

# WinUI alapú fejlesztés

< Albert István, [ialbert@aut.bme.hu](mailto:ialbert@aut.bme.hu) >



Automatizálási és  
Alkalmazott  
Informatikai Tanszék

# Tartalom

## WinUI

- Desktop alkalmazás fejlesztéséhez Windows platformra
- A Windows UI technológiája
  - > Nem az operációs rendszer része, külön csomag
- Natív kódban írt vezérlők



WinUI

## XAML alapok

```
<Border Background="DeepSkyBlue" HorizontalA...>
```



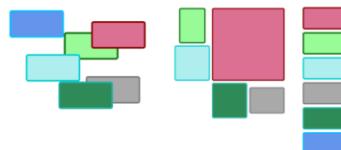
## Komponálhatóság

Project A Success Delete and archive

## ContentControl

## Dependency Property

```
<TextBlock Name="txt" Text="Cép" />
```

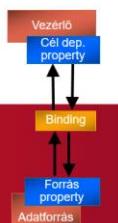


## Elrendezés

## Erőforrások



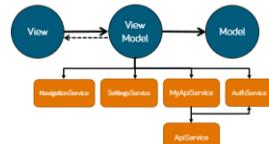
## Adatkötés



## Sablonok



## MVVM

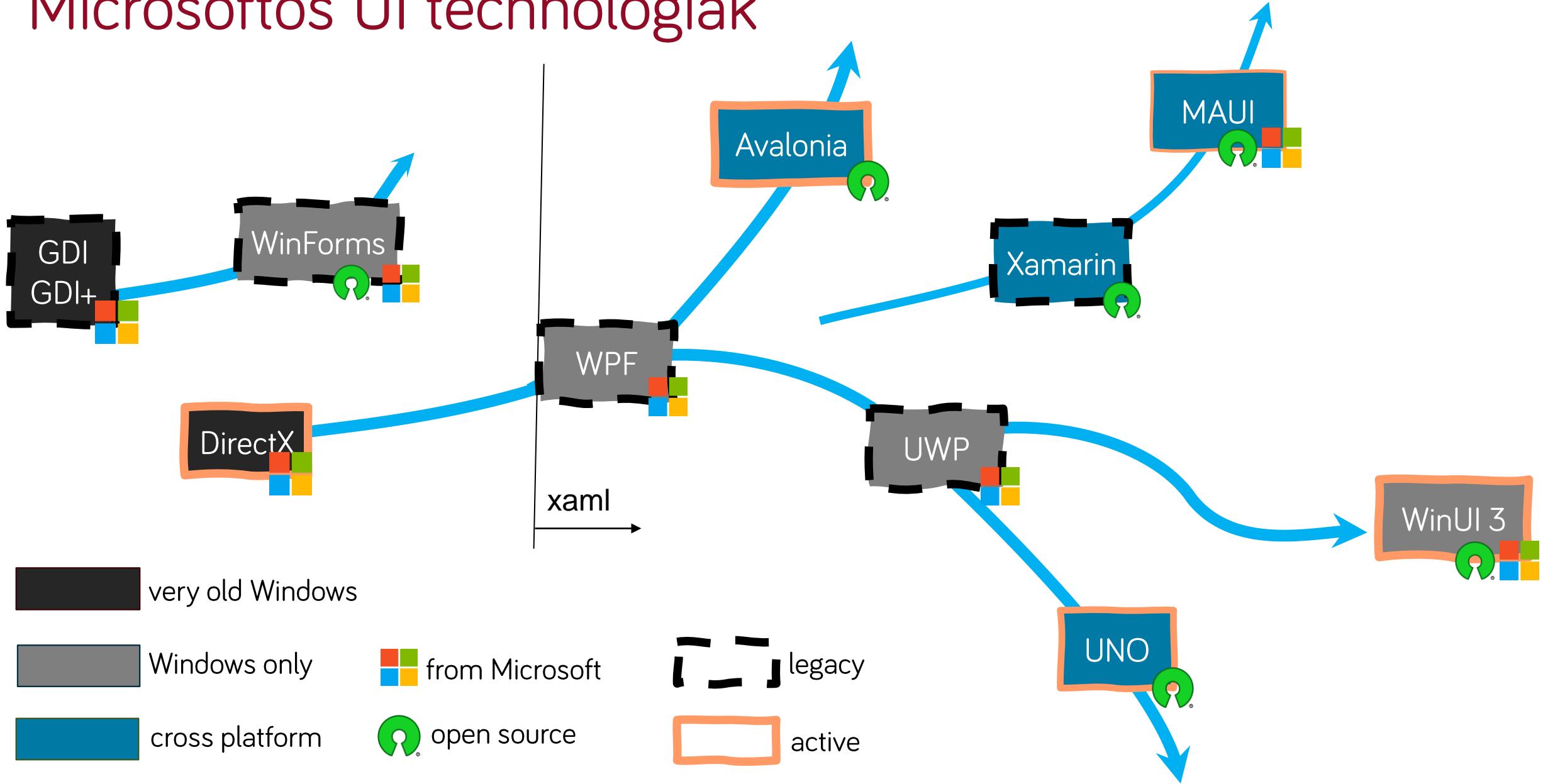


# WinUI

- Desktop alkalmazás fejlesztéséhez Windows platformra
- A Windows UI technológiája
  - > Nem az operációs rendszer része, külön csomag
- Natív kódban írt vezérlők



# Microsoftos UI technológiák



# Egyéb technológiák Windowson

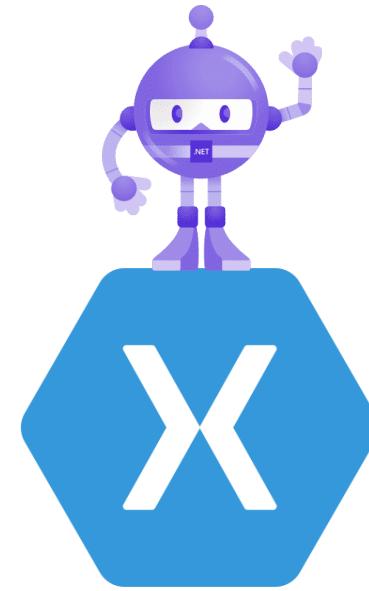
- 1983. GDI: C API
  - > Még DirectX előtt
  - > Ne közvetlenül írunk a videó memóriába
  - > Hardveresen gyorsított, képernyő és printer ☺
- 2001. GDI+: C++ API
  - > Objektum alapú API, CPU-val rajzolja a szöveget ☹
- ComCtl32.dll: ebben vannak implementálva a beépített vezérlők, például a nyomógomb stb.
- Ez mind elérhető .NET-ből Windows Forms-on keresztül!

# A „XAML” platformok

- 2006. WPF – C#, .NET, Windows, Win32
- 2012. UWP – C++, Windows, WinRT
- 2014. Xamarin Forms – Android, iOS
- 2022 MAUI – XAML (Xamarin dialektus)
  - > Xamarin -> Android, iOS, macOS, Windows
- 2022 WinUI – XAML (UWP dialektus)
  - > Win32-es és WinRT alkalmazások támogatása
  - > Windows + .NET

# MAUI - Microsoft

- Multi-platform keretrendszer
  - > iOS, macOS, Android, Windows
- Nyílt forráskódú
- A platform *natív vezérlőit* használja
- C#, F# ...
- XAML / code
- .NET ökoszisztéma
- Visual Studio / VS Code



```
<Grid RowSpacing="{v:Height 3}" RowDefinitions="3*, *" ColumnDefinitions="* * *">  
    <Label  
        Text="{Binding TimeText}"  
        Style="{StaticResource TimerLabel}"  
        VerticalTextAlignment="Center"  
        HorizontalTextAlignment="Center"  
        local:LabelAutoFit.IsAutoFit="true"  
        Grid.Row="0"  
        Grid.ColumnSpan = "3"  
    />
```

# Avalonia <- WPF



- Multi-platform keretrendszer
  - > iOS, macOS, Linux, Android, Windows, Web, WebAssembly
- WPF-szerű XAML dialektus
- Skia/Direct2D – minden platformon ugyanúgy néz ki
- Nyílt forráskódú
  - > XPF nem
- C#, F#, XAML
- Visual Studio / Rider
- .NET ökoszisztéma
  - > Core / Mono / .NET Framework

```
<Window xmlns="https://github.com/avaloniaui"  
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">  
    <Window.Styles>  
        <Style Selector="TextBlock.h1">  
            <Setter Property="FontSize" Value="24"/>  
            <Setter Property="FontWeight" Value="Bold"/>  
        </Style>  
    </Window.Styles>  
  
    <TextBlock Classes="h1">I'm a Heading!</TextBlock>  
</Window>
```

# UNO <- UWP



- Multi-platform keretrendszer
  - > iOS, macOS, Linux, Android, Windows, HTML, WebAssembly
- UWP-szerű XAML dialektus
- Nyílt forráskódú
- Skia - minden platformon ugyanúgy néz ki
- Natív renderer – natív vezérlők is
- C#, F#, XAML / kód
- .NET ökoszisztéma
- Visual Studio / VS Code / Rider

```
<TextBlock Text="Planning" FontWeight="Bold" FontSize="16" Grid.Row="3" Margin="10,0" />

<StackPanel Orientation="Horizontal" Grid.Row="4" Margin="10,0" Spacing="20">
    <StackPanel Background="LightGray" Padding="20">
        <TextBlock Text="Effort" FontWeight="Bold" FontSize="16" Margin="10,0" />
        <TextBox Text="{x:Bind Item.Effort,Mode=TwoWay}"
            HorizontalTextAlignment="Center"
            HorizontalAlignment="Center"
            HorizontalContentAlignment="Center"
            BorderBrush="Transparent"
            Background="Transparent"/>
    </StackPanel>
    <Slider Value="{x:Bind Item.Effort,Mode=TwoWay}" Width="100" Minimum="0" Maximum="15" />
</StackPanel>
```

# WinUI 3 - Microsoft



- Natív vezérlők Windows platformra
  - > Az operációs rendszertől független könyvtár
- Nyílt forráskódú
- Fluent Design System
- A Windowson futó alkalmazások natív felhasználói felülete
  - > UNO, React Native, MAUI is ezt használja
- C#, C++
- XAML / code

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">
    <Button x:Name="myButton" Click="myButton_Click">Display loaded modules</Button>

    <ContentDialog x:Name="contentDialog" CloseButtonText="Close">
        <ScrollViewer>
            <TextBlock x:Name="cdTextBlock" TextWrapping="Wrap" />
        </ScrollViewer>
    </ContentDialog>
</StackPanel>
```

# XAML alapok

```
<Border Background="#DeepSkyBlue" Height="100" Width="100">
|   <TextBlock Text="☀️" HorizontalAlignment="Center" VerticalAlignment="Center"/>
|   </TextBlock>
</Border>
```



# Mire jó a XAML?

- Deklaratív, objektum fa példányosító nyelv
- Objektumok
  - > Tipikusan vezérlők
  - > Vezérlő tulajdonságok
- Fa
  - > Jól követhető hierarchia
  - > XML formátum
- A vektor grafikus megjelenítés a rajzoló alrendszer feladata!

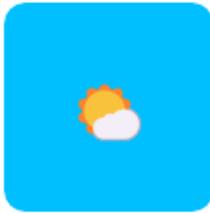
```
<Border Background="DeepSkyBlue" Height="100" Width="100">  
    | <TextBlock Text="Cloud" HorizontalAlignment="Center" VerticalAlignment="Center"/>  
</Border>
```



# Objektum példányosítás

- A **tegek** példányosítandó osztály nevek
- Az **attribútumok** tulajdonságok
- Vannak beépített konverziók: szám, enum, ...

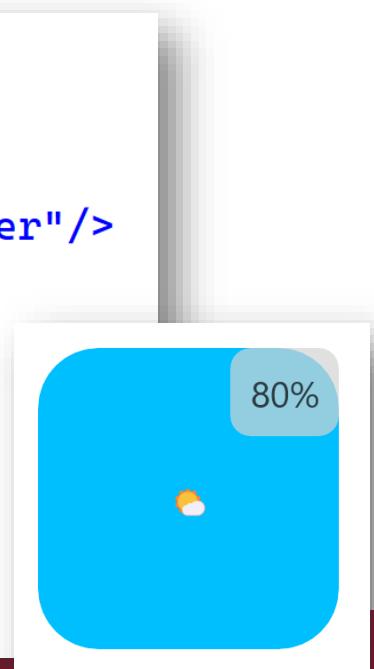
```
<Border Background="DeepSkyBlue"  
        Height="50" Width="50" CornerRadius="5">  
    <TextBlock Text="Cloud" HorizontalAlignment="Center" VerticalAlignment="Center"/>  
</Border>
```



# Hierarchia, elrendezés

- A Grid táblázatba rendezi az elemeket
  - > Itt egyetlen cellában van benne kettő, egymáson
- Állítható az elrendezés, térkihagyás
- Átlátszóság, színek

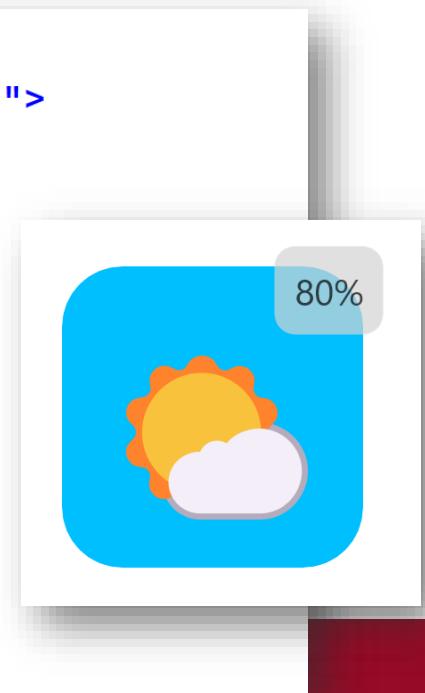
```
<Grid Height="150" Width="150">
    <Border Background="#DeepSkyBlue" CornerRadius="30" Padding="15">
        <TextBlock Text="Cloud"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Border>
    <Border Background="#LightGray" Opacity="0.7"
        CornerRadius="10" Padding="10"
        HorizontalAlignment="Right" VerticalAlignment="Top" >
        <TextBlock FontSize="18" Text="80%"/>
    </Border>
</Grid>
```



# Összetett tulajdonságok

- Az összetett tulajdonságok külön XML csomópontként adhatók meg  
`<osztály . tulajdonság>`  
`<összetett objektum />`
- **ViewBox:** vektorgrafikus felnagyítja a benne lévő elemet

```
<Grid Height="150" Width="150">
    <Border Background="#DeepSkyBlue" CornerRadius="30" Padding="15">
        <Viewbox>
            <TextBlock Text="Cloudy" />
        </Viewbox>
    </Border>
    <Border Background="#LightGray" Opacity="0.7" CornerRadius="10" BorderThickness="1">
        <Border.RenderTransform>
            <TranslateTransform X="10" Y="-10" />
        </Border.RenderTransform>
        <TextBlock FontSize="18" Text="80%"/>
    </Border>
</Grid>
```



# Névterek, hivatkozások

Kódot nem  
kell tudni

- A vezérlőkre névvel hivatkozhatunk
- Külső csomagokat névterekbe húzzuk be
- A XAML nézet össze van kötve a hozzá tartozó osztállyal: code behind

```
<Grid Height="150" Width="150">
    <Border x:Name="shadowTarget" Background="#DeepSkyBlue">
        <Viewbox>
            <TextBlock Text="Cloudy" />
        </Viewbox>
    </Border>
    <Border Background="#LightGray" Opacity="0.7" CornerRadius="10" HorizontalAlignment="Center" VerticalAlignment="Bottom">
        <Border.RenderTransform>
            <TranslateTransform X="10" Y="-10" />
        </Border.RenderTransform>
        <ui:Effects.Shadow>
            <ui:AttachedDropShadow CastTo="{x:Bind shadowTarget}" BlurRadius="80"/>
        </ui:Effects.Shadow>
        <TextBlock FontSize="18" Text="80%"/>
    </Border>
</Grid>
```

```
<Window
    x:Class="XAMLsample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:ui="using:CommunityToolkit.WinUI"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```



# XAML és kód ekvivalens

- XAML tegek és attribútumok

> XAML

```
<Button  
    Content="OK"  
    Click="Button_Click"/>
```

> C#

```
using Windows.UI.Xaml;  
  
using Windows.UI.Xaml.Controls;  
  
Button b = new Button();  
b.Content = "OK";  
b.Click += Button_Click;
```



# Markup kiterjesztések

- A tulajdonságok megadásánál használt '{' jel egy markup kiterjesztést aktivál

```
<TextBlock Text="{Binding}" ...>
```

- Ezek a kifejezések egymásba ágyazhatók
- Tipikusan speciális hivatkozásokat jelentenek
  - > Adatkötés
  - > Erőforrás hivatkozás
  - > Sablon hivatkozás

# UI és mögöttes kód (code-behind)

- minden XAML felülethez tartozik egy osztály
  - > Eseménykezelők
  - > Vezérlők tagváltozói
  - > Egyebek
- A vezérlőknek **x:Name**-mel adunk nevet, így hivatkozhatunk rá kódból
- Az eseménykezelőbe a **metódus nevét** írjuk
- Ritkán használjuk, lásd MVVM

# Példa code-behindra 1

```
<Window
    x:Class="CodeBehindSample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:CodeBehindSample"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Canvas>
        <Button x:Name="myButton" Click="myButton_Click">Catch me!</Button>
    </Canvas>
</Window>
```

```
public sealed partial class MainWindow : Window
{
    private void myButton_Click(object sender, RoutedEventArgs e)
    {
        tmr.Stop();
        myButton.Content = "You win!";
    }
}
```

# Példa code-behindra 2

- Vajon mit csinál a program?

