# Interfész

## Objektumorientált programozás C# alapokon tananyag

Krizsán Zoltán[1]

Általános Informatikai Tanszék
Miskolci Egyetem

2012. május 9.

# Tartalom

- Egy osztály interfészén a nyilvános elemeinek összességét értjük, ami a használatához szükséges.
- Az információ rejtés elve miatt általában csak metódusszerű elemekből áll.
- Specifikáció, tervezés során készül el, hamarabb, mint a kód.
- Kényszeríti az osztály készítőjét a megfelelő elemek implementálására.
- "Protokollok" (a kommunikáció szabályai) az objektumok között.

# Interfészek C#-ban I

- Szintaktikája hasonló az osztályhoz, de a *class* kulcsszó helyett *interface*.
- Kiterjeszthet más interfészeket (":"), létezik többszörös interfész öröklés .
- Az interfész nem tartalmaz végrehajtható kódot, azt a megvalósító osztályban (":") kell megadni.
- Megállapodás, hogy "I"-vel kezdődik a neve, pl.: IVizsgalat.
- Az elemek aláírásait tartalmazza.

# Interfész részei I

Minden elem automatikusan **public** és **abstract**

- Methods
- Properties
- Indexers
- Events

# Szabályok I

- Nem példányosítható, hiszen nem tartalmaz definíciókat.
- Lehet referencia statikus típusa.
- Az implementáló osztály lehet egy ilyen referencia dinamikus típusa.

# Interfész szintaktika I

```
1  [modositok] interface IAzonosito [:
       osInterface[, osInterface2]]
2  {
3  // elemek deklaracioi
4  }
```

# Interface kiterjesztése I

- Minden interfész kiterjeszthet egy vagy több interfészt. Ha több interface-ból származik, akkor vesszővel választjuk el.
- Fontos, hogy önmagát nem terjesztheti ki sem közvetve, sem közvetlen. Nem alakulhat ki körkörös lánc.

# Absztrakt osztály <-> interfész I

- Vannak hasonlóságok és különbségek.
- Az interface tisztán osztály vázat definiál, az absztrakt osztály tartalmazhat implementációt is.
- Nem lehet példányosítani objektumot egyikből sem.
- Mindkettő típust definiál.

# Típus operátorok I

- az interfész használata valamilyen típus konverzió eredménye, mivel nem lehet példányosítani.
  - implicit típuskényszerítés
    - is operátor: logikai eredmény, vajon a bal oldal jobb oldali típusú e. pl.: if (referencia is IValami)
    - as operátor: rákényszeríti a bal oldali referenciára a jobb oldali típust: IValami ref = ref2 as IValami. (ha nem sikerül nincs kivét, null lesz)
  - explicit típuskényszerítés: castolás *(típus)referencia* (ha nem lehetséges kivétel keletkezik)

# is operátor I

- Logikai művelet. Az a is B művelet igaz, ha az a változó dinamikus típusa leszármazottja a B típusnak, egyébként hamis.
- Akkor igaz tehát, ha az a változó dinamikus típusát meghatározó osztály
  - maga a B osztály
  - vagy annak leszármazottja
  - vagy implementálja a B interface-t.

# Tartalom

# Új osztály szintaktika I

**Osztály őse, implementált interfészek a ":" után, de első helyen az osztály áll, hiszen belőle csak 1 van.**

```
1  [modosito] class osztalynev [:
2  Ososztaly] [, Iint1[,Iint2...[,]]]
3  {
4  [Elemek deklaracioi]
5  }
```

# Implementálásai szabályok I

- **Ha egy osztály implementál egy interfészt, akkor köteles annak minden metódusát implementálni!**
- Az implementált elemeket nem módosíthatja.
- Metódusok esetében a fejlécnek teljesen egyeznie kell!

# Tulajdonságok interfészben I

```
 1  interface IEmployee
 2  {
 3      string Name
 4      {
 5          get;
 6          set;
 7      }
 8
 9      int Counter
10      {
11          get;
12      }
13  }
```

# Interfész használata I

- Egy interfész új referencia típust vezet be => mindenhol használható, ahol egy osztály.
- Változó deklarációban szerepelhet (megadhatja annak statikus típusát).
- Bármelyik osztállyal helyettesíthető, amely implementálja (megadva ezzel a dinamikus típust).

