészt** is megvalósíthat, sőt interfészt akár struktúrák esetében is használhatunk.

Az *interfész publikus absztrakt metódus deklarációk, tulajdonságok, indexelők és események* összessége.

Az absztrakt osztályok és interfészek nagyban hasonlítanak egymásra, ha egy interfész egy osztály ősének állítunk, akkor **meghatározza az osztály felületét, előír egy mintát.**

A nagy különbség a kettő közt az, hogy míg az absztrakt osztály eleve meghatároz egy osztályhierarchiát, egy interfész nem köthető közvetlenül egy osztályhoz, mindössze előír egy mintát, amit meg kell valósítani.

A .NET interfész egy olyan szerződés, amelyhez minden leszármazott osztálynak tartania kell magát. Ez a szerződés formai és működésbeli elemeket is tartalmazhat. Az interfészek lehetővé teszik olyan típusok létrehozását .NET környezetben, amik többféle viselkedésmódot is támogatnak. Az interfészt úgy is megfogalmazhatjuk, hogy definiálja mindazon képességek halmazát (metódusokat, property-ket), amelyekkel az ezen interfészt implementáló objektumoknak rendelkezni kell. A leszármazott osztály pedig megadja, hogy ezeket a képsségeket miként kell megvalósítani, vagyis implementálja a metódusokat.

Általában akkor használunk interfészt, amikor logikailag nem összetartozó osztályok egyforma metódust használnak, azaz ugyanazt a metódust használják, de más – más megvalósítással.

Az interfész metódusai üres törzzsel vannak deklarálva, tagjai nem tartalmazhatnak hozzáférés módosítót (mindig *public*). Önmagukban nem tudjuk használni, implementálni kell őket. Az implementációt egy-egy osztály végzi. Egy osztály több interfészt is implementálhat. Az interfész neve konvenció szerint nagy **I** betűvel kezdődik.

Szintaxisa:

```
[módosító] interface Iinterfésznév
{
 // absztrakt metódusok;
}
```

Amennyiben egy osztályból és interfészből is származtatunk, akkor a felsorolásnál az ososztály nevét kell előrevenni, utána jönnek az interfészek:

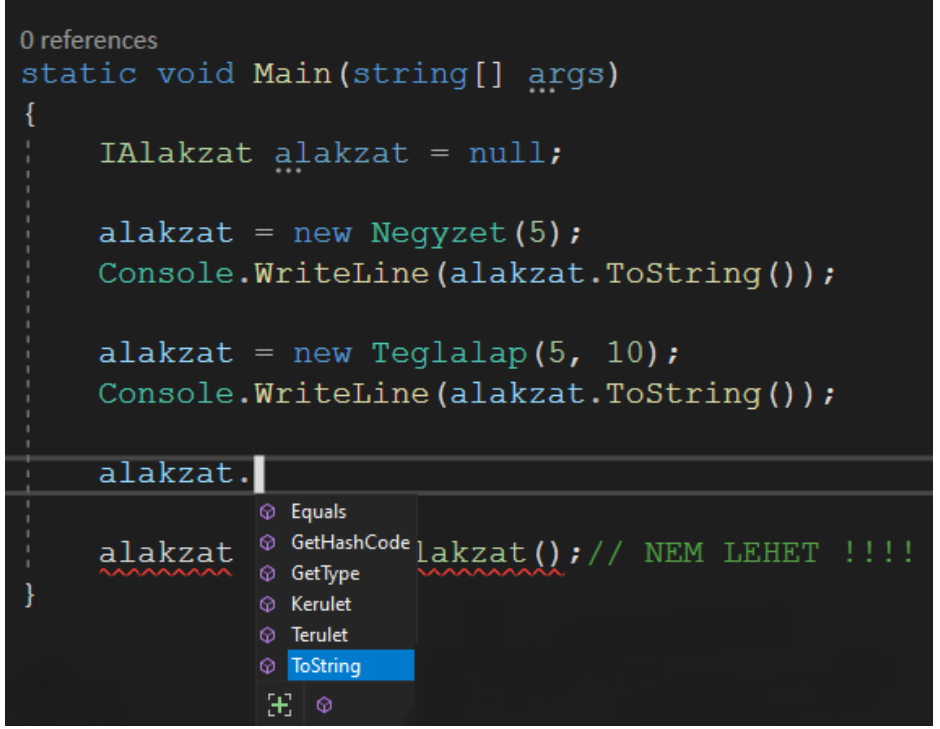
```
public class osztálynév : ŐsOsztaly, Interfész1, Interfész2,…
{
 // az osztály tagjai
}
```

```
public interface IAlakzat
{
    double Terulet();

    double Kerulet();
}
```

```
public class Negyzet : IAlakzat
{
    public double A { get; set; }

    public Negyzet()
    { }

    public Negyzet(double a)
    {
        A = a;
    }

    public double Kerulet()
    {
        return 4 * A;
    }

    public double Terulet()
    {
        return A * A;
    }

    public override string ToString()
    {
        return $"A {A} oldalu negyzet terulete {Terulet()}, kerulete:
{Kerulet()}.";
    }
}
```

```
public class Teglalap : IAlakzat
{
    public double A { get; set; }
    public double B { get; set; }

     public Teglalap()
     { }

     public Teglalap(double a, double b)
     {
         A = a;
         B = b;
     }

     public double Kerulet()
     {
        return 2 * (A + B);
     }

     public double Terulet()
     {
         return A * B;
     }

     public override string ToString()
     {
         return $"A {A} es {B} oldalu teglalap
terulete: {Terulet()}, kerulete: {Kerulet()}.";
     }
}
```

```
static void Main(string[] args)
{
    IAlakzat alakzat = null;

    alakzat = new Negyzet(5);
    Console.WriteLine(alakzat.ToString());

    alakzat = new Teglalap(5, 10);
    Console.WriteLine(alakzat.ToString());

    alakzat = new IAlakzat();// NEM LEHET !!!!
}
```