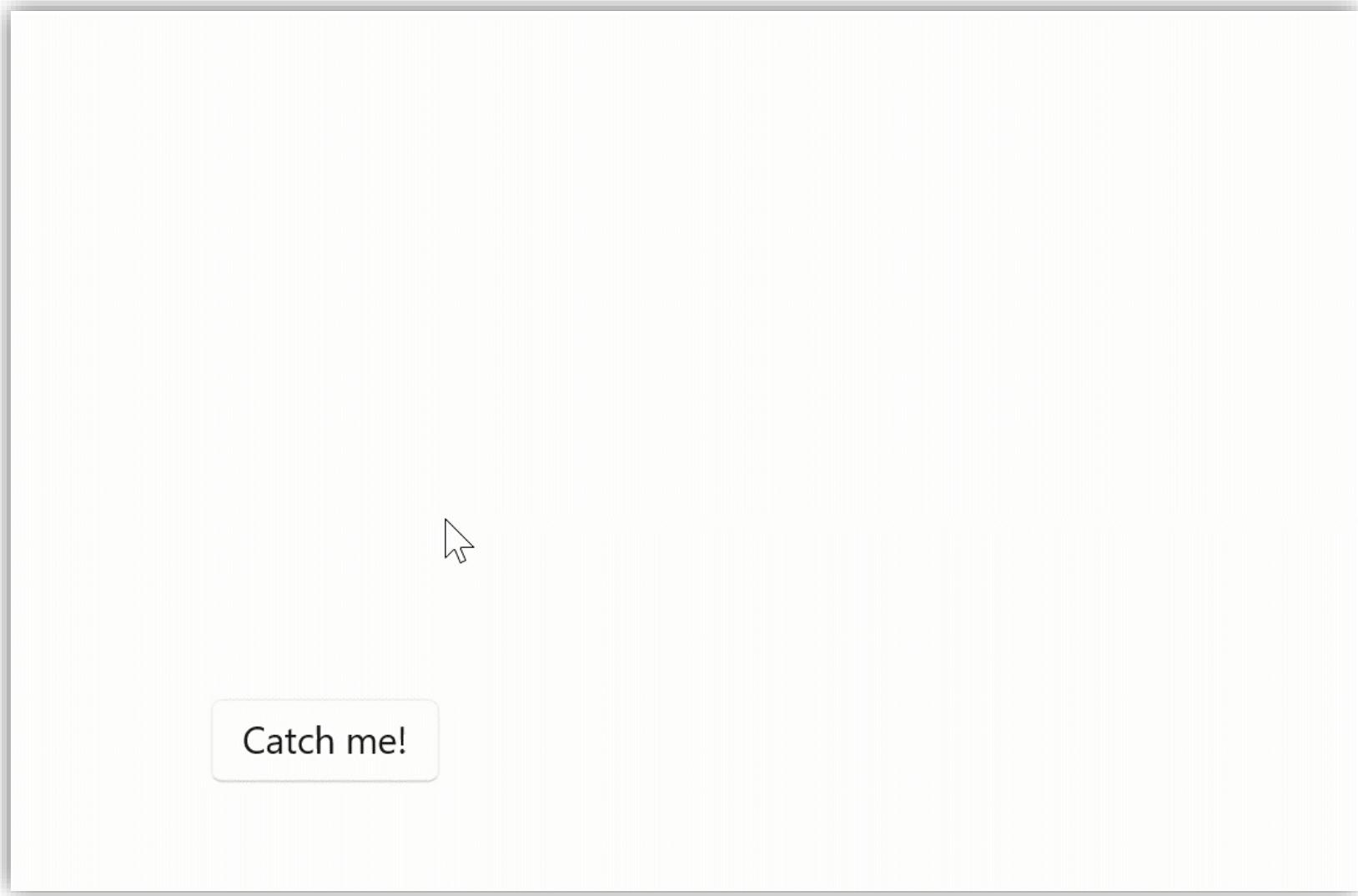
```
DispatcherTimer tmr = new DispatcherTimer();
Random random = new Random();

public MainWindow()
{
    this.InitializeComponent();
    tmr.Interval = TimeSpan.FromMilliseconds(600);
    tmr.Tick += Tmr_Tick;
    tmr.Start();
}

private void Tmr_Tick(object sender, object e)
{
    Canvas.SetLeft(myButton, random.Next(0, 400));
    Canvas.SetTop(myButton, random.Next(0, 300));
}

private void myButton_Click(object sender, RoutedEventArgs e)
{
    tmr.Stop();
    myButton.Content = "You win!";
}
```

# Demó



# Komponálhatóság

Project A   Success



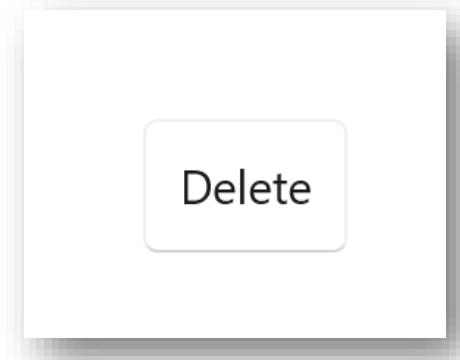
Delete



and archive

# Komponálhatóság

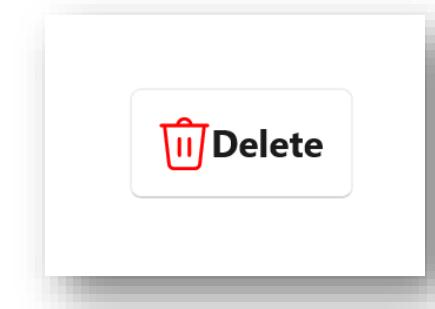
```
<Button Padding="10" HorizontalAlignment="Center">
    Delete
</Button>
```



Delete

# Komponálhatóság

```
<Button Padding="10" HorizontalAlignment="Center">
    <StackPanel Orientation="Horizontal">
        <SymbolIcon Symbol="Delete" Foreground="Red" />
        <TextBlock FontWeight="Bold" Text="Delete" />
    </StackPanel>
</Button>
```



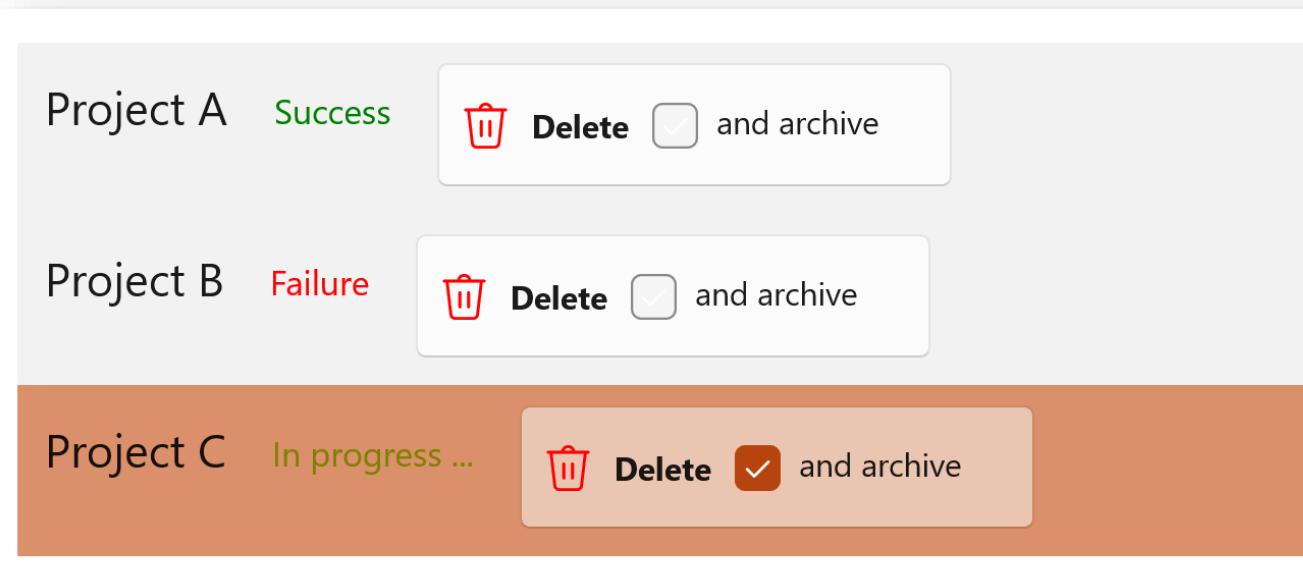
# Komponálhatóság

```
<Button Padding="10" HorizontalAlignment="Center">
    <StackPanel Orientation="Horizontal">
        <SymbolIcon Symbol="Delete" Foreground="Red" />
        <TextBlock Padding="10 0" VerticalAlignment="Center"
            FontWeight="Bold" Text="Delete" />
        <CheckBox Content="and archive" />
    </StackPanel>
</Button>
```



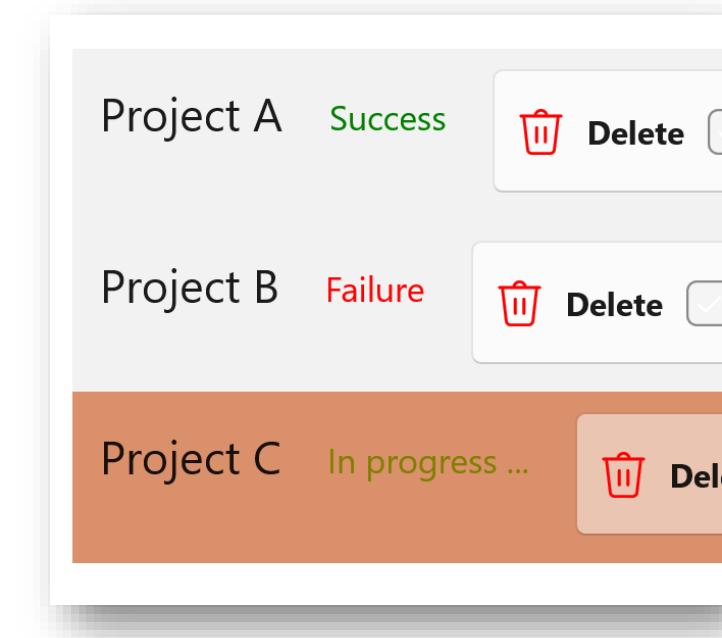
# Komponálhatóság

```
<ListBox Margin="20" HorizontalAlignment="Center" VerticalAlignment="Center">
    <ListBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock Padding="0 5 20 0" FontSize="20" Text="Project A" />
            <TextBlock Padding="0 10 20 0" FontSize="15" Foreground="Green" Text="Success" />
            <Button Padding="10" HorizontalAlignment="Center" ...>
        </StackPanel>
    </ListBoxItem>
    <ListBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock Padding="0 5 20 0" FontSize="20" Text="Project B" />
            <TextBlock Padding="0 10 20 0" FontSize="15" Foreground="Red" Text="Failure" />
            <Button Padding="10" HorizontalAlignment="Center" ...>
        </StackPanel>
    </ListBoxItem>
```



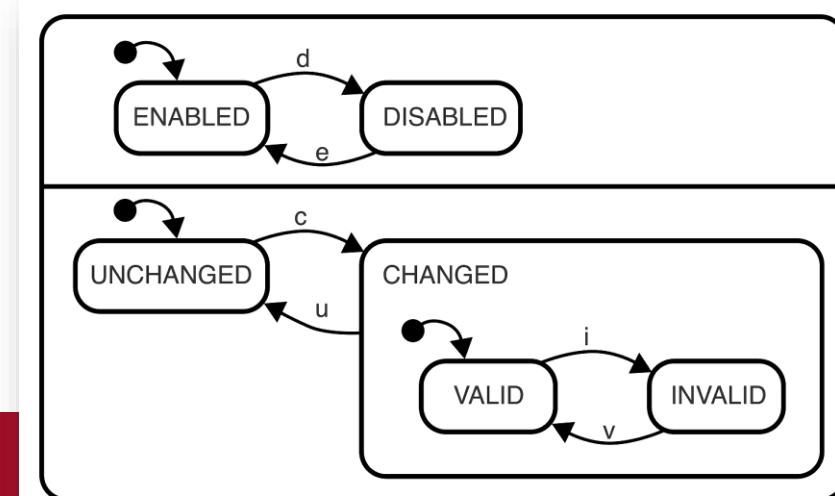
# Komponálhatóság

- A vezérlők **tetszőleges tartalmat** meg tudnak jeleníteni
  - > Például a Button egy „Content”-et
  - > Content bármi lehet: szöveg, kép, **más vezérlők**
    - StackPanel + szöveg + ...
- Az összetett vezérlők más vezérlőkből állnak
  - > Például: ListView
    - ScrollView a görgetéshez
    - minden listaelem egy ListViewItem
    - A ListViewItem tetszőleges Contentet tud megjeleníteni



# Testreszabható megjelenés

- Deklaratív an, grafikus objektumokkal írható le az egyes vezérlők megjelenése
  - > Vonal, háttér stb.
- Leírhatók a különböző állapotokhoz tartozó változások
  - > Például letiltott vezérlő
- Az állapotok eseményekhez köthetők
  - > Például megnyomják a gombot
- A változások animálhatók



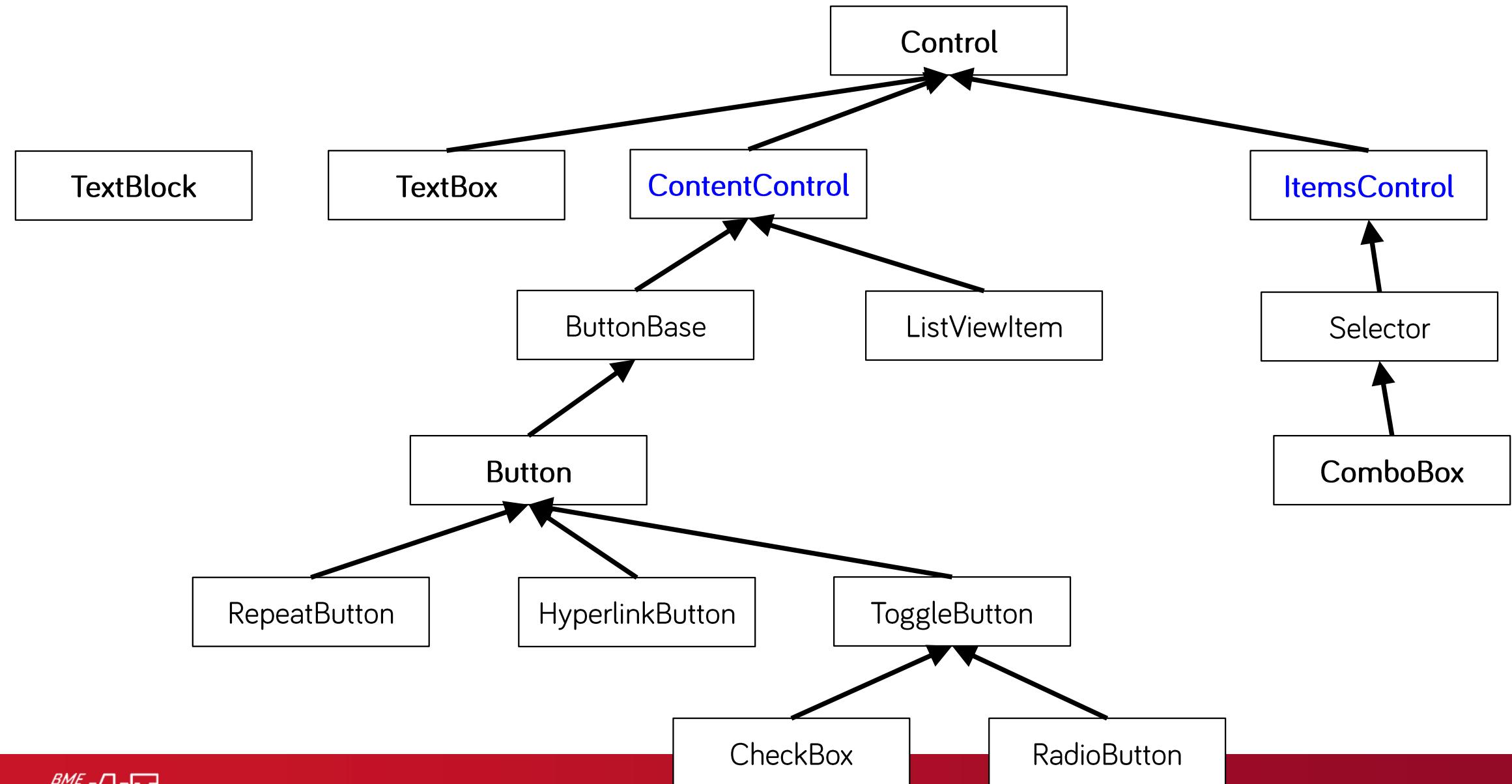
Button

RepeatButton

HyperlinkButton

# ContentControl

# Néhány vezérlő a hierarchiában



# TextBlock és TextBox

- TextBlock: formázott szöveg megjelenítése, egy vagy több sorban

```
<TextBlock>
    <Run FontFamily="Times New Roman" Foreground="DarkGray">
        Text in a TextBlock doesn't have to be a simple string.</Run>
    <LineBreak/>
    <Span>Text can be <Bold>bold</Bold>,
        <Italic>italic</Italic>, or <Underline>underlined</Underline>. </Span>
</TextBlock>
```

Text in a TextBlock doesn't have to be a simple string.  
Text can be **bold**, *italic*, or underlined.

