Szoftvertechnológia 10. gyakorlat

Dr. Johanyák Zsolt Csaba
http://johanyak.hu

1. Adapter tervezési minta

Az adapter tervezési minta megoldást nyújt arra az esetre, amikor van egy osztályunk, ami alkalmas egy bizonyos feladat elvégzésére (vagy adattárolásra), de az általa nyújtott interfész eltér attól, amit a szolgáltatást igénybevevő osztály (ügyfél) elvár. Az adapter minta konvertálja az osztály interfészét az ügyfél által elvárt alakra.

Tehát a cél egy olyan új osztály létrehozása, ami a korábbi osztály felhasználásával megvalósít egy elvárt új interfészt.

Vegyünk egy egyszerű példát, ahol az alap osztály (Személy) két nyilvános tulajdonsággal rendelkezik.

```
public class Személy
{
   public Személy(string V, string U)
   {
      Vezetéknév = V;
      Utónév = U;
   }

   public string Vezetéknév { get; set; }
   public string Utónév { get; set; }
}
```

Tegyük fel, hogy az ügyfél osztály az INév interfész megvalósítását igényli, ami a fentiek mellett a teljes név lekérdezhetőségét is tartalmazza.

```
interface INév
{
  string Vezetéknév { get; set; }
  string Utónév { get; set; }
  string TeljesNév { get; }
}
```

Az Ügyfél osztály a következő

```
   class Ügyfél
   {
      private INév n;
      public Ügyfél(INév újNév)
      {
        n = újNév;
      }

      public void Kiír()
      {
        Console.WriteLine(n.Vezetéknév + "\t" + n.Utónév + "\t" + "\t" + n.TeljesNév);
```

```
    }
}
```

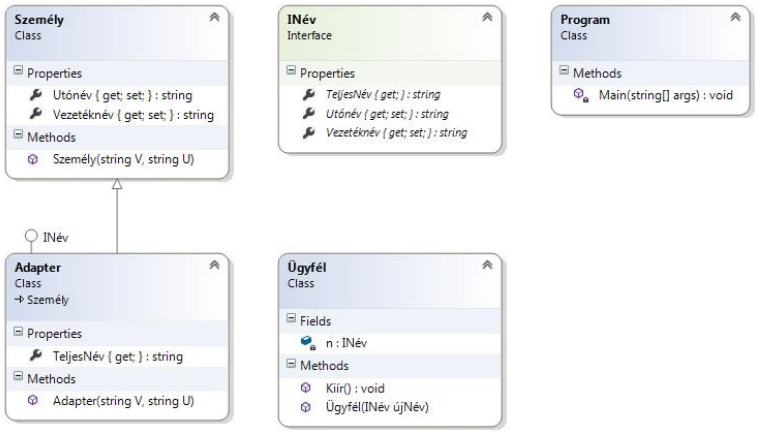
Az Ügyfél osztályt használó Program osztály az alábbi

```
class Program
{
    static void Main(string[] args)
    {
      Adapter a = new Adapter("Nagyon", "Ügyes");
      //...
      Ügyfél ü = new Ügyfél(a);
      ü.Kiír();
      Console.ReadLine();
    }
}
```

Az illesztést megoldó Adapter osztály a Személy osztály leszármazottja lesz, és megvalósítja az INév interfészt.

```
class Adapter : Személy, INév
{
    public Adapter(string V, string U) : base(V, U) { }

    public string TeljesNév
    {
      get
      {
            return Vezetéknév + " " + Utónév;
      }
    }
}
```



2. További megoldási lehetőségek

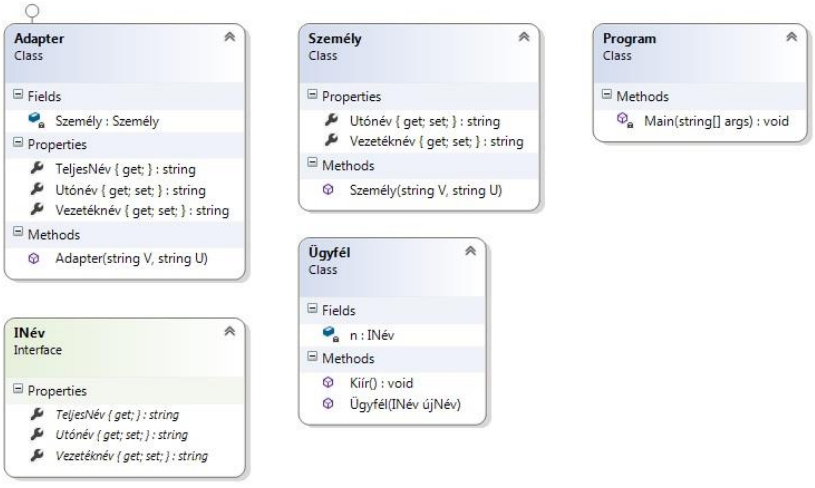
Nézzünk meg két további megoldási módot, ahol nem használunk öröklődést (pl. az alap osztály **sealed** jelzővel van ellátva)

- Beágyazás
- Kiegészítés

Mindegyik módszerhez egy külön projektet készítünk a megoldáson (solution) belül.

2.1. Beágyazás

```
public class Adapter : INév
{
    Személy Személy;
    public string TeljesNév
    {
      get { return Személy.Vezetéknév + " " + Személy.Utónév; }
    }
    public string Vezetéknév
    {
      get { return Személy.Vezetéknév; }
      set { Személy.Vezetéknév = value; }
    }
    public string Utónév
    {
      get { return Személy.Utónév; }
      set { Személy.Utónév = value; }
    }
    public Adapter(string V, string U)
    {
      Személy = new Személy(V, U);
    }
}
```



2.2. Kiegészítés

Ez a megoldás akkor alkalmazható, ha az eredeti osztály rendelkezik "partial" jelzővel. Pl. az Entity Framework használatánál ezt a megoldást alkalmaztuk Vizuális programozás gyakorlaton.

```
public partial class Személy : INév
{

   public string TeljesNév
   {
     get { return Vezetéknév + " " + Utónév; }
   }

}
```

Ebben az esetben a Main metódus is változik, mivel nem Adapter nevű osztályt használunk.

```
   static void Main(string[] args)
   {
      Személy a = new Személy("Nagyon", "Ügyes");
      //...
      Ügyfél ü = new Ügyfél(a);
      ü.Kiír();
      Console.ReadLine();
   }
```