# Tartalom

# Ugyanaz az elem több interfészben I

- Egy osztály implementál több interfészt is
  - Ugyanaz az elem fordul
  - különböző interfészekben.
- Ilyenkor két helyes megoldás lehet:
  - A két más interfészben levő elemnek ugyanaz az implementációja => egy metódus
  - Más implementáció tartozik a két elemhez (ez valószínűbb). => több metódus, amely teljes neve IInterface.metodusnev lesz.
- **Explicit implementált interfész esetén a metódust csak az interfészen keresztül lehet meghívni!**

# Két interfész egy implementáció I

```
 1  interface IControl
 2  {
 3      void Paint();
 4  }
 5  interface ISurface
 6  {
 7      void Paint();
 8  }
 9  class SampleClass : IControl, ISurface
10  {
11      // Both ISurface.Paint and IControl.Paint
            call this method.
12      public void Paint()
13      {
14      }
15  }
```

# Két interfész két implementáció I

```
1  public class SampleClass : IControl, ISurface
2  {
3      void IControl.Paint()
4      {
5          System.Console.WriteLine("IControl.Paint");
6      }
7      void ISurface.Paint()
8      {
9          System.Console.WriteLine("ISurface.Paint");
10     }
11 }
```

# Két interfész egy osztályban, használata I

```
1  SampleClass obj = new SampleClass();
2  //obj.Paint();  // Compiler error.
3
4  IControl c = (IControl)obj;
5  c.Paint();  // Calls IControl.Paint on
       SampleClass.
6
7  ISurface s = (ISurface)obj;
8  s.Paint(); // Calls ISurface.Paint on
       SampleClass.
```

# Tartalom

# Hozzunk létre egy interfészt I

Hozzuk létre az interfészt, ha szükséges örököltessük más(ok)ból!

```
1  interface ITest{
2      bool JoE { get; }
3  };
```

# Hozzunk létre egy osztályt I

Hozzuk létre az osztályt, mely tesztelhető.

```
1  class SulyzoRud : ITest{
2      public int Hossz { get; set; }
3      public int Suly { get; set; }
4      public SulyzoRud(int suly, int hossz)
5      {
6          Hossz = hossz;
7          Suly = suly;
8      }
9
10     public bool  JoE
11     {
12             get { return (Hossz > 180) && (Suly
                   > 15); }
13     }
14 }
```

# Hozzunk létre másik osztályt I

Hozzuk létre az osztályt, mely tesztelhető. Hogy mi a jó, azt itt adjuk meg

```
1  class Sulyzotarcsa : ITest{
2      public int Suly { get; set; }
3
4      public Sulyzotarcsa(int suly)
5      {
6          Suly = suly;
7      }
8      public bool  JoE
9      {
10             get { return (Suly > 10); }
11     }
12 }
```

# Használjuk az interfészt, mint elemet I

Hozzuk létre az osztályt, mely használja az interfészt!

```
1  class Tarolo
2  {
3      private ArrayList _munkadarabok = new
           ArrayList();
4
5      public void UjDarab(ITest ujdb)
6      {
7          _munkadarabok.Add(ujdb);
8      }
```

# Használjuk az interfészt, mint elemet II

```
1       public void KiirSelejtSorszam(){
2           // univerzalis kod, barmely ITest-et
3           // implementalo osztalyra mukodik
4           for(int i=0; i < _munkadarabok.Count;
                i++){
5               if( !((ITest)_munkadarabok[i]).JoE)
6               {
7                   Console.WriteLine(i);
8               }
9           } // for vege
10      } // metodus vege
11  }// class tarolo vege
```

# Használjuk az osztályokat, mint interfészeket I

Hozzuk létre az osztályt, mely használja az interfészt!

```
1                   Tarolo tar = new Tarolo();
2
3                   SulyzoRud rud = new SulyzoRud(12,
                        120);
4                   tar.UjDarab(rud);
5                   SulyzoRud rud2 = new SulyzoRud(15,
                        190);
6                   tar.UjDarab(rud2);
7
8                   Sulyzotarcsa tarcsa = new
                        Sulyzotarcsa(12);
9                   tar.UjDarab(tarcsa);
10
11                  tar.KiirSelejtSorszam();
```