- TextBox: szövegdoboz
  - Egysoros / többsoros
  - Helyesírásellenőrzés

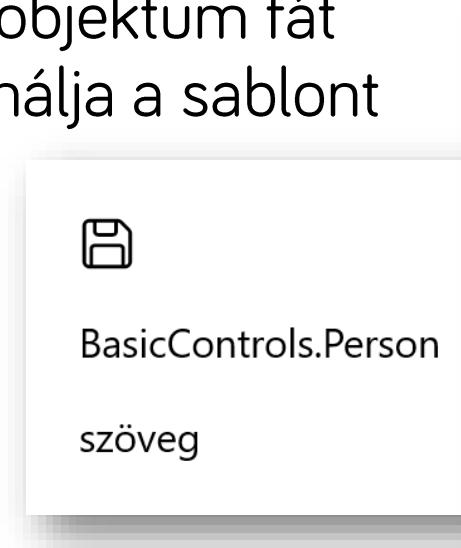
KORREKT FUTUROLÓGIAI JELENTÉS

Egészen biztos,  
hogy előbb-utóbb ez, vagy  
az lesz; így vagy úgy.

```
<TextBox AutomationProperties.Name="multi-line TextBox"
    TextWrapping="Wrap" AcceptsReturn="True" IsSpellCheckEnabled="True"
    SelectionHighlightColor="Green" MinWidth="400" >
```

# ContentControl : Control

- Valamilyen **tartalom** megjelenítésére szolgáló vezérlő
- Content property (object): a tartalom típusa szerint
  1. **UIElement**: megjeleníti azt az objektum fát
  2. **ContentTemplate** adott: használja a sablont
  3. **ToString**-et használ -> szöveg



```
<ContentControl>
|   <SymbolIcon Symbol="Save" />
</ContentControl>

<ContentControl>
|   <local:Person Name="Ken Wilber" />
</ContentControl>

<ContentControl Content="szöveg" />
```

- A local:Person egy saját Person osztályt példányosít a projekt névteréből (localként definiált névtér), amihez nem tartozik sablon  
➢ -> a ToString() alapértelmezetten a típust adja vissza

# ButtonBase : ContentControl

- Kezeli az **egér, érintés eseményeket**
  - > Click esemény
- Tulajdonságok
  - > IsPressed
  - > ClickMode: Release, Press, Hover
- Integrálódik a **Command mintával (MVVM)**
  - > Command, CommandParameter, CommandTarget

# Button, RepeatButton, HyperlinkButton

- **Button** : ButtonBase



- **RepeatButton** : ButtonBase

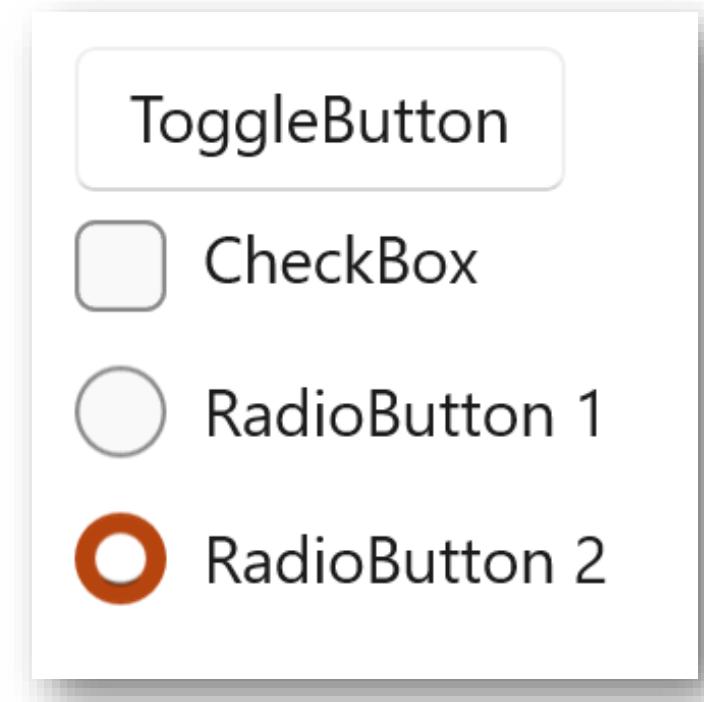
- > Delay property
- > Interval property
- > Nyomva tartva többször lövi el a Click eseményt

- **HyperlinkButton** : ButtonBase

- > NavigateUri, automatikusan böngészőt nyit

# ToggleButton, CheckBox

- *ToggleButton* : ButtonBase
  - > IsThreeState
    - IsChecked: lehet null is
  - > Checked események
- *CheckBox* : *ToggleButton*
  - > Saját megjelenése van
- *RadioButton* : *ToggleButton*
  - > Saját megjelenése van
  - > GroupName-mel lehet csoportba foglalni azokat amiből csak egyet lehet kiválasztani



```
<ToggleButton>ToggleButton</ToggleButton>
<CheckBox>CheckBox</CheckBox>
<RadioButton>RadioButton 1</RadioButton>
<RadioButton IsChecked="True">RadioButton 2</RadioButton>
```

# ItemsControl : Control

- Egyszerű lista megjelenítése kiegészítő funkciók nélkül (nincs görgetés, kijelölés, elemkattintás)
  - > Akkor használjuk, ha a listát csak megjeleníteni kell
- Megjelenítendő lista
  - > Items: XAML-ben adjuk meg, fix lista
  - > **ItemsSource**: tipikusan adatkötéssel adjuk meg, alább imperatívan

```
<ItemsControl x:Name="items"/>

public MainWindow()
{
    InitializeComponent();

    items.ItemsSource = new[] { "alma", "körte", "banán" };
}
```

alma  
körte  
banán

Kódot nem  
kell tudni

# ItemsControl . ItemsPanel property

- ItemsPanel property: elem elrendező vezérlő

```
<ItemsControl x:Name="items">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <StackPanel Orientation="Horizontal" />
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>
</ItemsControl>
```

almakörtebanán

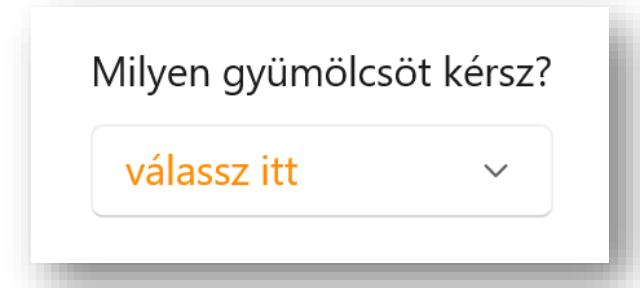
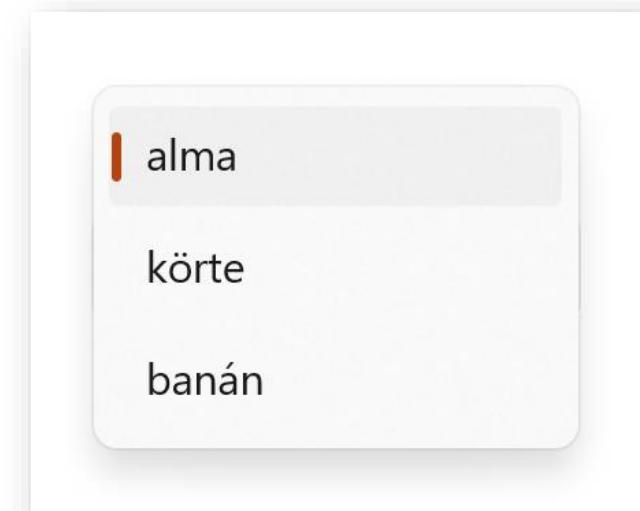
- ItemTemplate: az elemek megjelenését írja le
  - Ez lesz a benne lévő ContentControlok (például ListViewItem) ContentTemplate-je
- GroupStyle: csoportok stilizálása
  - Fejléc, tartalom, tartalom elrendező panel

# Selector : ItemsControl

- Az elemek kiválaszthatóak
- SelectedIndex: 0 alapú index
- SelectedItem
  - > A kiválasztott objektum
- SelectionChanged esemény

# ComboBox : Selector

- IsDropDownOpen
  - > Programozottan is lenyitható
  - > DropDownOpened/Closed események
- Header, HeaderTemplate
  - > A felette lévő szöveget lehet megadni
- PlaceholderText: üres megjelenés helyett



```
<ComboBox Header="Milyen gyümölcsöt kérsz?" PlaceholderText="válassz itt"  
PlaceholderForeground="#DarkOrange" x:Name="items"/>
```

# Dependency Property

```
<TextBlock Name="txt" Text="Cloud" />
```

# Vezérlők paraméterei

- Egy vezérlőnek rengeteg tulajdonsága van
  - > Egy Buttonnak 135 propertyje és 52 eseménye van
- A komponálás miatt a **sok objektum** ágyazódik egymásba -> sok objektumunk van
- A legtöbb tulajdonságot, eseményt nem állítjuk
- Jó lenne csak azokat az adatokat tárolni, ami valóban be van állítva és más, mint az alapértelmezett érték!

# További célok

- Egy vezérlőhöz lehessen **kontextus függően további információkat** is hozzáadni
  - > Ha táblázatban van, akkor melyik sorban, oszlopban
- **Adatkötés:** a tulajdonságok értéke automatikusan frissüljön egy adatforrásból
- **Stilizálás:** lehessen vezérlők tulajdonságait egyszerre, egységesen beállítani
- **Animáció:** egységes módszer egy tulajdonság időbeni változtatására
- Mindezt reflexió nélkül!

# Megoldás: DependencyProperty

- A XAML vezérlők nem hagyományos C#-os mezőkkel vagy autopropertykkal rendelkeznek
- minden vezérlő alaposztálya a **DependencyObject**, ami tárolja a vezérlő paramétereit, tulajdonságait
  - > Klasszikus kulcs-érték párok, dictionary, mint JavaScript object
- Típusos programozói felület propertyken keresztül
- Tetszőlegesen, futásidőben is bővíthető tároló
- Metaprogramozás, általános szolgáltatások: adatkötés, stilizálás, animáció

# DependencyProperty használata

Kódot nem  
kell tudni

- Hagyományos propertyként jelenik meg, típusos
  - > Mind XAML-ben, mind C#-ban

```
<TextBlock Name="txt" Text="HU" />
```

```
// normál property
txt.Text = "HU";
```

- Használhatjuk a kulcs-érték pár beállítást is

```
// kulcs-érték pár beállítása az alaposztályon
DependencyObject dpObj = txt;
// a dependency property a tárolás kulcsa
dpObj.SetValue(textBox.TextProperty, "HU");
```

# DependencyProperty deklarálása

```
public class TextBox : Control
{
    public string Text
    {
        // az alaposztály kulcs-érték tárolója
        get { return (string)GetValue(TextBox.TextProperty); }
        set { SetValue(TextBox.TextProperty, value); }
    }

    // statikus kulcs objektum a hivatkozáshoz
    public static readonly DependencyProperty TextProperty =
        DependencyProperty.Register(
            nameof(Text),
            typeof(string),
            typeof(TextBox),
            new PropertyMetadata("", OnIsDefaultChanged));

    private static void OnIsDefaultChanged(
        DependencyObject o, DependencyPropertyChangedEventArgs e) {}
}
```

Kódot nem  
kell tudni

# Csatolt tulajdonságok (attached properties)

- Az objektumhoz a saját típusán kívül eső **további tulajdonságok** is eltárolhatók
  - > Ugyanaz a tárolási mechanizmus a DependencyObjectben, csak kell egy másik kulcs
- **Bárki eltárolhat bármit és lekérdezhet bármit**
  - > Nem kell hierarchiában felette lenni stb.
- **Tipikusan vezérlő hierarchiában használjuk**
  - > Canvas: rajta lévő elemek pozíciója
  - > Grid: rajta lévő elemek helye a táblázatban
  - > Árnyék, animáció stb.

# Csatolt tulajdonságok használata

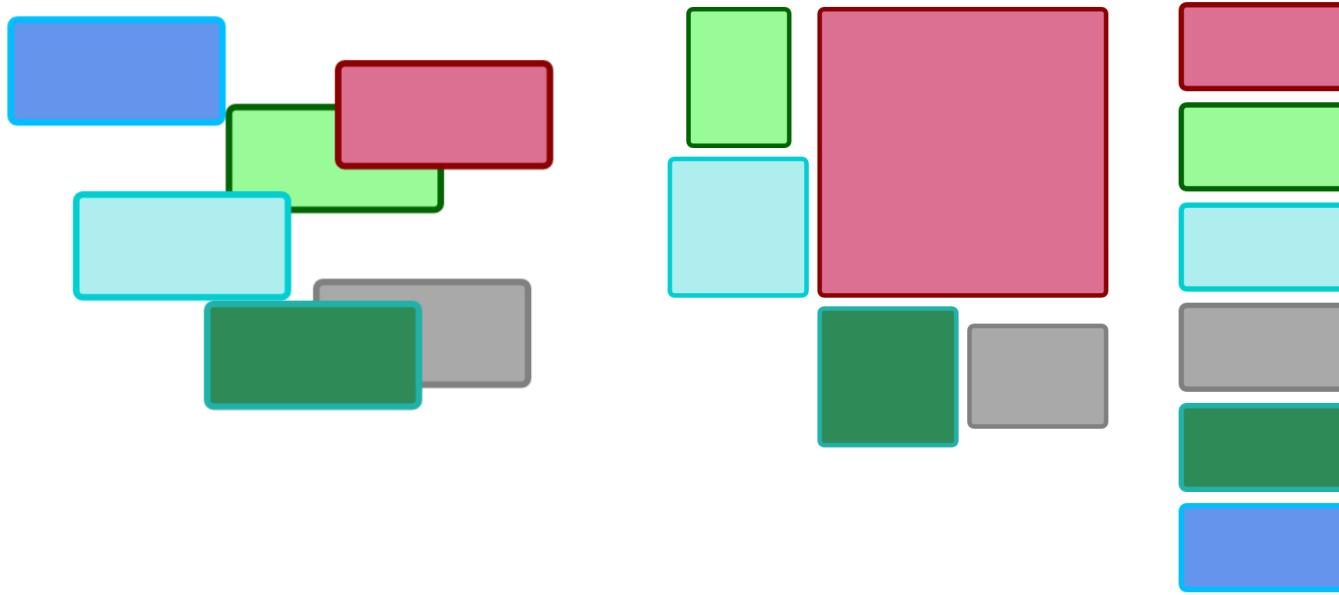
- A csatolt tulajdonság tulajdonosa aki azt deklarálja, le fogja kérdezni és fel fogja használni
  - > A Canvas.Left a Canvason kívül is beállítható, csak senki nem fogja lekérdezni, egyszerű adat

```
<Canvas>
  <TextBlock Canvas.Left="120" Canvas.Top="20" Name="txt" Text="bla" />
</Canvas>
```

```
// kulcs-érték párokkal
dpObj.SetValue(Canvas.LeftProperty, 120);
dpObj.SetValue(Canvas.TopProperty, 20);
// a csatolt tulajdonságot definiáló osztál metódusával
Canvas.SetLeft(txt, 120);
Canvas.SetTop(txt, 20);
```

Az alsó kódot nem kell tudni

# Elrendezés



# Vezérlő méretezése

```
<Rectangle Height="40" Width="50"  
HorizontalAlignment="Left"  
VerticalAlignment="Bottom"
```

- Vezérlő szélessége (Width) és magassága (Height)
  - Fix méret
    - > A vezérlő a megadott méretet veszi fel az adott irányban, függetlenül a tartalmától - így a tartalom levágásra kerülhet!
    - > **Width / Height = "30"**
    - > Ha az igazítás Stretch (lásd következő slide), akkor is fix a méret, a vezérlő középre kerül
    - > Ritkán javasolt, mert nem alkalmazkodik a tartalomhoz és az átméretezéshez
  - Automatikus méret - alapértelmezett
    - > **Width / Height = "Auto"** (mivel ez az alapértelmezett, nem szoktuk kiírni)
    - > Ha az igazítás Left / Right / Center -> tartalom alapján méreteződik
    - > Ha az igazítás Stretch -> a szülő méretezi, kitölți a rendelkezésre álló teret az adott irányban

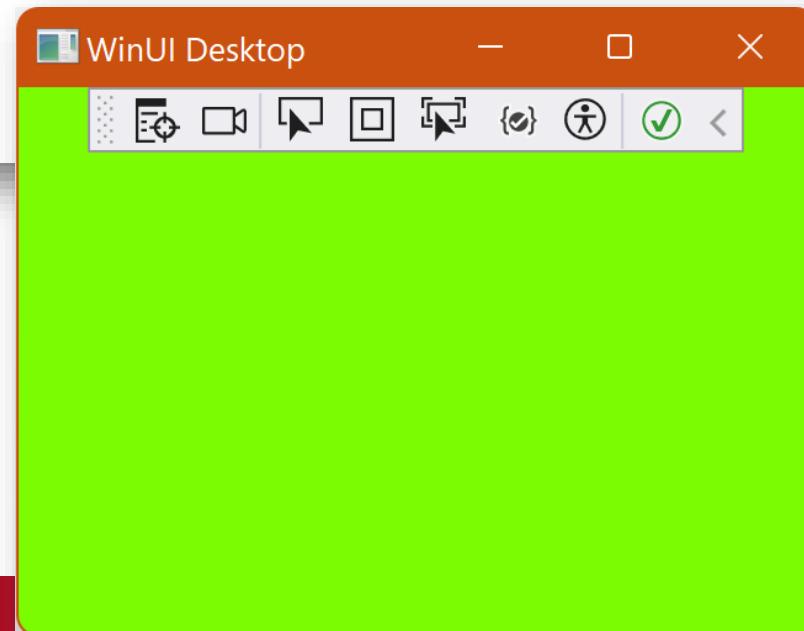
# Vezérlő igazítása

```
<Rectangle Height="40" Width="50"  
HorizontalAlignment="Left"  
VerticalAlignment="Bottom"
```

- Elrendezés a szülő panelen belül:  
**HorizontalAlignment / VerticalAlignment** tulajdonságok
  - > **Left / Top / Center / Right / Bottom**
    - Rendezi az elemet balra, felülrre, középre, jobbra, alulra
  - > **Stretch** - az igazításon túl a *méretezésre is hatással* van
    - "Auto" méretezés esetén kitölti a rendelkezésre álló teret, a szülő mérete a meghatározó, nem a tartalom
    - Fix méretezés esetén középre kerül a vezérlő
- Alapértelmezett érték a **Stretch**
  - Kivéve pár vezérlőt, például **TextBlock**, **ComboBox**, **Checkbox**

# Alapértelmezett elrendezés

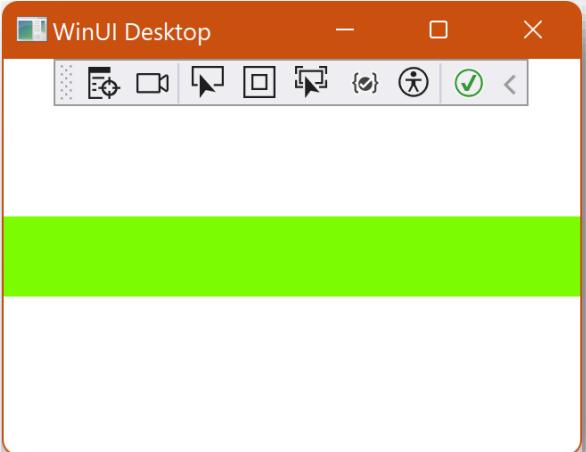
```
<?xml version="1.0" encoding="utf-8"?>
<Window
    x:Class="LayoutSample.MainWindow2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:LayoutSample"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
    <Rectangle Fill="#LawnGreen" />
```



# Fix méretű elrendezések

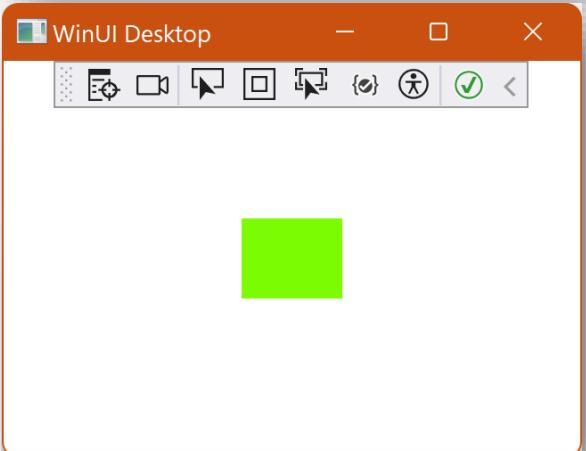
- A Stretch nem írja felül a fix méreteket

```
<Rectangle Height="40" Fill="LawnGreen" />
```



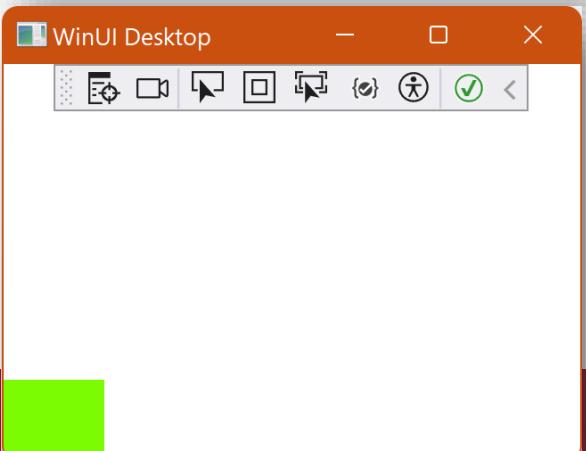
- A Stretch középre igazít ha be van állítva a méret

```
<Rectangle Height="40" Width="50" Fill="LawnGreen" />
```



- Igazítás állítása

```
<Rectangle Height="40" Width="50"  
         HorizontalAlignment="Left"  
         VerticalAlignment="Bottom" Fill="LawnGreen" />
```



# Vezérlők távolsága

- Margin: Távolság másik vezérlőtől, a szülő panel széleitől
  - > Thickness típus, négy irányt lehet megadni

```
<Border BorderBrush="Black" BorderThickness="3"  
        HorizontalAlignment="Center"  
        VerticalAlignment="Center" >  
  
    <TextBlock Text="0" Margin="0" Foreground="DarkGreen" />  
  
</Border>
```

0

10

10,20

10,20,35,5

Ez a TextBlock széle

# Vezérlők tartalma

- **Padding:** A tartalom távolsága a szülő vezérlő széleitől befelé
  - > Thickness típus, négy irányt lehet megadni
  - > <https://learn.microsoft.com/en-us/windows/apps/design/layout/alignment-margin-padding>
- Csak néhány vezérlőnek van ilyen tulajdonsága!

