Adatkötés NEM UI elemhez

FONTOS! Az adatkötés alapértelmezett forrása mindig a **DataContext**-ben lévő objektum! Ezt eddig mindig felülírtuk az **ElementName**-mel egy konkrét vezérlőre. A **DataContext** függőségi tulajdonság, tehát értéke a tartalmazóktól örökölhető és így lehetőségünk van ablak, tartalommenedzser vagy vezérlő szintjén is beállítani. Ekkor minden adatkötési **Path** alapértelmezetten erre az objektumra mutat, így **NEM** is kell megadni a **PATH** értékét!

Például van egy Person osztályunk:

```
public class Person
{
    public string Name { get; set; } = string.Empty;
    public int Age { get; set; }
}
```

Beállítjuk a **Window** konstruktorában, hogy a **DataContext** ez a **Person** osztály legyen.

```
public MainWindow()
{
    InitializeComponent();
    this.DataContext = new Person()
    {
        Name = "Dezső",
        Age = 25
    };
}
```

Ekkor a XAML-ban a következő adatkötés kiírja a nevet:

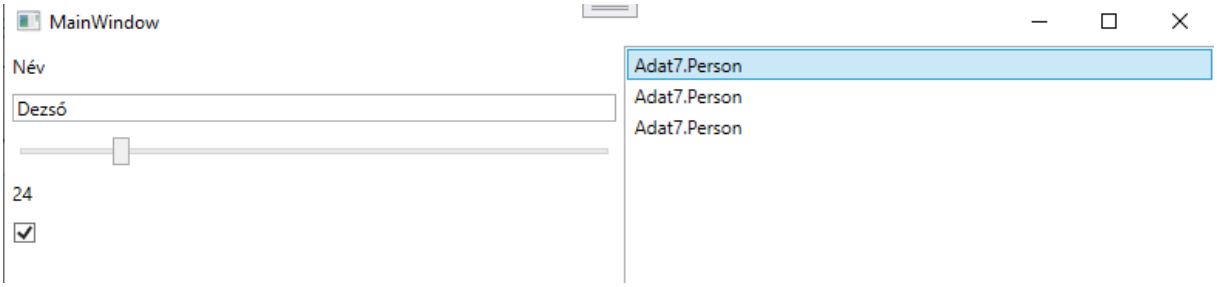
```
<TextBlock Text="{Binding Name}"/>
```

FONTOS! A **DataContext** az ablakhoz van kötve, de ezt a kötést lefelé **ÖRÖKLI** minden tartalmazott eleme!

A **DataContext** is kaphatja az értékét adatkötésből!
• Pl: egy **StackPanel**-ben lévő űrlapelem **DataContext**-je egy **ListBox**-ban kijelölt elem megfelelő paramétere!

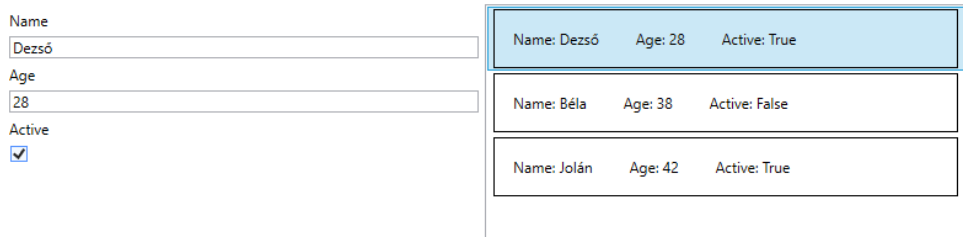
```
<ListBox Grid.Column="1" Name="lbox">
    <local:Person Name="Dezső" Age="28" />
    <local:Person Name="Rozál" Age="48" />
</ListBox>

<StackPanel DataContext="{Binding ElementName=lbox, Path=SelectedItem}">
    <Label Content="Név" Background="LightBlue" Padding="10" />
    <TextBox Text="{Binding Name}" Padding="10"/>
</StackPanel>
```

Így is kinézhet:

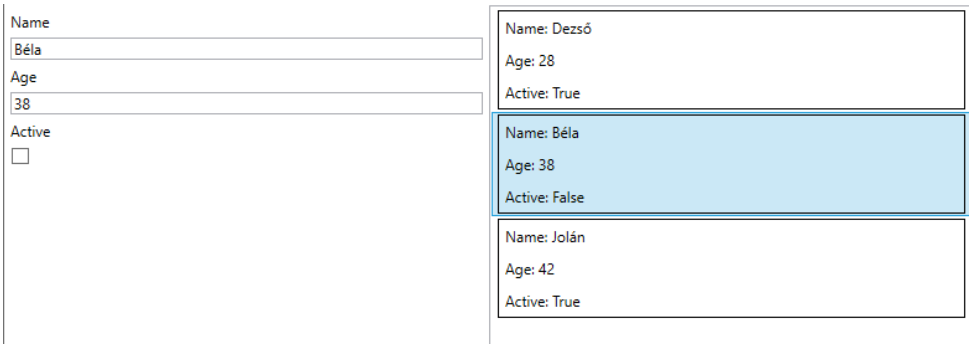


Feladat: csináljunk egy apró alkalmazását, ahol a jobb és a bal oldal adatkötéssel össze van kötve, a jobb oldalon lévő ListBox alapértelmezett értékeit a bal oldalon lévő vezérlők segítségével átírhatjuk.

