Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése II

9. előadás

WPF erőforrások kezelése

Giachetta Roberto

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

WPF erőforrások kezelése

Erőforrások

- A Windows Presentation Foundation általánosítja az *erőforrás* fogalmát
  - a Windows Forms erőforrások azok a képek, hangok, stb. amelyeket csatolunk az egyes felületi osztályokhoz
  - a WPF-ben erőforrás lehet bármely külső fájl, sőt bármely osztály példánya, elsősorban:
    - *stílusok* (`Style`): a felületi elemek egységes megjelenését definiálják
    - *sablonok* (`Template`): a vezérlők felépülését és adatkötéseit definiálják
    - *forgatókönyvek* (`StoryBoard`): animációk végrehajtását biztosítják

WPF erőforrások kezelése

Erőforrások a felületi kódban

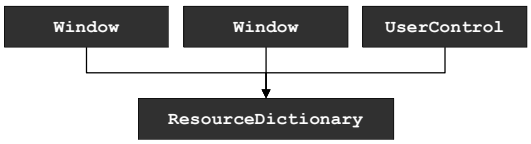
- Bármely felületi elem (`UIElement`) tartalmazhat erőforrásokat a `Resources` tulajdonság segítségével, pl.:

```
<Window … >
   <Window.Resources>
      … <!– erőforrások az egész ablakra -->
   </Window.Resources>
   <Grid Name="LayoutRoot">
      <Grid.Resources>
         … <!-- rácson belüli erőforrások -->
      </Grid.Resources>
      …
   </Grid>
</Window>
```


WPF erőforrások kezelése

Erőforrásfájlok

- Amennyiben több ablak, vagy vezérlő számára biztosítani akarjuk ugyanazt a stílus-, animáció- és sablonkészletet, akkor használhatunk *erőforrásfájl*okat (*Resource Dictionary*)



- csak XAML erőforrásokat tartalmazó fájlok
- használatba vehetőek bármely ablakban és egyedi vezérlőben, vagy akár a teljes alkalmazásban (az `App` osztályon keresztül)

WPF erőforrások kezelése

Erőforrásfájlok

- Pl.:
  erőforrásfájl (`StyleDict.xaml`):

```
<ResourceDictionary … >
   <Style x:Key=… > <!-- stíluselem -->
   …
</ResourceDictionary>
```

  felhasználása egy ablakban (`MainWindow.xaml`):

```
…
<Window.Resources>
   <ResourceDictionary Source="styleDict.xaml" />
   <!-- erőforrásfájl betöltése -->
</Window.Resources>
```


WPF erőforrások kezelése

Erőforrások használata

- Az erőforrás kulccsal (`x:Key`) rendelkezik, amely alapján lekérdezhetjük a `StaticResource` hivatkozással, pl.:

```
<Grid Name="grid">
   <Grid.Resources>
      <Style x:Key="buttonStyle"> … </Style>
      <!-- megadtuk az erőforrás célját -->
   </Grid.Resources>
   …
   <Button Style="{StaticResource buttonStyle}">
```

- Maga a `Resources` tulajdonság egy asszociatív tömb, amely a kulcsok szerint indexelt, pl.:

```
Style myButtonStyle =
   (grid.Resources["buttonStyle"] as Style);
```

WPF erőforrások kezelése

Vezérlők megjelenése

- A vezérlők megjelenése sokféleképpen befolyásolható, a függőségi tulajdonságok állításával, pl.:

```
<Label Content="Hello World" FontSize="20">
   <Label.Background> <!-- háttér -->
      <LinearGradientBrush> <!-- átmenetes -->
         <GradientStop Color="Green" Offset="0"/>
         <GradientStop Color="Red" Offset="1"/>
      </LinearGradientBrush>
   </Label.Background>
   <Label.Effect> <!-- speciális hatások -->
      <DropShadowEffect BlurRadius="40"
         Direction="50" Opacity="1"/>
         <!-- árnyék -->
   …
```