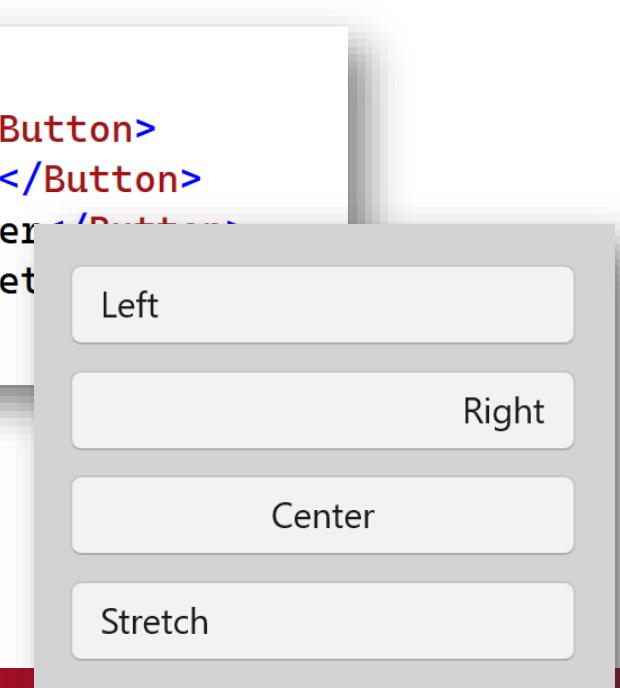
```
<Border Background="DarkGreen" HorizontalAlignment="Center" VerticalAlignment="Center" >
|   <TextBlock Text="0" Padding="0" Foreground="LawnGreen" />
</Border>
```



# Tartalom igazítása

- Tartalom igazítása a szülőn belül
  - > Vertical / HorizontalContentAlignment
    - Left / Top / Center / Stretch...
- Csak néhány vezérlőnek van ilyen tulajdonsága

```
<StackPanel Padding="20" Spacing="10" Background="LightGray">
    <Button Width="200" HorizontalContentAlignment="Left">Left</Button>
    <Button Width="200" HorizontalContentAlignment="Right">Right</Button>
    <Button Width="200" HorizontalContentAlignment="Center">Center</Button>
    <Button Width="200" HorizontalContentAlignment="Stretch">Stretch</Button>
</StackPanel>
```



# Láthatóság

- Visibility tulajdonság
  - > Enum, nem boolean! ☹
  - > Visible / Collapsed

```
<Button Visibility="Collapsed" >0k</Button>
```

# Méretezés – Width, Height

- Tulajdonságok:
  - > Width, Height (def: Auto = double.NaN)
  - > MinWidth, MinHeight (def: 0)
  - > MaxWidth, MaxHeight (def: double.PositiveInfinity)
- Ezek javasolt értékek a layout rendszernek
- A panelek helyezik el a vezérlőket és állítják be a méretet
- **A valódi méret: ActualHeight, ActualWidth**
  - > DesiredSize -> RenderSize

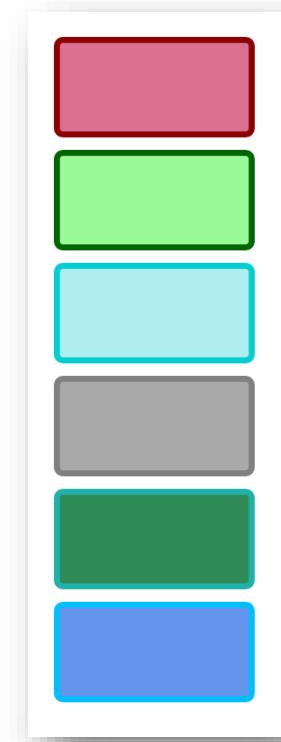
# Panel alaposztály

- A többi panel ebből származik, alaptulajdonságok:
  - > Children: gyerek elemek
  - > Background: háttér
- Fontsabb panelek:
  - > **StackPanel, Canvas, Grid**
  - > VariableSizedWrapGrid, RelativePanel
  - > VirtualizingPanel: alaposztály a UI virtualizáláshoz
    - Csak az a gyerek vezérlő jön létre, ami látszik a képernyőn
    - -> Nagyobb listák esetén kötelező ilyen paneleket használni!

# StackPanel

- Elemek egymás alatt vagy mellett
  - > Orientation: Horizontal / Vertical
- Virtualizált változat:
  - > VirtualizingStackPanel

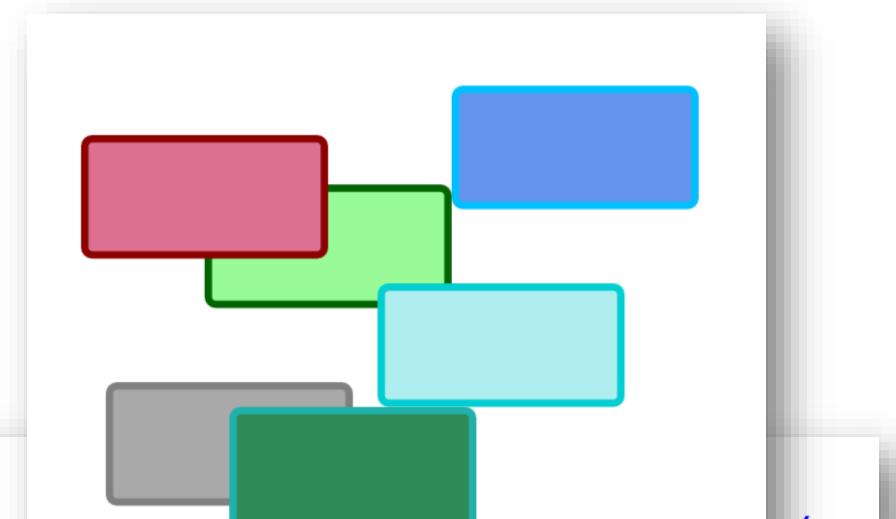
```
<StackPanel Orientation="Vertical" Spacing="6">
    <Rectangle Style="{StaticResource rect1}" />
    <Rectangle Style="{StaticResource rect2}" />
    <Rectangle Style="{StaticResource rect3}" />
    <Rectangle Style="{StaticResource rect4}" />
    <Rectangle Style="{StaticResource rect5}" />
    <Rectangle Style="{StaticResource rect6}" />
</StackPanel>
```



# Canvas

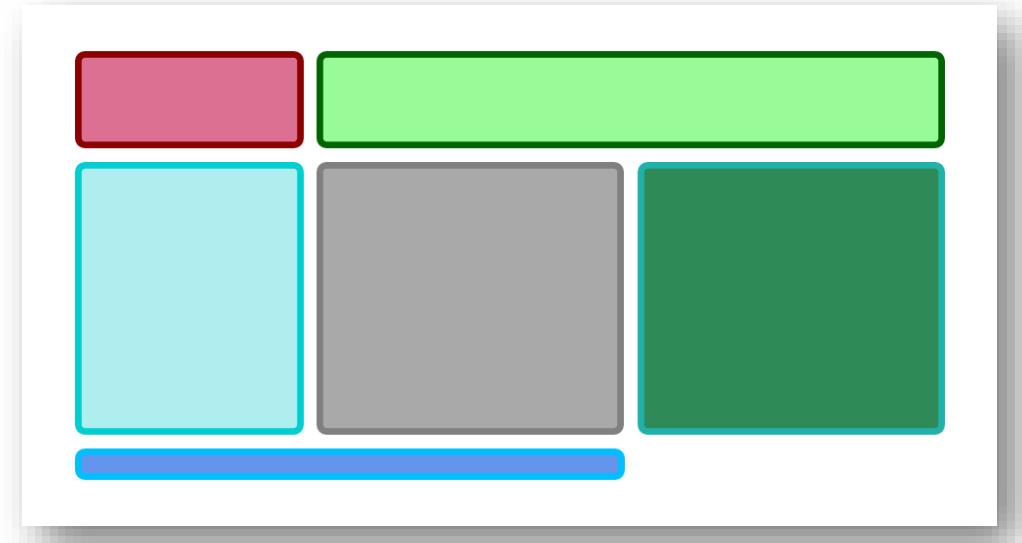
- Gyerek vezérlők szabadon elhelyezhetők fix pixel koordinátákkal:
  - > Canvas . Left / Top / ZIndex
- A vezérlők átfedhetnek!

```
<Canvas>
    <Rectangle Canvas.Left="30" Canvas.Top="50" Canvas.ZIndex="10" />
    <Rectangle Canvas.Left="80" Canvas.Top="70" Canvas.ZIndex="2" />
    <Rectangle Canvas.Left="150" Canvas.Top="110" Canvas.ZIndex="5" Style="{StaticResource rect3}" />
    <Rectangle Canvas.Left="40" Canvas.Top="150" Canvas.ZIndex="3" Style="{StaticResource rect4}" />
    <Rectangle Canvas.Left="90" Canvas.Top="160" Canvas.ZIndex="40" Style="{StaticResource rect5}" />
    <Rectangle Canvas.Left="180" Canvas.Top="30" Canvas.ZIndex="-3" Style="{StaticResource rect6}" />
</Canvas>
```



# Grid

- Táblázatos megjelenítés
- Sorok/oszlopok méretének megadása
  - > Fix pixel méret: "30"
  - > Automatikus: "Auto"
    - A tartalmazott vezérlőktől függ
  - > Arányos (\*): "2\*" / "3\*"
  - > Maximális / minimális méret
- Gyerekelemek elhelyezése csatolt tulajdonságokkal
  - > Grid . Row
    - > melyik sorba kerüljön az elem
  - > Grid . Column
    - > melyik oszlopra kerüljön az elem
  - > Grid . RowSpan
    - > több cellát foglaljon el abban az oszloban
  - > Grid . ColumnSpan
    - > több cellát foglaljon el abban a sorban

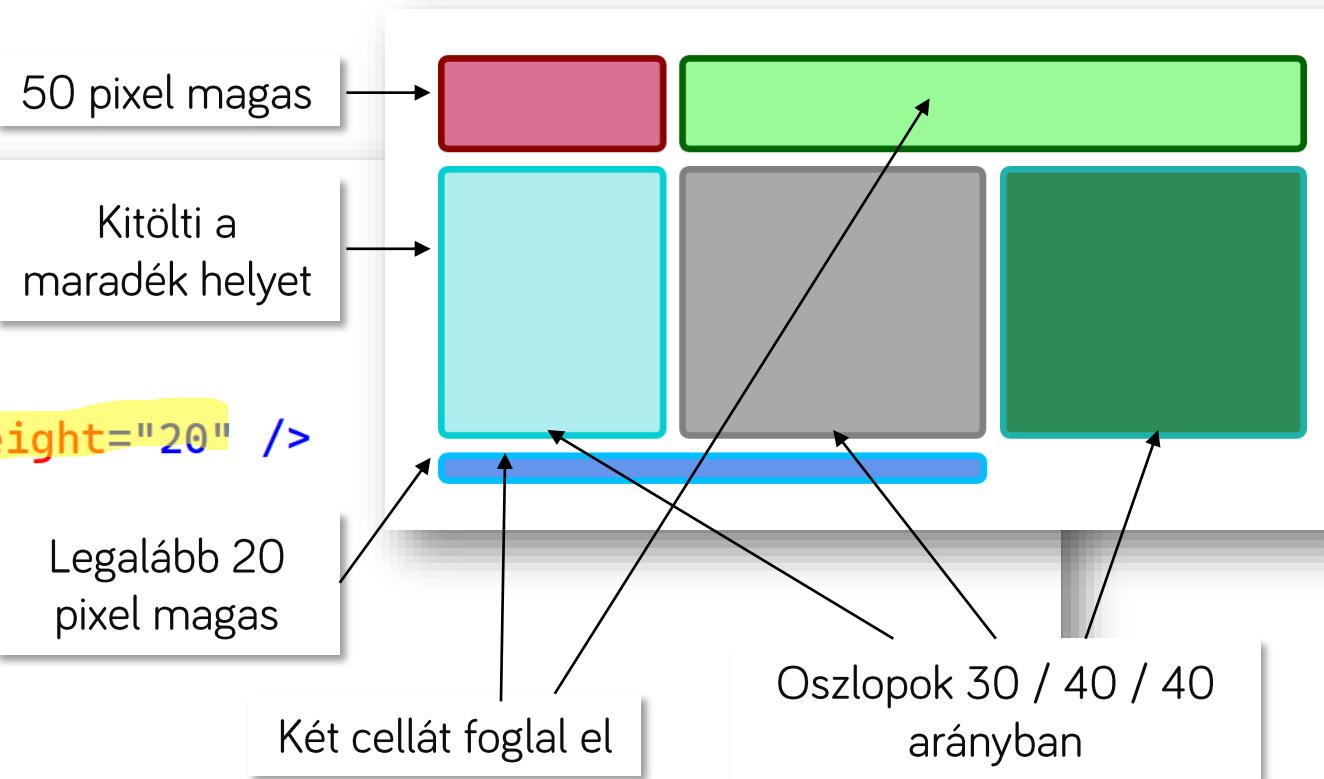


# Az előző példa kódja

```
<Grid Height="200" Width="400">
    <Grid.RowDefinitions>
        <RowDefinition Height="50" />
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" MinHeight="20" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="30*" />
        <ColumnDefinition Width="40*" />
        <ColumnDefinition Width="40*" />
    </Grid.ColumnDefinitions>

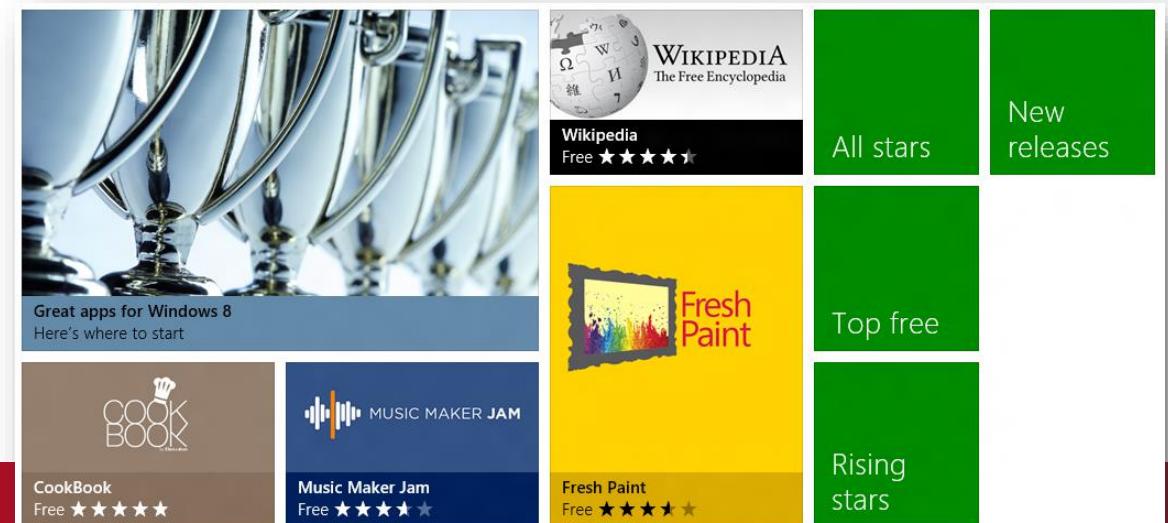
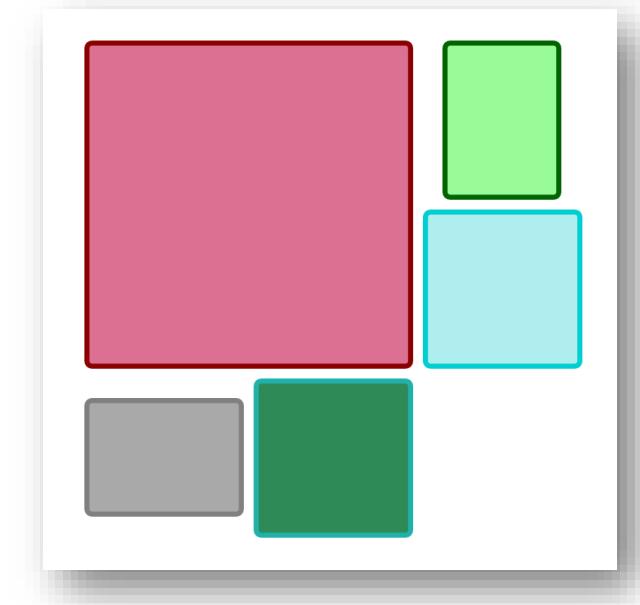
    <Rectangle Grid.Row="0" Grid.Column="0" Width="Auto" Height="Auto" Style="{Stat:...}>
    <Rectangle Grid.Row="0" Grid.Column="1" Grid.ColumnSpan="2" Width="Auto" Height="Auto" Style="{Stat:...}>
    <Rectangle Grid.Row="1" Grid.Column="0" Width="Auto" Height="Auto" Style="{Stat:...}>
    <Rectangle Grid.Row="1" Grid.Column="1" Width="Auto" Height="Auto" Style="{Stat:...}>
    <Rectangle Grid.Row="1" Grid.Column="2" Width="Auto" Height="Auto" Style="{Stat:...}>
    <Rectangle Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2" Width="Auto" Height="Auto" Style="{Stat:...}>

</Grid>
```