Látható, hogy a `Negyzet` és a `Teglalap` osztályok példányosítani tudják az `IAlakzat` interfészt, mivel megvalósítják azt. A példányosított objektum (alakzat), csak az intefész álatal definiált metódusokat tartalmazza, mivel csak őket ismeri, és nincs tudomása más, az interfészt példányosító osztályok adattagjairól vagy metódusairól (az interface és az osztály csak erre a két metódusra kötöttek szerződést: `Terulet és Kerulet [lásd a képet]`).

```
public class Kor
    {
        public double R { get; set; }

        public Kor()
        { }

        public Kor(double r)
        {
            R = r;
        }

        public double Kerulet()
        {
            return 2*R*Math.PI;
        }

        public double Terulet()
        {
            return R * R * Math.PI;
        }

        public override string ToString()
        {
            return $"A(z) {R} sugaru kor terulete
{Terulet()}, kerulete: {Kerulet()}.";
        }
    }
```

```
static void Main(string[] args)
{
 IAlakzat alakzat = new Kor(5); //NEM LEHET !!!
}
```

Vajon a fent leírt kód hibát jelezne-e ?

Mindannak ellenére, hogy a mi `Kor` osztályunk-nak van *Kerulet*, illetve *Terulet* függvénye, mégis hibát fog jelezni a fordító, mert a `Kor` osztály „nem kötött szerződést” az `IAlakzat` interfésszel. Látható, hogy a `Kor` osztály nem valósítja meg *(implementálja) az* `IAlakzat` interfészt.

Az interfészek első látásra az absztrakt osztályokhoz nagyon hasonlónak tűnhetnek. Ha egy osztály absztraktként van megjelölve, akkor lehet, hogy rendelkezik tetszőleges számú absztrakt metódussal, hogy ezen metódusok implementálását előírja a lezármazott osztályok számára. Amellett, hogy az absztrakt osztály absztrakt metódust, vagy metódusokat tartalmaz, lehetősége van konstruktorok, változók, nem absztrakt tagok definiálására is.

Az interfészek felhasználása abban az esetben kerül előtérbe, ha egy adott funkció ellátását több, akár egymástól teljesen eltérő felépítésű osztálytól is elvárjuk. Ennek egyszerű megoldása egy absztrakt ősosztály volna, amiből a többi osztály örökölné a megfelelő funkciót. Az öröklés nem a legmegfelelőbb megoldás, hiszen, ha több funkciót is el kellene látnia az adott osztályoknak, akkor előfordulna olyan eset, amikor több ősosztályból kellene örökölniük. Ez a megoldás .NET környezetben és Java-ban sem lenne kielégítő, hiszen egyik sem támogatja a többszörös öröklést.

Az interfészek használatakor néhány fontos alapelvet be kell tartanunk, melyek a következők:

- Az interfésznek nem lehet konstruktora, sem destruktora.
- Az interfésznek nem lehetnek mezői.
- Hozzáférés módosítókat nem használhatunk, hallgatólagosan minden metódusa publikus.
- Ha egy interfész ősének egy interfészt teszünk meg, akkor később az interfészből létrehozott ősztályoknál az ősinterfésznek is meg kell valósítani a tagjait.

A .NET Framework osztálykönyvtára néhány fontos interfészt tartalmaz, a legfontosabb az *IComparable*, az *IComponent*, az *IDisposable* és az *IEnumerator* interfész.

Azok az osztályok, amelyek implementálják az *IComparable* interfészt, megvalósítják a *CompareTo()* metódust, amely összehasonlítja a metódust hívó objektumot a paraméterként kapott objektummal.

Az *IComponent* interfészt minden olyan osztály megvalósítja, amely komponenst reprezentál.

Az *IDisposable* interfészt azok az osztályok valósítják meg, amelyeknek eroforrások felszabadítását kell biztosítaniuk.

Az *IEnumerator* interfész megvalósításával egy gyüjtemény elemein haladhatunk végig.

FORRÁS:

- https://felsofokon.hu/informatika/net-interface-1-resz-interface-ek-bemutatasa/
- Reiter István - c# jegyzetek
- Karsa Zoltán - c# Programozás
- Achs Ágnes–Szendroi Etelka - Az objektumorientált paradigma alapjai