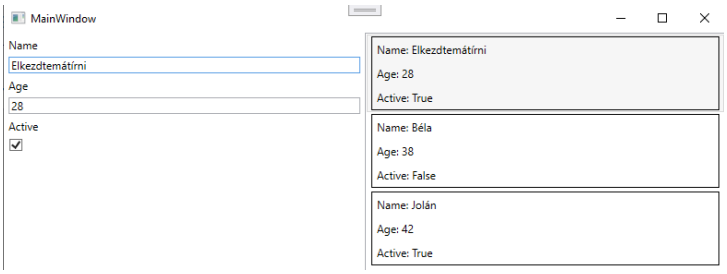
Ilyen (`<StackPanel Orientation="Horizontal">`)



Vagy ilyen kellene:



Ha jól csináltuk, akkor a TextBoxba írt szöveg megjelenik a ListBoxban. Így:



Haladjunk lépésenként.
Először osszuk két részre a Grid-et.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="1*"/>
    </Grid.ColumnDefinitions>
```

Utána a baloldaliban hozzuk létre a megfelelő vezérlőket az adatkötésekkel együtt. Figyeljünk arra, hogy a lehető legtöbb dolgot a **XAML** segítségével állítsuk be, tehát itt adom meg, hogy a **StackPanel** a **ListBox**-tól kapja majd az adatokat adatkötéssel, tehát a **ListBox** lesz a forrás.

```
<StackPanel DataContext="{Binding ElementName=lb, Path=SelectedItem}">
     <TextBlock Text="Name" Margin="5,5,5,0"/>
     <TextBox Text="{Binding Name, UpdateSourceTrigger=PropertyChanged}"
Margin="5,5,5,0"/>
```

A **ListBox** létrehozása egy kicsit bonyolultabb lesz, mivel itt nem a teljes **ListBox**-ot akarjuk definiálni, hanem elemenként állítjuk össze, tehát azt határozzuk meg, hogy nézzen ki egy eleme. Ezt az **ItemTemplate**, **DataTemplate** segítségével tudjuk megadni. A **DataTemplate** tartalmazza majd 1 db **ListBox** elem felépítését.
A `<Label ContentStringFormat="Name: {0}" Content="{Binding Name}" />` sorban a `ContentStringFormat` arra szolgál, hogy statikus értéket tudjunk beszurni a tulajdonság elé.

```
<ListBox x:Name="lb" Grid.Column="1" HorizontalContentAlignment="Stretch" >
     <ListBox.ItemTemplate>
         <DataTemplate>
             <Border BorderThickness="1" BorderBrush="Black">
                 <StackPanel>
                     <Label ContentStringFormat="Name: {0}" Content="{Binding Name}" />
```

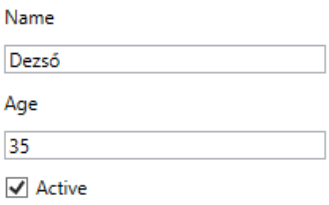
+

A wpf03.pdf

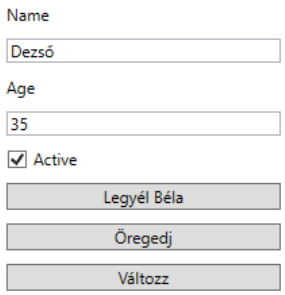
és ezzel gyakorlatilag kész a kis programunk. Minden UI érték egyből frissül, mert adatkötéssel kap értéket.

Mi történik akkor, ha nem adatkötésből kapja az értéket? Akkor sajnos nekünk kell gondoskodnunk a UI elem frissítéséről. Ezt többféle módon végrehajthatjuk pl. az **InotifyPropertyChanged** interfész segítségével.

Kezdjük egy egyszerű feladattal. Adott egy Person osztály Name, Age, Active propertykkel. Létrehozunk egy objektumot ebből az osztályból (példányosítunk) és ennek az értékeit megjelenítjük egy ablakban valahogy így:



Majd ideteszünk pár gombot. Ha az elsőre kattintunk, akkor a név megváltozik Bélára. Ahányszor a másodikra kattintunk, annyiszor 1 évvel öregedik a jóember. Ha a harmadikra kattintunk, akkor ellenkezőjére változik az Active értéke. Legyen ilyen és kattintsunk:



Azt látjuk, hogy hiába kattintunk, csak a forrás változik a kiírás nem!

Első megoldás (parasztos, NE HASZNÁLJUK!) Működik ugyan, de sok vezérlő esetén meghal az egész!