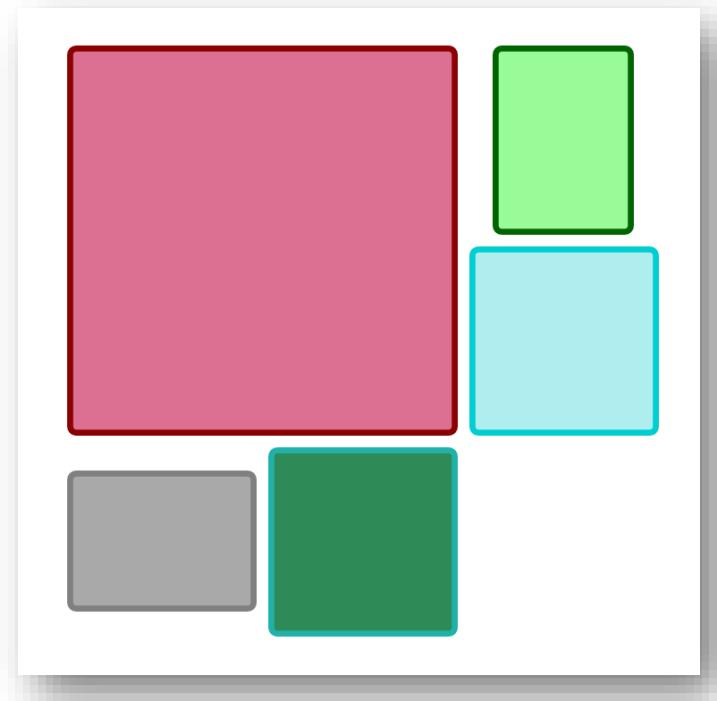
# VariableSizedWrapGrid

- Sor- vagy oszlopfolytonos elrendezés
  - > Orientation / MaximumRowsOrColumns
- Elemek mérete:
  - > Mindegyik saját mérete
  - > Alapértelmezett: ItemHeight, ItemWidth
  - > Több cella kitöltése: csatolt RowSpan, ColumnSpan
- Nem virtualizált! ☹



# Az előző példa kódja

- Alapértelmezett méretek beállítása
  - > 100x100
- Egyes vezérlők ezt átírják -> 70
  - > Egyébként a Rectangle kitölti a teret
- Több cella elfoglalása



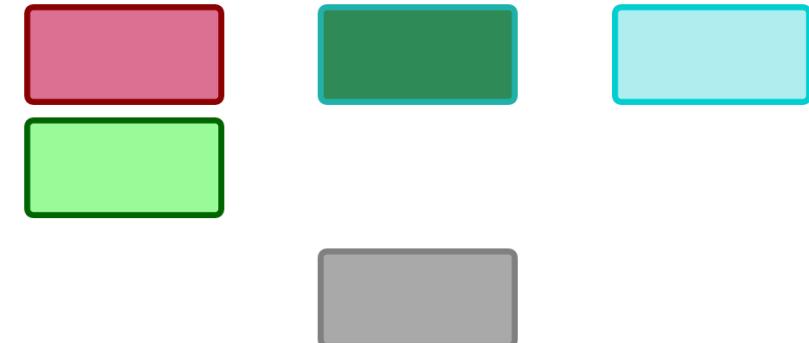
```
<VariableSizedWrapGrid Height="300" Width="400" ItemWidth="100" ItemHeight="100"
    MaximumRowsOrColumns="3" Orientation="Horizontal">
    <Rectangle Width="Auto" Height="Auto"
        VariableSizedWrapGrid.RowSpan="2"
        VariableSizedWrapGrid.ColumnSpan="2" Style="{StaticResource rect1}" />
    <Rectangle Width="70" Height="Auto" Style="{StaticResource rect2}" />
    <Rectangle Width="Auto" Height="Auto" Style="{StaticResource rect3}" />
    <Rectangle Width="Auto" Height="70" Style="{StaticResource rect4}" />
    <Rectangle Width="Auto" Height="Auto" Style="{StaticResource rect5}" />
</VariableSizedWrapGrid>
```

Kódot nem  
kell tudni

# RelativePanel

- Az elemek helyét egymáshoz és a panelhez képest adjuk meg:  
alatta, fölötté, balra, jobbra
  - > Kevesebb panel egybeágazás
  - > Akkor hasznos, ha nem lineáris a layout

```
<RelativePanel Width="400" Height="300">
    <Rectangle x:Name="r1" Style="{StaticResource rect1}" />
    <Rectangle x:Name="r2" RelativePanel.Below="r1" Style="{StaticResource rect2}" />
    <Rectangle x:Name="r3" RelativePanel.AlignRightWithPanel="True" Style="{StaticResource rect3}" />
    <Rectangle x:Name="r4"
        RelativePanel.AlignHorizontalCenterWithPanel="True"
        RelativePanel.AlignVerticalCenterWithPanel="True" Style="{Sta
    <Rectangle x:Name="r5"
        RelativePanel.AlignBottomWith="r3"
        RelativePanel.AlignLeftWith="r4" Style="{StaticResource rect5
</RelativePanel>
```



# Példa: RelativePanel nélkül

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Image x:Name="img" .../>
    <StackPanel Grid.Column="1" ...>
        <TextBlock x:Name="title" .../>
        <TextBlock x:Name="authors" .../>
        <TextBlock x:Name="summary" .../>
        <Button Content="Download" .../>
    </StackPanel>
</Grid>
```



# RelativePanel

```
<RelativePanel>
    <Image x:Name="img" .../>
    <TextBlock x:Name="title"
        RelativePanel.RightOf="img"
        RelativePanel.AlignTopWith="img" .../>
    <TextBlock x:Name="authors,,"
        RelativePanel.RightOf="img"
        RelativePanel.Below="title" .../>
    <TextBlock x:Name="summary,,"
        RelativePanel.RightOf="img"
        RelativePanel.Below="authors" .../>
    <Button Content="Download" RelativePanel.RightOf="img"
        RelativePanel.AlignBottomWithPanel="True" .../>
</RelativePanel>
```



The Game  
*Jack London*

On the eve of their wedding, twenty-year-old Anna has invited her sweetheart to view her only rival: the "game" of prizefighting. She has chosen the pr

Download

# Erőforrások



**Emeline Rochefeuille**

Follower

Points 38



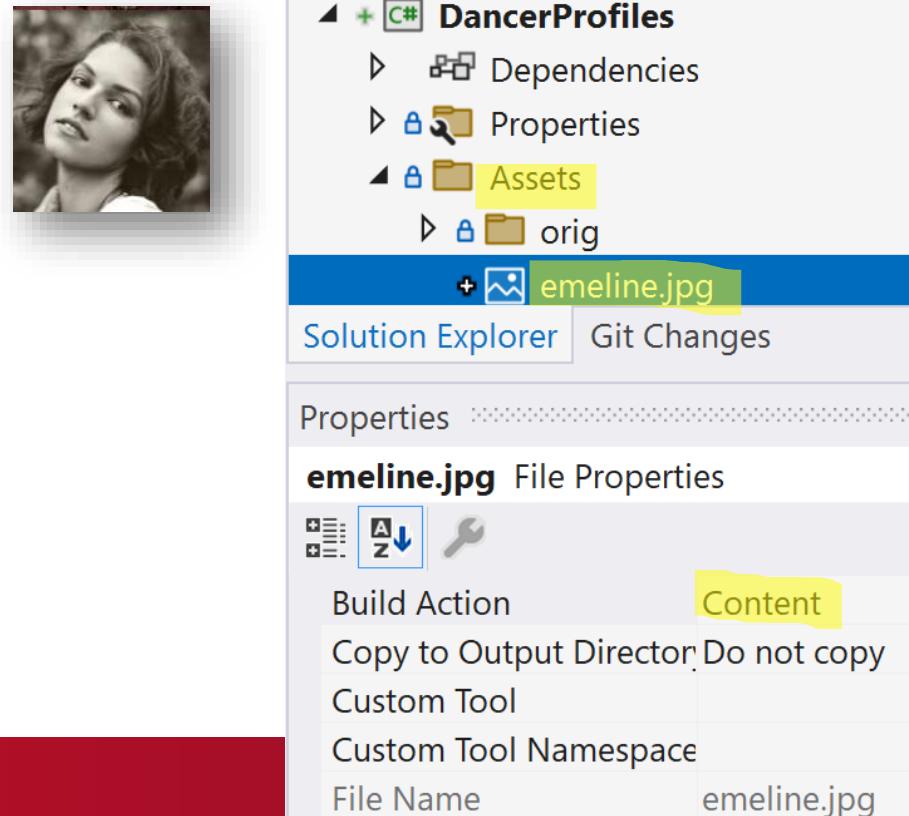
**Maxence Martin**

Leader

# Fájlként csatolt, bináris erőforrások - KÉPEK

- Visual Studio-ban Content-ként csatolt fájlok
  - > Egyszerű hivatkozás XAML-ből:

```
<Image Source="assets/emeline.jpg" />
```



# Szöveges erőforrások és vezérlők

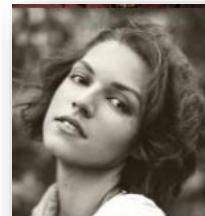
- A vezérlők azonosítója az x:Uid
- Az erőforrás fájlban a kulcs első eleme azUid a második a beállítandó tulajdonság
  - > Tetszőleges tulajdonság beállítható
  - > Nyelvenként készíthető erőforrás fájl

```
<Run x:Uid="txtPoints" />
<Run Text="{x:Bind Points}" />
```

Resources.resw

Add Resource Remove Resource

	Name	Value
▶	txtPoints.Text	Points
*		



**Emeline Rochefeuille**  
Follower  
Points 38



**Maxence Martin**  
Leader  
Points 92



**Virginie Grondin**  
Follower  
Points 24

# Stílusok

- Tetszőleges tulajdonság egységes beállítása

```
<Style TargetType="Rectangle" x:Key="baseRectStyle">
    <Setter Property="RadiusX" Value="3"/>
    <Setter Property="RadiusY" Value="3"/>
    <Setter Property="StrokeThickness" Value="3" />
```

- Egy vezérlőhöz egyszerre csak egy stílus tartozhat
  - > A stílusok származhatnak egymásból
- Tipikusan erőforrásként definiáljuk a stílusokat
  - > **Implicit:** TargetType megadásával és az x:Key elhagyásával automatikusan érvényre jut
  - > **Explicit:** a stílust explicit kell hivatkozni a nevével (x:Key)
- minden beépített vezérlőnek van saját stílusa, az határozza meg a kinézetét a **Template**-tel

# XAML erőforrások

- Tetszőleges C#-os osztály példányosítása, paraméterezése XAML-ből
- Az x:Key az erőforrás "neve", a kulcs az erőforrás dictionaryben, amivel hivatkozunk rá
  - > Keresés: a vezérlőhierarchiában bentről kifelé, végül a globális *app.xaml*ben

```
<Grid>
    <Grid.Resources>
        <SolidColorBrush Color="LightBlue" x:Key="background1"/>
    </Grid.Resources>
```

- Felhasználása StaticResource-szal

```
<TextBox Background="{StaticResource background1}" Width="100" Height="24" />
```

- Dekomponálás, erőforrásfájlok összefűzése

Kék? szép?

# Stílus + erőforrás példa

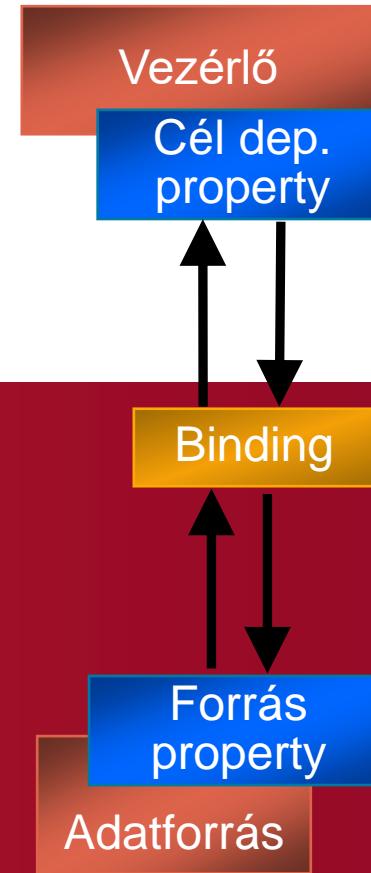
- Lokális, explicit erőforrások a griden belül, származással

```
<Grid>
    <Grid.Resources>
        <Style TargetType="Rectangle" x:Key="baseRectStyle">
            <Setter Property="RadiusX" Value="3"/>
            <Setter Property="RadiusY" Value="3"/>
            <Setter Property="StrokeThickness" Value="3" />
            <Setter Property="Width" Value="100" />
            <Setter Property="Height" Value="50" />
            <Setter Property="Margin" Value="3" />
        </Style>
        <Style TargetType="Rectangle" x:Key="rect1"
            BasedOn="{StaticResource baseRectStyle}">
            <Setter Property="Fill" Value="PaleVioletRed" />
            <Setter Property="Stroke" Value="DarkRed" />
        </Style>
    </Grid.Resources>
    <Rectangle Style="{StaticResource rect1}" />

```



# Adatkötés



# Példa: source, path

The screenshot shows a Windows application window titled "Fred Astaire". Inside the window, there are four input fields: "Név" (Name) containing "Fred Astaire", "Tánc" (Dance) set to "Sztepp", "Pontok" (Points) set to "0", and "Szerep" (Role) with three radio button options: "Szóló" (Speaker), "Vezető" (Leader), and "Követő" (Follow). Below the window is a C# code editor displaying the following code:

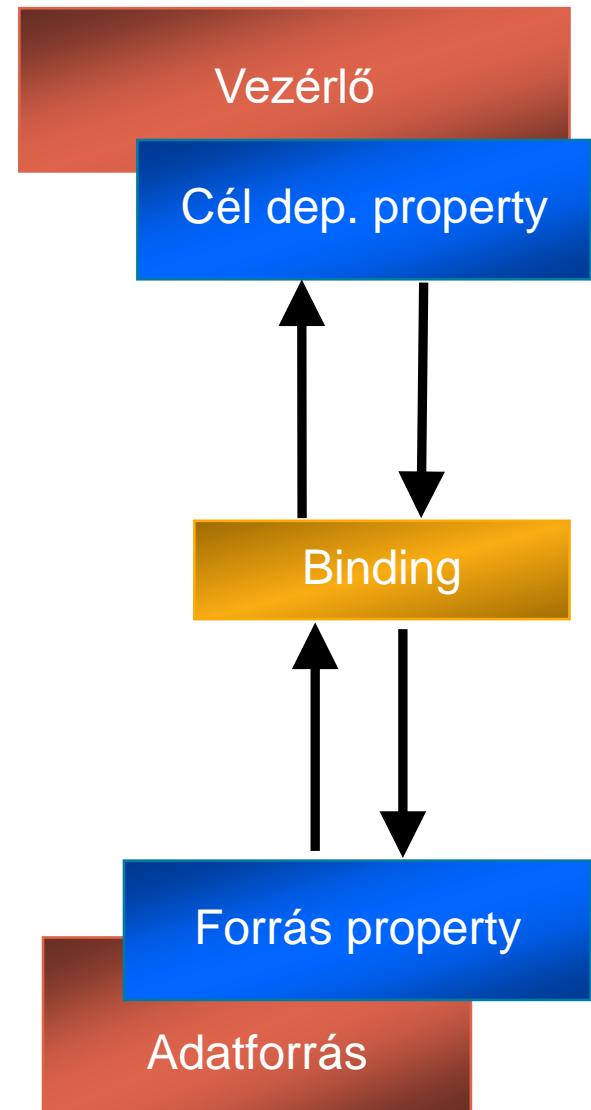
```
public sealed partial class MainWindow : Window
{
    public Dancer Dancer { get; set; } = new Dancer();

    public class Dancer
    {
        public string Name { get; set; } = "Fred Astaire";
        public int? Points { get; set; }
        public string Nationality { get; set; } = "American";
        public Role Role { get; set; } = Role.Leader | Role.Solo;
    }
}
```

Blue arrows point from the "Név" field to the "Name" property in the code, and from the "Pontok" field to the "Points" property. A red arrow points from the "Szerep" radio buttons to the "Role" property.

# Adatkötés XAML-ben

- Egy vezérlő tulajdonságainak hozzákötése az adatforráshoz
  - > Vezérlő dependency propertyjéhez kötünk
  - > Adatforrás: tetszőleges objektum propertyje
  - > Kötés: változik az egyik -> változzon a másik!
- Két fajta szintaktika
  - > {x:Bind ...} markup extension
  - > {Binding ...} korábbi megoldás



# Binding osztály

- Az adatkötés beállításakor implicit létrejön, tárolja az adatkötés jellemzőit
- Adatforrás
  - > {x:Bind} esetén *mindig* a szülő vezérlő, amin a vezérlő rajta van, annak a code behindja (kivéve DataTemplate esetén)
  - > {Binding} esetén beállítható, precedencia:
    1. Explicit: Binding . Source
    2. A vezérlő DataContext propertyje
    3. A vezérlő szülőjének DataContext propertyje

```
<Grid.Resources>
    <local:Dancer x:Key="myDancer" Name="Michael Jackson" />
</Grid.Resources>

<TextBlock Text="{Binding Source={StaticResource myDancer}, Path=Name}">
```

# x:Bind . Path

- Az adatforrás elemét határozza meg
- Megadási lehetőségek:
  - > egyszerű tulajdonság név, pl.:PropertyName
  - > elérési útvonal a propertyken keresztül pl.: A.B
  - > hivatkozás csatolt tulajdonságra pl.: (Canvas.Left)
  - > indexelt elem pl.: Items[0]
  - > gyűjtemény esetén az aktuális elem kiválasztása: „/”
- A Path szintaktikailag elhagyható, ha első helyen szerepel, pl.:

```
<TextBox Text="{x:Bind,  
Mode=TwoWay,  
Path=Dancer.Name}" />
```

```
<TextBox Text="{x:Bind Path=Dancer.Name, Mode=TwoWay}" />
```

ezzel ekvivalens:

```
<TextBox Text="{x:Bind Dancer.Name, Mode=TwoWay}" />
```

# x:Bind . Mode

- Az adatkötés iránya
- **OneTime**: csak a felület betöltésekor olvassa ki az értéket - **alapértelmezett** 😊
- **TwoWay**: kétirányú kötés, a forrás- és cél property változása frissíti a másikat
  - > **TextBox esetén is kötelező megadni!**
- **OneWay**: amikor a forrás megváltozik, frissül a cél
  - > Gyakran be kell állítani, hogy figyelje a változásokat!
- **OneWayToSource**: csak a forrás frissül, ha a cél megváltozik

# x:Bind érvénytelen adatok

- **FallbackValue**
  - > Ha hiba történt a forrás olvasásánál - például még nem jött meg az aszinkron adat a szerverről stb.

```
<TextBox Text="{x:Bind Dancer.Name, FallbackValue='Loading...', ... }" />
```

- **TargetNullValue**
  - > Ha a levél elem (itt a Name property) null, például nincs kitöltve stb.

```
<TextBox Text="{x:Bind Dancer.Name,
    TargetNullValue=' - Unknown - ',
    FallbackValue='Loading...', 
    Mode=TwoWay}" />
```

# x:Bind mikor frissül az adat?

- Alapértelmezett: amikor változik a property
  - > Frissül a vezérlő propertyje -> frissül az adatforrás
- Kivéve TextBox esetén: LostFocus
  - > Akkor frissül, amikor a felhasználó kilép a szövegdobozból
- Testreszabható: UpdateSourceTrigger
  - > *PropertyChanged / LostFocus*
  - > *Explicit* csak a {Binding} támogatja: csak amikor kódból meghívjuk az Update metódust a Bindingon

```
<TextBox Text="{x:Bind Dancer.Name,  
UpdateSourceTrigger=PropertyChanged, ... }" />
```

# Események és funkciók kötése

- Események kötése metódusokhoz

```
<Button Click="{x:Bind ViewModel.OnSave}" />
```

- Property kötése funkciókhöz

```
<Button IsEnabled="{x:Bind IsStringNotEmpty(Dancer.Name), Mode=OneWay}">OK</Button>
```

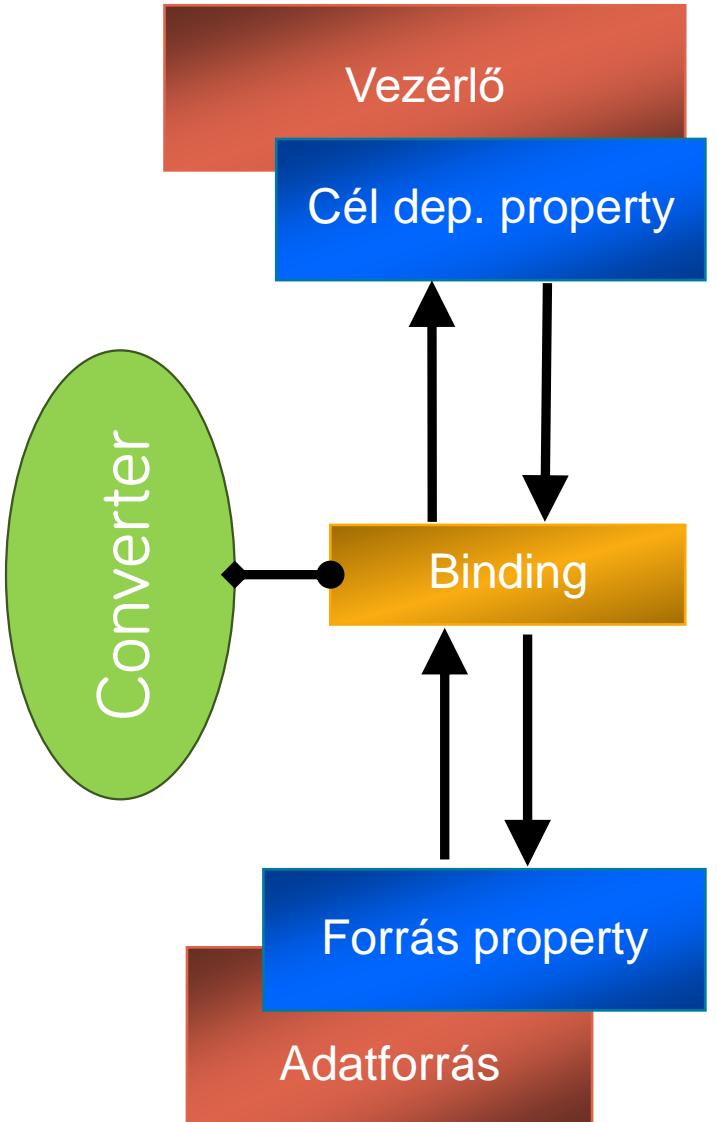
```
public bool IsStringNotEmpty(string str)  
=> !string.IsNullOrWhiteSpace(str);
```

- Ha a paraméterek változnak (itt Dancer.Name), meghívódik a függvény
- Kasztolás: csak Visibilityre, az van beépítve

```
<Button Visibility="{x:Bind ((Visibility)IsValid), Mode=OneWay}">OK</Button>
```

# Adatkonverzió

- A cél és forrás property típusa nem egyezik
  - > Technikailag használható másra is: formázás, többnyelvűsítés stb.
- **IValueConverter** interfészét kell implementálni
  - > Convert metódus
  - > ConvertBack csak ha kétféle irányú
  - > Paraméter is átadható az adatkötés helyéről



# IValueConverter implementáció

```
public class StringIsEmptyConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, string language)
        => !string.IsNullOrWhiteSpace(value as string);

    public object ConvertBack(object value, Type targetType, object parameter, string language)
        => throw new NotImplementedException();
}
```

Használat előtt példányosítani kell

1. ✓ Konverter implementálása
2. Konverterből statikus erőforrásként példány létrehozása
3. Adatkötésnél a konverter példány megadása

# IValueConverter felhasználása

- Tipikusan alkalmazás szintű erőforrásként az app.xamlban

The screenshot shows a Windows application window with two main panes. The top pane displays the XAML code for the application's resources:

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>...
            <local:StringIsEmptyConverter x:Key="IsEmptyConverter" />
        <!-- Other app resources here -->
    </ResourceDictionary>
</Application.Resources>
```

The bottom pane shows a button control with its properties and events set up. The Converter property is highlighted in yellow:

```
<Button IsEnabled="{x:Bind Dancer.Name, Mode=OneWay,
    Converter={StaticResource IsEmptyConverter}}>OK</Button>
```

On the right side of the application window, there is a resource editor dialog with the following fields:

Név	IsEmptyConverter
Szerep	<input type="checkbox"/> Szóló <input type="radio"/> Vezető <input checked="" type="radio"/> Követő
OK	Cancel

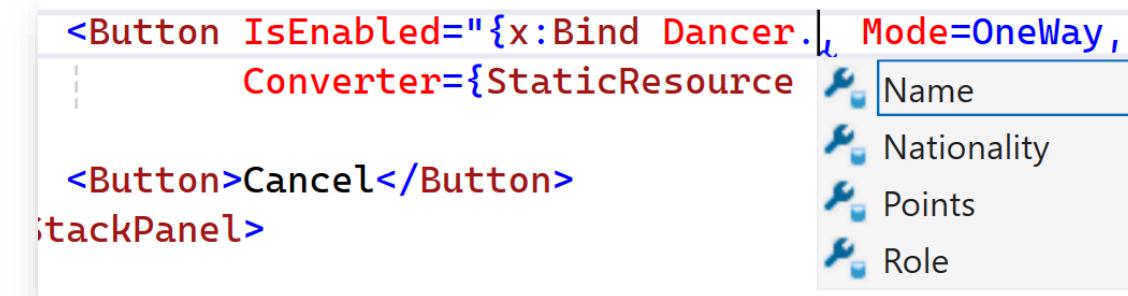
- A Mode-ot is meg kell adni,  
hogy a változást kezelje!

# Megjegyzés: Converter a Windowban ☹

- Convertert a Windowban közvetlenül nem tudunk használni ☹
  - > A Window osztály nem FrameworkElement
  - > Nincs saját Resources gyűjteménye ☹
  - > A generált kód az erőforrások között keresi a konverter példányt ☹
  - > Fordítási hiba lesz ☹
- Workaround: tudd bele a tartalmat egy UserControlba (ami már FrameworkElement) és azt tudd bele a Windowba ☹
  - > <https://github.com/microsoft/microsoft-ui-xaml/issues/6369>

# Binding vs x:Bind

- Azonos szintaxis, hasonló működés
- x:Bind
  - > Erősen, fordítás időben típusos, IntelliSense
  - > Kódgenerált -> gyors
  - > Más alapértelmezett értékek, Mode = OneTime
- Binding
  - > Dinamikusan, explicit állítható az adatforrás – ritkán kell
  - > Másik vezérlő használható forrásként (ElementName) – név alapján x:Bind-dal is rá tudjuk kötni a másik vezérlőt



# Listás adatkötés

- A listát az `ItemsControl . ItemsSource` propertyra kell kötni
  - > A `ListBox`, `ListView` stb. mind az `ItemsControl`ból származik
  - > `List<T>`, `ObservableCollection<T>` leszármazottak
  - > Köthetünk „okosabb” burkoló osztályt, `CollectionView`-t
    - Aktuális elem nyilvántartása
    - Adatlista rendezése, szűrése és csoportosítása
- A kiválasztott elemet a `SelectedItem`re

# Listás adatkötés példa

- A DanceStyles lista a vezérlő code behind osztály tagja, jöhetne egy szerverről is

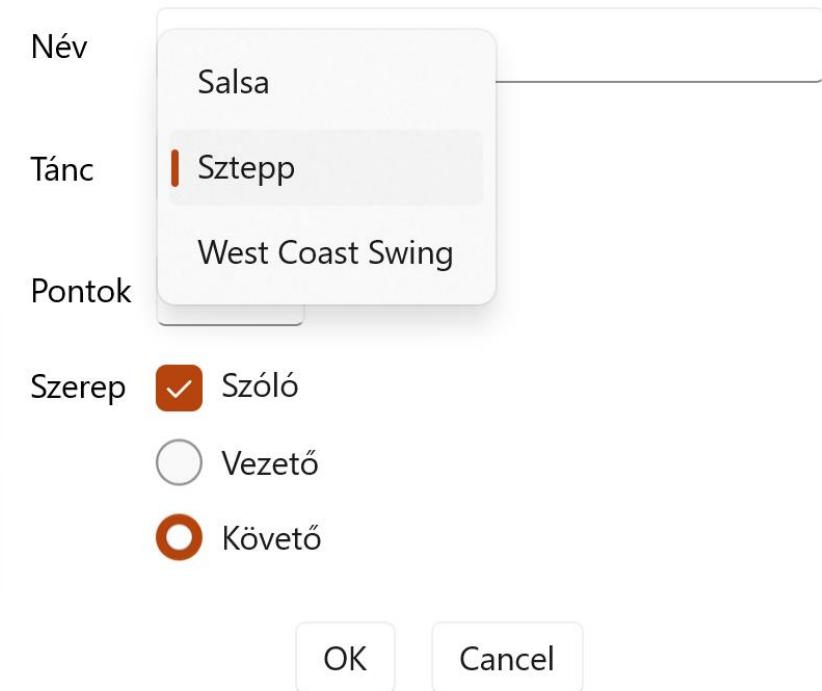
```
<ComboBox Grid.Row="1" Grid.Column="1" ItemsSource="{x:Bind DanceStyles}"  
| SelectedItem="{x:Bind Dancer.DanceStyle, Mode=TwoWay}" Margin="10" />
```

```
public ObservableCollection<string> DanceStyles { get; }  
| = new(new[] { "Salsa", "Sztepp", "West Coast Swing" });
```

- A DanceStyle a Dancer példány tagja

```
public Dancer Dancer { get; set; } = new Dancer();
```

```
public string DanceStyle { get; set; } = "Sztepp";
```



# Adatforrás változás jelzés

- Ha nincs értesítés a .NET-es objektum propertyjének változásáról, akkor a felhasználói felület nem frissül
- Property változása
  - > **INotifyPropertyChanged (INPC)** interfész
  - > **PropertyChanged** event pattern
    - A szövegesen hivatkozott tulajdonság megváltozott
- Gyűjtemény változása
  - > **INotifyCollectionChanged** interfész
    - Új elemek vagy mozgatás/törlés/változás, illetve reset
  - > **ObservableCollection<T>**: INCC

# INPC példa 1 - manuális

```
class Dancer : INotifyPropertyChanged
{
    string name;
    public string Name
    {
        get { return name; }
        set
        {
            if(name == value) return;
            name = value;
            PropertyChanged?.Invoke(this,
                new PropertyChangedEventArgs(nameof(Name)));
        }
    }

    // INPC:
    public event PropertyChangedEventHandler PropertyChanged;
}
```

# INPC példa 2 - Caller Info attribútumok

```
class Dancer : INotifyPropertyChanged
{
    string name;
    public string Name
    {
        get { return name; }
        set
        {
            if(name == value) return;
            name = value;
            OnPropertyChanged();
        }
    }

    void OnPropertyChanged([CallerMemberName] string prop = null)
    {
        PropertyChanged
            ?.Invoke(this, new PropertyChangedEventArgs(prop));
    }
    // INPC:
    public event PropertyChangedEventHandler PropertyChanged;
}
```

The diagram illustrates the flow of the 'prop' parameter. An arrow points from the 'prop' parameter in the `OnPropertyChanged` method to the `name` field in the `Dancer` class. Another arrow points from the 'prop' parameter to the `value` parameter in the `set` block of the `Name` property.

Ha az opcionális paraméter  
(`=null`) nincs megadva, akkor a  
hívó neve ("Name") lesz a *prop*  
paraméter értéke

# INPC példa 3 - még kevesebb kóddal

- Általában kivezetjük ősosztályba

```
protected bool SetProperty<T>(ref T storage, T value,
    [CallerMemberName] String propertyName = null)
{
    if (object.Equals(storage, value)) return false;
    storage = value;
    this.OnPropertyChanged(propertyName);
    return true;
}

// Használata
private string name;
public string Name
{
    get { return name; }
    set { SetProperty(ref name, value); }
}
```

# INPC példa 4 – MVVM toolkittel

- Forráskód generálás
  - > NuGet: CommunityToolkit.MVVM
  - > Részleges osztály, csak mezők deklarálása

```
public partial class Dancer : ObservableObject
{
    [ObservableProperty]
    string name = "Fred Astaire";

    [ObservableProperty]
    string danceStyle = "Sztepp";

    [ObservableProperty]
    int? points = 0;

    [ObservableProperty]
    Role role = Role.Leader | Role.Solo;
```

# Sablonok

WinUI Desktop

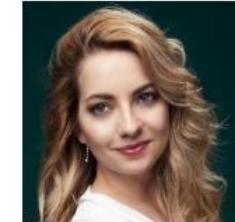
User	Role	Points
Kowalska Izabella	Follower	6
Jakub Jakoubek	Leader	12
Emeline Rocheffeille	Follower	38
Maxence Martin	Leader	92
Virginie Grondin	Follower	24

# Sablonok feladata, típusai

- Azonos típusú adatok egységes megjelenítése
  - > minden adathoz azonos vizuális fa jön létre
  - > A megjelenés paramétereit az objektum tartalmazza
- Vezérlők (például gomb) egységes megjelenése
- A sablonok típusai
  - > Adat sablon (DataTemplate): adat megjelenítése
  - > Vezérlő sablon (ControlTemplate): vezérlő
  - > Listázó panel (ItemsPanelTemplate): elem listázás

# Adat sablon példa: DataTemplate definíció

```
<DataTemplate x:Key="DancerTemplate" x:DataType="local:Dancer">
    <StackPanel Orientation="Horizontal">
        <Image Source="{x:Bind ProfilePicture}" Height="70" />
        <TextBlock Padding="10 5" MaxHeight="70" Width="150">
            <Run FontWeight="Bold" Text="{x:Bind Name}" />
            <LineBreak/>
            <Run Text="{x:Bind Role}" />
            <LineBreak/>
            <Run Text="Points: " />
            <Run Text="{x:Bind Points}" />
        </TextBlock>
    </StackPanel>
</DataTemplate>
public partial class Dancer : ObservableObject
{
    public Dancer(string name, string profilePicture, Role role, int points)
    {
        this.name = name;
        this.profilePicture = profilePicture;
        this.role = role;
        this.points = points;
    }
}
```



**Kowalska Izabella**

Follower

Points: 6



**Jakub Jakoubek**

Leader

Points: 12

# Adat sablon példa



# Adatsablon tipikusan külön fájlban

- Az erőforrás fájl hivatkozása az App.xaml-ből

```
<ResourceDictionary.MergedDictionaries>
    <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls" />
    <local:TemplatesResourceDictionary />
</ResourceDictionary.MergedDictionaries>
```

- Az erőforrás fájlhoz osztály, inicializálással

```
<ResourceDictionary
    x:Class="DancerProfiles.TemplatesResourceDictionary">

    public partial class TemplatesResourceDictionary
    {
        public TemplatesResourceDictionary()
        {
            InitializeComponent();
        }
    }

```

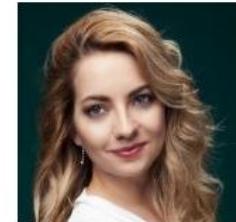
- Adatsablon definiálása, típussal, kulcssal

```
<DataTemplate x:Key="DancerTemplate" x:DataType="local:Dancer">
```

# Adatsablon felhasználása

- minden ContentControl a Content propertyt a sablon alapján tudja megjeleníteni
  - > ContentTemplate tulajdonság