```
private void Button_Click(object sender, RoutedEventArgs e)
 {
     p.Name = "Béla";
     this.DataContext = null;
     this.DataContext = p;
 }
```

A második megoldás a legáltalánosabb, kezdőként ezt használjuk szinte mindig. Ez pedig az `InotifyPropertyChanged` intefész használata a Person osztályban. Így:

```
public class Person : InotifyPropertyChanged
```

Ez az interfész egyetlen **eseményt** tartalmaz, amit úgy érdemes elképzelnünk, mintha egy szignál lenne, ami szól a UI-nak, hogy valami változás történt, változz te is! A feladatunk az lesz, hogy kiváltsuk ezt az **eseményt**, amikor történik valamilyen változás a forrásban. Az esemény maga:

```
public event PropertyChangedEventHandler? PropertyChanged;
```

Az **esemény** kiváltását a **property** *setterében* tudjuk megadni! Tehát a ***propfull*** *snipetet* érdemes használni, hogy egy ilyet kapjunk (bár van ennél modernebb jelölés):

```
private int name;

public int Name
{
    get { return name; }
    set { name = value; }
}
```

A `PropertyChanged` eseményt mindig meg kell hívni és ugyanazt a 3-4 sort kell leírni, egyszerűbb függvénybe kiszervezni a dolgot. Itt is van pár lehetőség a függvényekre. Először nézzük a régebbit.

```
public void OnPropertyChanged(string PropName)
{
    PropertyChangedEventHandler? handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(PropName));
    }
}
```

És innen már csak annyi dolgunk van, hogy a setterben beállítjuk, ha változás történt, akkor hívja meg ezt a függvényt (`OnPropertyChanged`). Így:

```
private string name =string.Empty;
public string Name
{
    get { return name; }
    set {
        name = value;
        OnPropertyChanged("Name");
    }
}
```

És kész, ettől kezdve ha változik a név, meghívódik a függvény, a szignál esemény kiváltódik, szól a felhasználói felületnek és ott is megváltozik az érték.

Persze túl hosszú volt a függvény, így lerövidítették erre, itt még a meghívás a régi:

```
protected void OnPropertyChanged(string PropName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropName));
}
```

Az **Invoke** adja a szignált, meghív minden kezelőt, aki feliratkozott az eseményre. A **WPF binding** rendszer meg mindig **feliratkozik erre az eseményre**. Ettől kezdve olyan, mintha adatkötésből kapná az értéket, az meg mindig frissül a felhasználói felületen!

De itt még a setterben meg kell adni a property nevét, amit paraméterként vesz át a fv. Elgépelhetjük, így a legmodernebb változat a következő (ebben már nem kell nevet megadni, mert a [CallerMemberName] elintézi, hogyha nem adsz át paramétert, akkor automatikusan annak a property-nek a nevét adja át, ahonnan hívták)

```
protected void OnPropertyChanged([CallerMemberName] string PropName = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropName));
}
```

Ebben az esetben a függvény meghívása így néz ki:

```
public string Name
{
    get { return name; }
    set {
        name = value;
        OnPropertyChanged();
    }
}
```

Viszont ez így még mindig elég macerás, így kitaláltak rá egyszerűbb módszereket is. Mivel lassan fáradok elég lesz a legjobbat megmutatnom.
Kell hozzá egy NuGet csomag: ***CommunityToolkit.Mvvm***

Majd egy pár beállítás és már kész is az automatikusan legenerált kód!

Lépésenként:

- Telepítjük a `CommunityToolkit.Mvvm NuGet csomagot`
- használatba vesszük a névterét: `using CommunityToolkit.Mvvm.ComponentModel;`
- Az osztályt parciális osztályként hozzuk létre, hogy a NuGet csomagunk hozzá tudja generálni egy másik fájlban a szükséges kiegészítéseket! És az osztály mindent örököl a `ObservableObject` osztályból. Az `ObservableObject` a NuGet csomagban lakik és valójában ő valósítja meg az `InotifyPropertyChanged` intefészt!
  A kód: `public partial class Person : ObservableObject`
- A szükséges property-k, amelyek változásakor jelezni kell a UI-nak így néznek ki:

```
[ObservableProperty]
private string name = string.Empty;

[ObservableProperty]
private int age;
```

  stb.

Itt az `[ObservableProperty]` attribútum gondoskodik a megfelelő kódok legenerálásáról. Valójában a CommunityToolkit.Mvvm kód generátorát használja, hogy automatikusan property-t és OnPropertyChanged logikát generáljon. Tehát létrehozza és hozzáírja a parciális osztályunkhoz a `public string Name { get; set; }` property-t!

Innen folytatjuk!