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center" Spacing="10">
    <ContentControl Content="{x:Bind Dancers[0]}"
                    ContentTemplate="{StaticResource DancerTemplate}" />
    <ContentControl Content="{x:Bind Dancers[1]}"
                    ContentTemplate="{StaticResource DancerTemplate}" />
```



**Kowalska Izabella**

Follower

Points: 6



**Jakub Jakoubek**

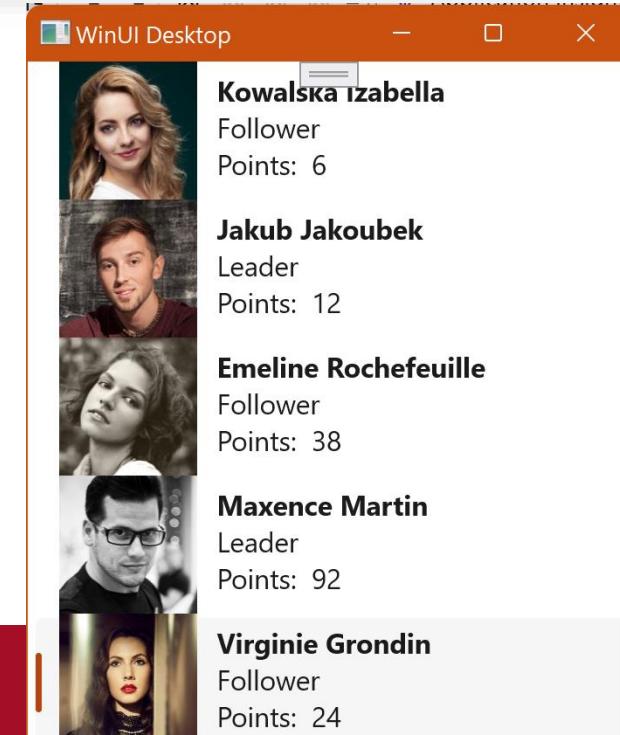
Leader

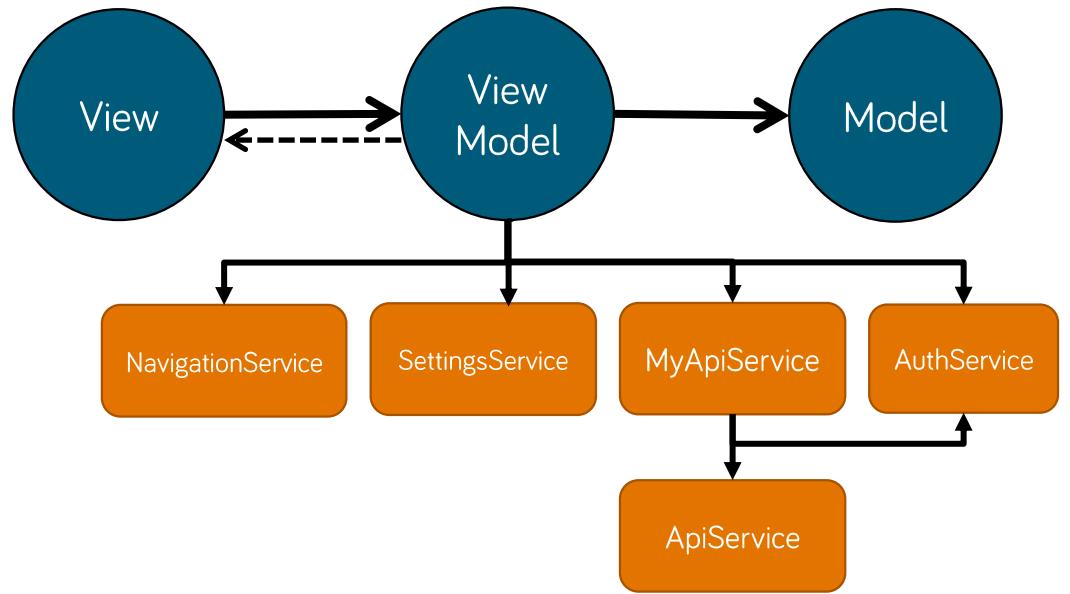
Points: 12

# Adatsablonok lista esetén

- A listák az ItemsControlból származnak
- A sablont az ItemTemplate-tel kell állítani

```
<ListView ItemsSource="{x:Bind Dancers}"  
         ItemTemplate="{StaticResource DancerTemplate}" />
```

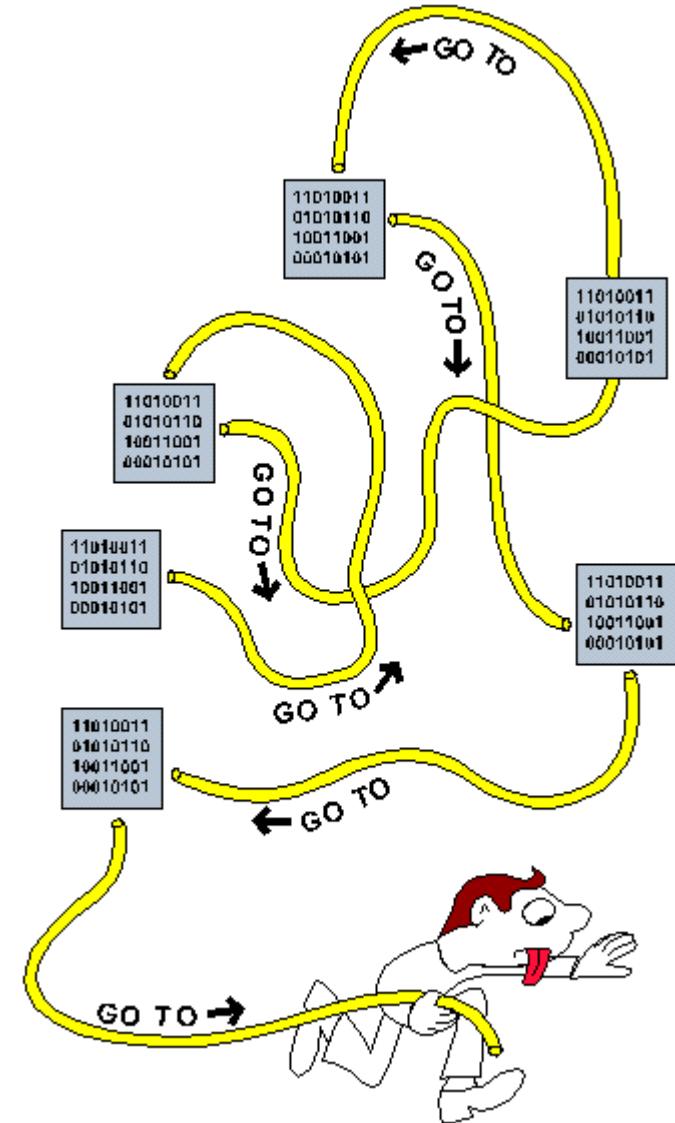




MVVM

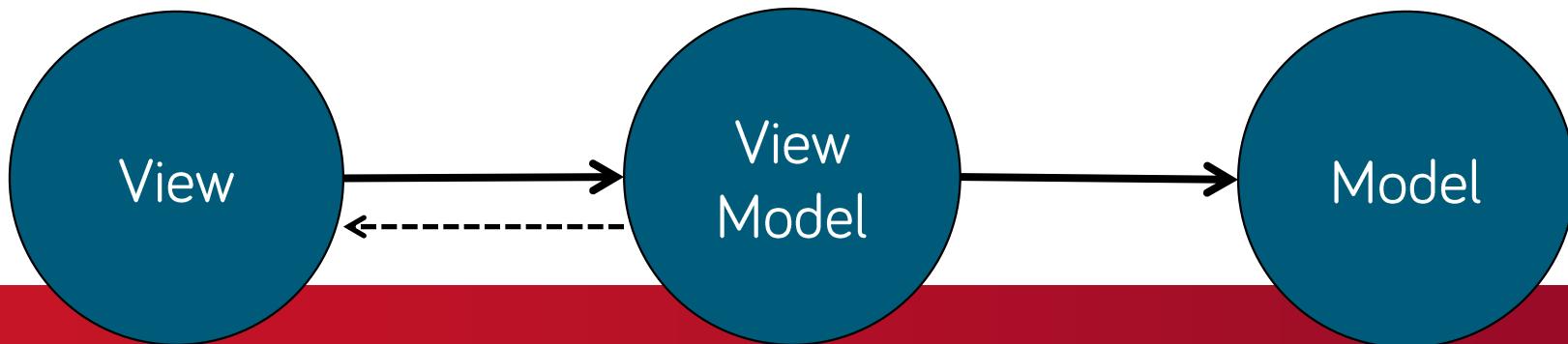
# MVVM minta nélkül

- Hol van a kód?
  - > Page vagy UserControl code-behind
  - > Modell osztályok, üzleti logikát, adathozzáférést végző osztályok, ...
- Probléma:
  - > A code-behindban keveredik az alkalmazás logika a megjelenítéshez tartozó részekkel
- Ez miért gond?
  - > Komplexebb alkalmazásokban áttekinthetetlen
  - > Nehéz karbantartani
  - > Nem unit tesztelhető a logika
  - > Nem válnak el a funkciók (biztos, hogy ugyan az a személy fejleszti a kettőt...?)



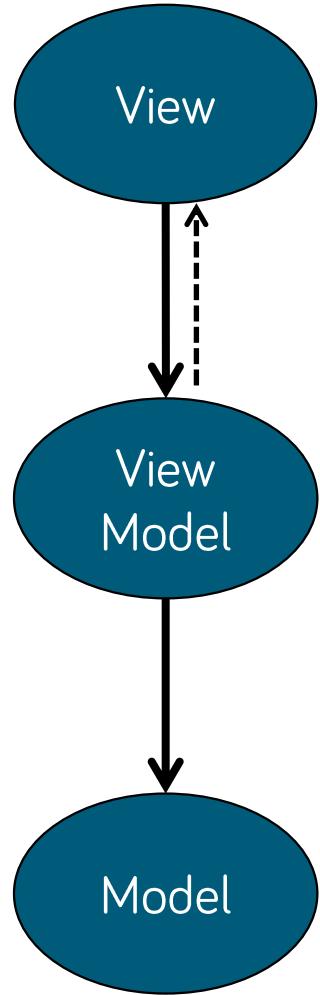
# MVVM minta felépítés

- Megoldás: válasszuk külön az alkalmazáslogikát a megjelenítéstől
- Model:
  - > Domain osztályok, DTO-k
- ViewModel:
  - > Felhasználói felület logika
  - > Felületen megjelenő adatok
  - > Kapcsolódik az alkalmazás logikához
- View:
  - > A megjelenítéshez kapcsolódó részek: nézetek, animáció, színek stb.



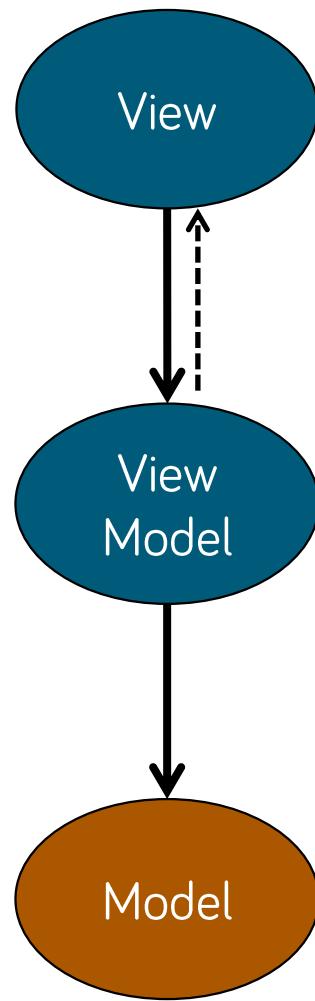
# MVVM általános alapelvek

- Model nem hivatkozik senkire
- ViewModel és Model kapcsolata
  - > Hivatkozás vagy burkolás
- View és ViewModel
  - > A View adatkötéssel hivatkozza a ViewModellt
  - > A ViewModel interfészeken (INPC) keresztül kommunikál a Viewval
  - > Például dialógusablakot kell feldobni a logika hatására -> a View-t egy interfészen keresztül lehet elérni, ami megvalósít egy dialógusablakot



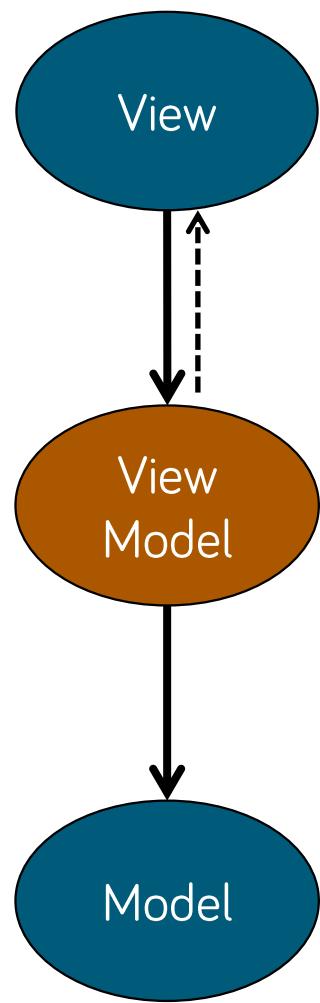
# Model

- Domain- és felületspecifikus adatosztály, DTO
  - > Üzleti objektum, szerver oldali adatokból építjük
- **Tartalmilag a felülethez alkalmazkodjon!**
  - > Ne adjunk át felesleges adatokat!
  - > Egy felület megjelenítéséhez egyszer forduljunk a szerverhez!
  - > A szolgáltatás interfészét, API-t, a UI határozza meg
- Formára legyen független a UI típusuktól stb.
  - > Például ne legyenek benne színek, hanem a szemantikát írjuk le, a színekre fordítás a UI feladata



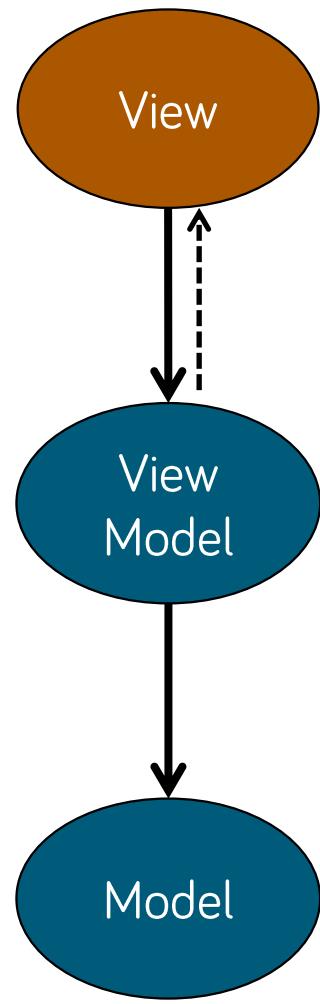
# ViewModel

- A felülethez kapcsolódó logika van benne
  - > UI állapotgép
    - Kiválasztott elem nyilvántartása, elemek eltüntetése, letiltása
    - Állapotgép leírása UI technológiától függetlenül: például Visibility helyett IsVisible
  - > Navigáció
  - > Felhasználói input ellenőrzés és visszajelzés, hibajelzés
  - > Felhasználói események kezelése
    - Ide futnak be a külső, szerver oldali események is, például chat
- A szerver oldalt hívva hozza létre a Model objektumokat
  - > A UI réteg innen kapcsolódik az alkalmazás többi részéhez



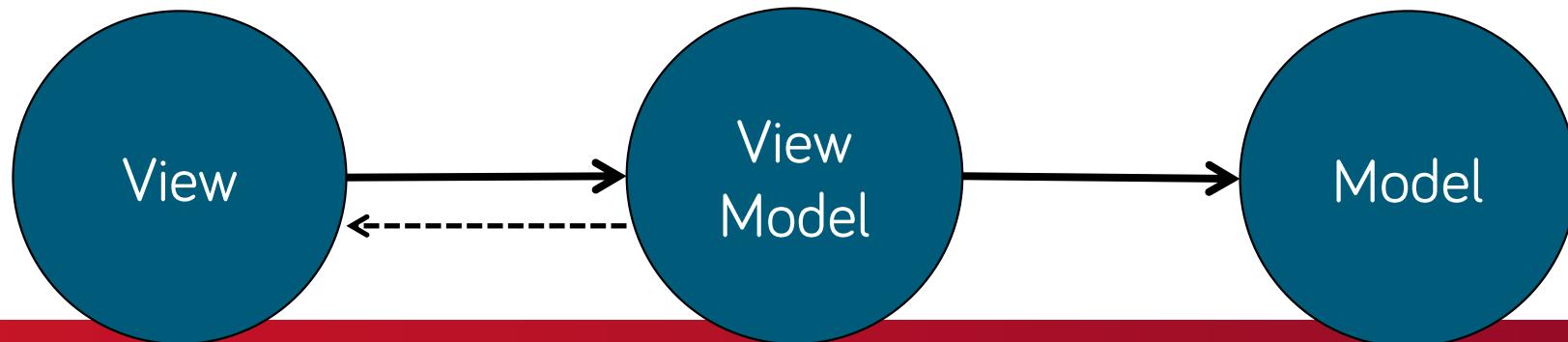
# View

- Megjelenítés, animáció
- "Passzív", deklaratív nézet, XAML-ben
- Adattranszformációk
- Adatkötéssel explicit hivatkozik a ViewModelre és a Modelre
- A code-behind tipikusan üres
  - > Speciális eseménykezelők, workarroundok stb.
- A ViewModel eseményeken vagy interfészen keresztül érheti el
  - > Mikor melyiket érdemes használni?

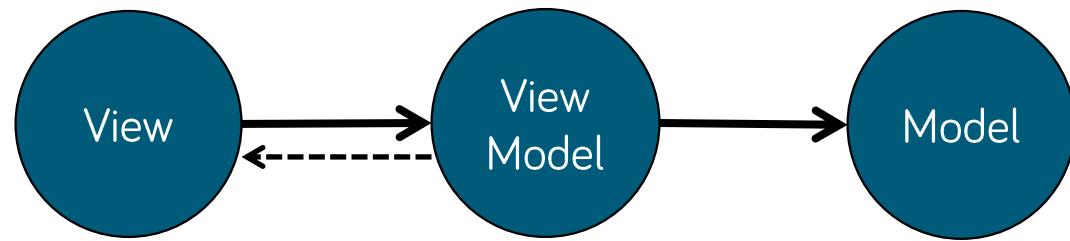


# Strict MVVM

- View nem hivatkozik a Modelre csak a ViewModelre
  - > Az adatkötésekben csak a ViewModel tulajdonságai szerepelnek, a Model tulajdonságai nem, a ViewModel propertyk áthívna a burkolt Model osztályba
- minden Model osztály be van csomagolva ViewModel osztályba
  - > A Model tulajdonságait a ViewModel csomagolja
  - > ViewModel implementálja a felületlogikához szükséges interfészeket, például INPC interfészt, formázást, adattranszformációt
  - > Gyakran kódgenerált
- A Model teljesen független lehet a felületlogikától, DTO-k



# Strict MVVM – beágyazás



```
// model class - no INPC
public class Dancer
{
    public string Name { get; set; }
}
```

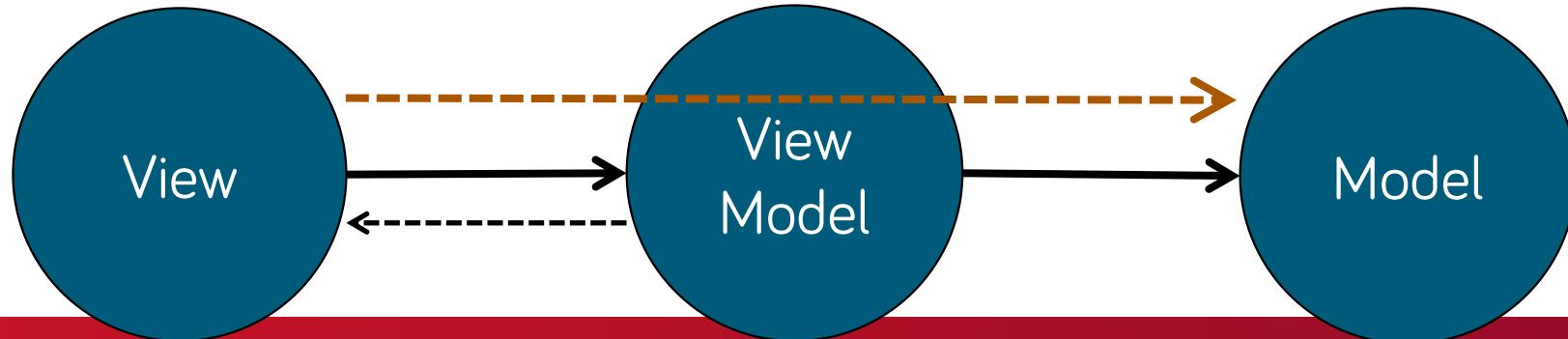
```
// Model is embedded in ViewModel, adds INPC
public class DancerViewModel : ObservableObject
{
    public DancerViewModel(Dancer dancer) { this.dancer = dancer; }
    Dancer dancer;
    public string Name
    {
        get => dancer.Name;
        set => SetProperty(dancer.Name, value, dancer,
            (dancer, value) => dancer.Name = value);
    }
}
```

```
// Page ViewModel uses only ViewModels
public partial class DancerDialogViewModel : ObservableObject
{
    [ObservableProperty]
    DancerViewModel selectedDancerVM;
}
```

```
<DataTemplate x:DataType="local:DancerViewModel">
    <StackPanel Orientation="Horizontal">
        <TextBlock Padding="10 5" MaxHeight="70" Width="150">
            <Run FontWeight="Bold" Text="{x:Bind Name}" />
```

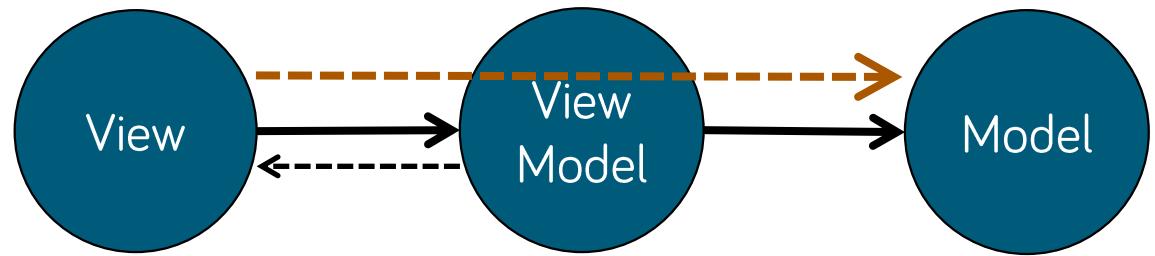
# Relaxed MVVM

- View közvetetten hivatkozik a Modelre az adatkötésben
  - > A Model tulajdonságait közvetlenül kötjük a felületre
- A nézet ViewModelje kiajánlja a Modelt
  - > Publikus tulajdonságokon, listákban elérhető a Model példány
  - > A Modelnek támogatni kell a szükséges interfészeket
    - Például INPC, adattranszformáció stb.
  - > A model generált DTO, tipikusan részleges osztály, kiegészítjük a UI számára szükséges implementációval vagy származtathatók is!



# Relaxed MVVM – hivatkozás

```
// model class - has INPC, properties etc.  
public partial class Dancer : ObservableObject  
{  
    [ObservableProperty]  
    string name;  
}
```



```
// Page ViewModel exposes Models  
public partial class DancerDialogViewModel : ObservableObject  
{  
    [ObservableProperty]  
    Dancer selectedDancer;
```

```
<DataTemplate x:DataType="local:Dancer">  
    <StackPanel Orientation="Horizontal">  
        <TextBlock Padding="10 5" MaxHeight="70" Width="150">  
            <Run FontWeight="Bold" Text="{x:Bind Name, Mode=OneWay}" />
```

# Adattranszformáció helye

- Amikor nem a nyers adatot szeretnénk megjeleníteni, hanem azt valamilyen módon transzformáljuk
  - > Dátum / deviza formázás
  - > `enum` érték színné, ikonná alakítása
  - > `bool` Visibility enummá alakítása
  - > Származtatott értékek
    - Például vezetéknév + keresztnév, cím stb.

# Alternatívák az adattranszformációra

- Az adatosztály (VM vagy M) külön propertykben publikálja az adatokat
  - > ☹ A VM-M rétegekbe megjelenítési adattípusok, például Color kerül
  - > ☺ Tömör, praktikus, jól áttekinthető, kompakt kódbázis
- Metódus a nézeten
  - > ☺ A VM-M rétegek teljesen függetlenek a nézettől
  - > ☹ A V réteg „messzebb” van az adattól
- Konvertert készítünk
  - > ☺ A VM-M rétegek teljesen függetlenek a nézettől
  - > ☹ A konvertereknek jelentős kódolási overheadjük van, „távol” vannak a forrás propertyktől (másik mappa, másik osztály, nem típusos stb.)
  - > -> Konvertert használunk inkább projektfüggetlen típusok átalakítására, például bool -> Visibility enum

# Tényezők

- Architekturális „tisztaság”, elvek
- Kódmennyiség
- Karbantarthatóság
- Átláthatóság

# Hány VM legyen?

- ViewModel minden nézethez: OK.
- ViewModel minden UserControlhoz?
  - > UserControl: összefogja a rajta lévő vezérlőket
  - > Gyakran üres, nincs hozzá tartozó logika
  - > A módosítások eseményekként kerülhetnek kivezetésre amire a tartalmazó nézethez tartozó VM iratkozik fel
- ViewModel minden domain osztályhoz?
  - > Lásd Strict versus Relaxed MVVM
  - > Gyakran csak egyszerű csomagolássá válik
  - > Gyűjtemények/objektum fák esetén hivatkozásonként kell csomagolni, nem triviális!
  - > Főleg többrétegű alkalmazásnál lehet nehézkes

# Hogyan kommunikálunk a Viewval?

- A ViewModel indítja az interakciót a felhasználóval
  - > Dialógus ablak: „Valóban törölni szeretné?”
  - > Hangjelzés
- A View publikál egy interfést
  - > -> A ViewModel meghívja ezt az interfést
- A ViewModel publikál egy eseményt
  - > -> A View feliratkozik az eseményre és reagál
- Tesztelhetőség!

# Eseménykezelők elérése a nézetből

- Feladat: a vezérlő eseményét kössük a ViewModel egy metódusára
- Az **x:Bind** támogatja a metódus meghívását, egyszerű
- **IsEnabled** x:Binddal köthető a ViewModel tulajdonságára, ügyelj az értesítésre!
- A kettőt együtt támogatja a **Command** minta is
- A parancsok összefogására további absztrakció is használható:  
XamlUICommand

# Command minta

- A minta célja: a View réteg vezérlőinek eseményeit közvetlenül kötni a ViewModell metódusaira
  - > x:Bind esetén nem kell, ott beírni a metódus nevét! 😊
  - > Deklaratív, így elkerüljük, hogy a code-behindba írunk
- A Command kétirányú: jelzi, ha a gomb megnyomható és ez az állapota változik
- A gomb az ICommand interfészen keresztül támogatja az események átadását
  - > A többi eseményhez és vezérlőhöz is készíthető csatolt tulajdonság, ami támogatja az ICommandot

# Az ICommand interfész

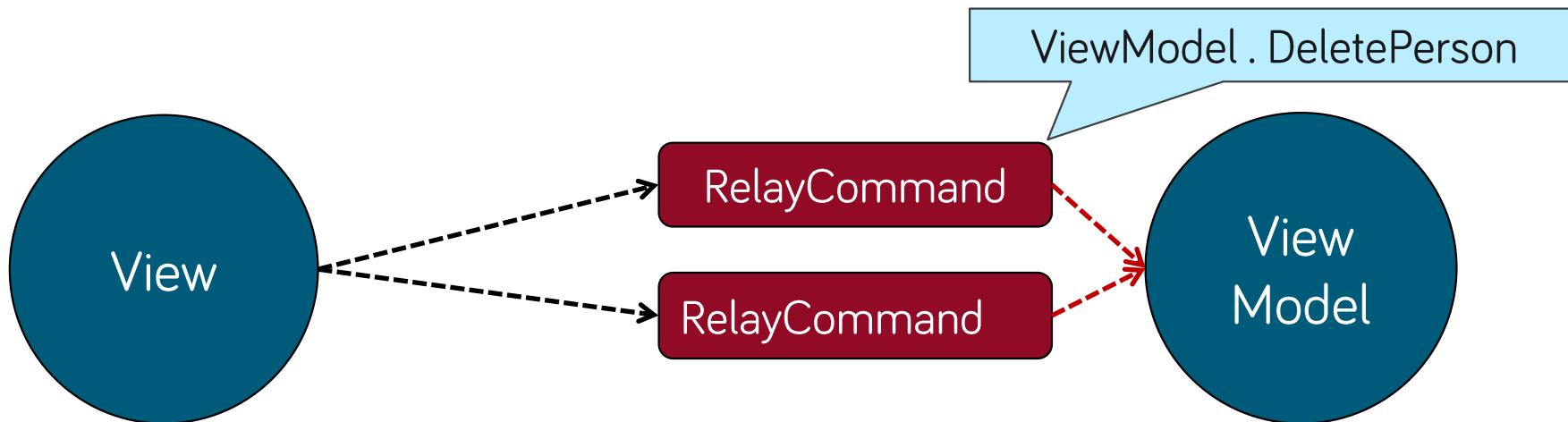
- Egyszerű interfész
  - > Esemény ellövése: Execute
  - > Tiltás és annak változása

```
...public interface ICommand
{
    ...event EventHandler? CanExecuteChanged;

    ...bool CanExecute(object? parameter);
    ...void Execute(object? parameter);
}
```

# Relay/DelegateCommand

- A ViewModel minden eseményhez egy külön híd objektumot publikál
  - > Ez a híd implementálja az ICommandot
  - > A hidat adatkötni kell a Button Click eseményére
  - > A híd az esemény hatására behív a ViewModel-be
  - > Például DeletePersonCommand



# Command példa

- Command publikálása a ViewModelből propertyn keresztül

```
public ICommand SaveDocCommand  
        => new RelayCommand<string>(SaveDoc);  
  
private void SaveDoc(string param)  
{  
    // ...  
}
```

- A Command bekötése a nyomógombra

```
<Button Content="Delete Person"  
       Command="{Binding SaveDocCommand}"  
       CommandParameter="MyParam"/>
```

# MVVM Toolkit támogatás

- Nem kell kézzel létrehozni a Commandot, elég használni a RelayCommand attribútumot

```
[RelayCommand]
private void GreetUser()
{
    Console.WriteLine("Hello!");
}
```

- Ezt generálja hozzá:

```
private RelayCommand? greetUserCommand;

public IRelayCommand GreetUserCommand => greetUserCommand ??= new RelayCommand(GreetUser);
```

# Ha nincs Command támogatás - Behaviors

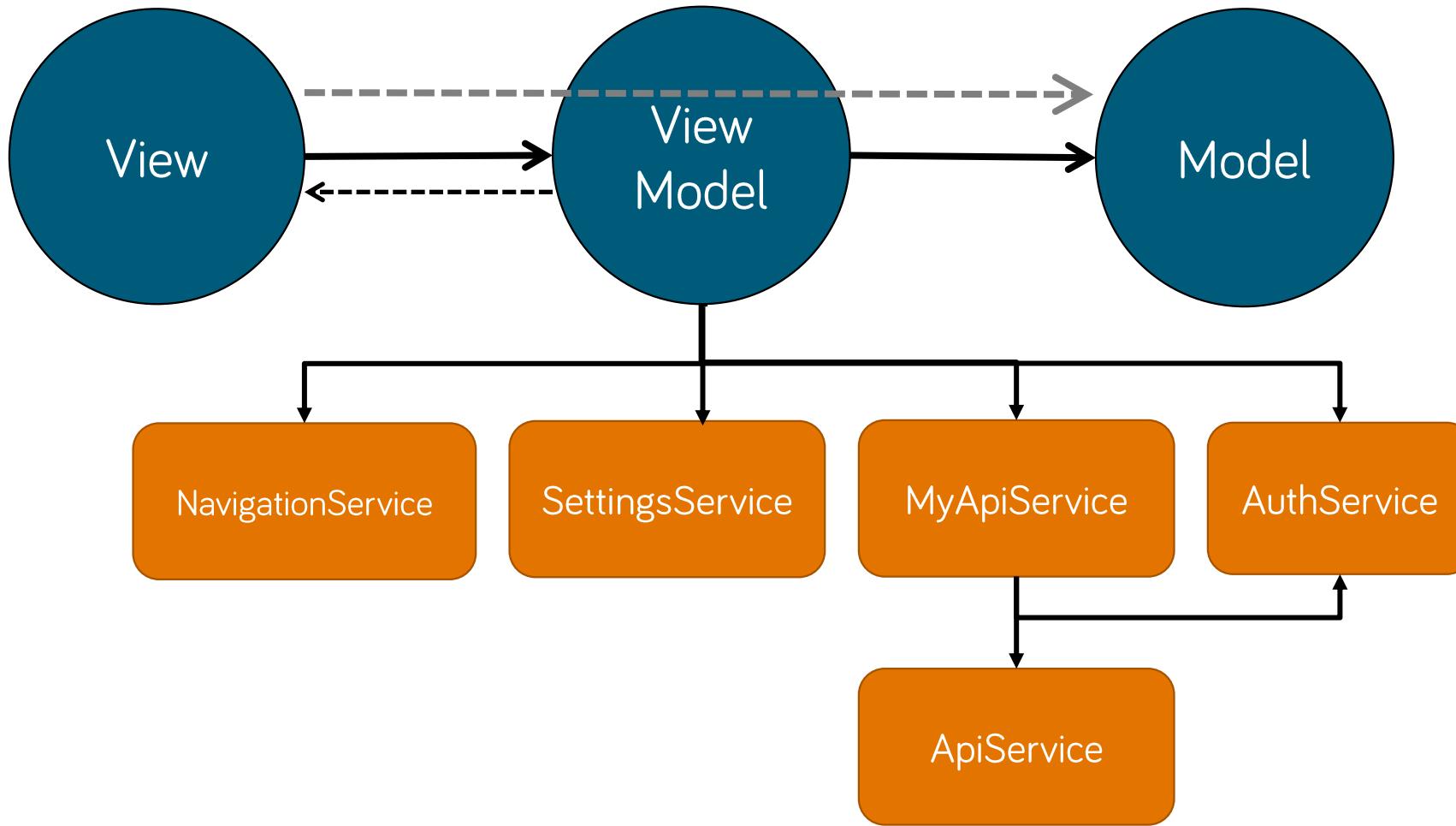
- Csak néhány vezérlő és esemény támogatja a Command mintát
- Vezérlők kiterjesztése Behavior objektumokkal
  - > Hasonló az attached property mechanizmushoz
  - > A [Windows Community Toolkit](#)-ben több kiegészítő is van, például Esemény – Command kapcsolás

```
<TextBlock Text="Delete">
  <i:Interaction.Behaviors>
    <ic:EventTriggerBehavior EventName="Tapped">
      <ic:InvokeCommandAction Command="{Binding SaveDocCommand}" />
    </ic:EventTriggerBehavior>
  </i:Interaction.Behaviors>
</TextBlock>
```

# Szolgáltatások MVVM-ben

- Az MVVM minta csak a UI szétválasztásáról szól!
- Ha csak a ViewModelbe rakkának a teljes üzleti logikát, nem sokban különbözne egy összehányt xaml.cs fájltól
- Szervezd a logika darabjait szolgáltatás osztályokba, rétegekbe
  - > Pl.: Rest API, adatelérés, NavigationService stb.
- A ViewModel ezeket felhasználva működik, próbáld vékonyan tartani a VM-eket!
- Használj Dependency Injectiont!

# Szolgáltatások



# MVVM minta hátrányai, korlátai

- Fő hátrány: implementációs overhead
- Amikor nem érdemes használni:
  - > Nagyon egyszerű alkalmazás
  - > Statikus adatok
  - > Nem hagyományos üzleti alkalmazás

# MVVM minta előnyei

- Áttekinthetőség
- Felelősségek szétválasztása
- Laza csatolás
- Könnyű módosíthatóság
- Tesztelhetőség
  - > Például Unit tesztek a ViewModelhez

# Könyvtárak, alkalmazások

- WinUI galéria alkalmazás
  - > WinUI vezérlők bemutatása, forráskód, példák
- Community Toolkit.Mvvm
  - > Platformfüggetlen Mvvm minta támogatás
- Windows Community Toolkit
  - > Vezérlők, kiegészítők WinUI platformhoz
  - > Windows Community Toolkit Gallery alkalmazás
- További vezérlő csomagok: Telerik, DevExpress

# WinUI 3 Gallery

← WinUI 3 Gallery

Search

- Basic input
- Collections
- Date & time
- CalendarDatePicker**
- CalendarView
- DatePicker
- TimePicker
- Dialogs & flyouts
- ContentDialog
- Flyout
- TeachingTip
- Layout
- Border
- Canvas
- Expander
- ItemsRepeater

**CalendarDatePicker**  
Microsoft.UI.Xaml.Controls

Documentation Source

A control that lets users pick a date value using a calendar.

CalendarDatePicker with a header and placeholder text.

Calendar

Pick a date

January 2024

Su	Mo	Tu	We	Th	Fr	Sa
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	<b>30</b>	31	1	2	3
4	5	6	7	8	9	10

← WinUI 3 Gallery

Search

- Basic input
- Collections
- FlipView
- GridView**
- ItemsView
- ListBox
- ListView
- PullToRefresh
- TreeView
- DataGrid
- Date & time
- Dialogs & flyouts
- ContentDialog
- Flyout

**GridView**  
Microsoft.UI.Xaml.Controls

Documentation Source

The GridView <https://github.com/microsoft/WinUI-Gallery/tree/main/WinUIGallery/ControlPages/GridViewPage.xaml.cs>

Basic GridView with S C# source code

This is a basic GridView that has the full source code below.  
Other samples on this page display only the additional markup needed to customize them.

Source code

# Windows Community Toolkit Gallery

The image shows two overlapping windows of the Windows Community Toolkit Gallery. The left window displays the **RadialGauge** control, which is a circular gauge with a needle pointing towards the top right. The right window displays the **ImageCropper** control, which allows users to crop rectangular and circular images. A snowy owl is shown as the target image for cropping.

**RadialGauge**  
The Radial Gauge Control displays a value.

**ImageCropper**  
Control to crop rectangular and circular images.

**Source code** | **Package info**

**Source code** | **Package info**

**SettingsExpander**

**Media**

**CameraPreview**

**ImageCropper**

**Sizers**

**ContentSizer**

**GridSplitter**

**PropertySizer**

**Sizer overview**

**StatusAndInfo**

**MetadataControl**

**Text**

**RichSuggestBox**

**TokenizingTextBox**

**Extensions**

**Helpers**

**Layouts**

**Xaml**

**Settings**

**Pick image** | **Save** | **Reset**

**BME / UT**

# Kérdések?

Albert István  
[ialbert@aut.bme.hu](mailto:ialbert@aut.bme.hu)