WPF erőforrások kezelése

Stílusok

- A stílusok (`Style`) olyan megjelenési beállítás gyűjtemények, amellyel egyszerre számos elem kinézetét vezérelhetjük
  - a `FrameworkElement` leszármazottaira használhatóak a `Style` függőségi tulajdonságon keresztül
  - lehetővé teszik, hogy vezérlők kinézetét egyszerre kezeljük, teljesen függetlenül az operációs rendszer beállításaitól
  - megadhatóak elemenként, pl.:

```
<Button Content="Blue Button">
  <Button.Style>
     <Setter Target="Foreground" Value="Blue" />
  </Button.Style>
</Button>
```


WPF erőforrások kezelése

Stílusok

- megadhatóak erőforrásként, pl.:

```
<Style x:Key="buttonStyle" TargetType="Button">
   <!-- megadható a céltípus is -->
   <Setter Target="Foreground" Value="Blue" />
</Style>
…
<Button Style="{StaticResource buttonStyle}" />
```

- a stílusoknak két típusát tartjuk nyilván:
  - *implicit*: mennyiben nem adunk meg kulcsot, úgy a stílus az összes megadott típusú elemre érvényes lesz, nem szükséges a `StaticResource` hivatkozás
  - *explicit*: a kulcs megadásával és a `Style` tulajdonság használatával definiáljuk a vezérlő stílusát

WPF erőforrások kezelése

Stílusok

- a stílusokban a `Setter` elem segítségével függőségi tulajdonságokra (`Property`) adunk a típusnak megfelelő értéket (`Value`), pl.:

```
<Style x:Key="buttonStyle" TargetType="Button">
   <Setter Property="Width" Value="400"/>
      <!-- egyszerű érték -->
   <Setter Property="Canvas.Left" Value="200" />
   <Setter Property="RenderTransform">
      <Setter.Value> <!-- összetett érték -->
         <TranslateTransform X="100" Y="50" />
      </Setter.Value>
   </Setter>
</Style>
```


WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

- Dinamikus felhasználói felületet `ItemsControl` vezérlő segítségével tudunk megjeleníteni
  - a megjelenítőt és az elemeket sablonok (`ItemsPanel`, `ItemTemplate`) segítségével adjuk meg
  - az elemek tárolókba kerülnek (`ItemContainer`)
- Amennyiben speciális megjelenítőt használunk, az elemekre függőségi tulajdonságokat alkalmazhatunk az elhelyezésre vonatkozóan
  - pl. `UniformGrid` esetén a `Grid.Row` és `Grid.Column` tulajdonságokkal szabályozhatjuk az elhelyezést

WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

- a függőségi tulajdonságot nem a dinamikus vezérlőn, hanem a tárolóban kell megadnunk, stílus használatával, erre szolgál az `ItemContainerStyle` tulajdonság
- pl.:

```
<ItemsControl ItemsSource="{Binding Fields}">
   <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
         <UniformGrid Rows="5" Columns="5" />
         <!-- egy 5x5-ös rácsot használunk -->
      </ItemsPanelTemplate>
   </ItemsControl.ItemsPanel>
   <ItemsControl.ItemTemplate>
      … <!-- a megjelenített elem -->
   </ItemsControl.ItemTemplate>
```

WPF erőforrások kezelése

Stílusok dinamikus felületű alkalmazásokban

```
<ItemsControl.ItemContainerStyle>
   <!-- az elemek megjelenítési stílusa -->
   <Style>
      <!-- az elemek elhelyezését stílus
           keretében adjuk meg -->
      <Setter Property="Grid.Row"
              Value="{Binding X}" />
      <Setter Property="Grid.Column"
              Value="{Binding Y}" />
   </Style>
</ItemsControl.ItemContainerStyle>
</ItemsControl>
```


WPF erőforrások kezelése

Példa

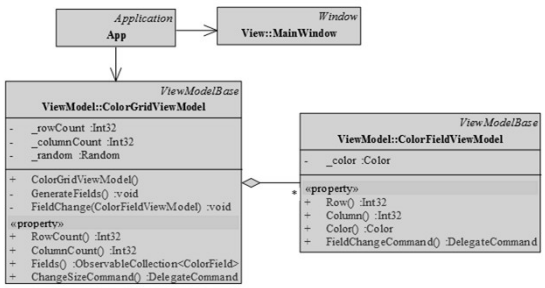
Feladat: Készítsünk egy dinamikus méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a felületen egy **`ItemsControl`** vezérlőben helyezzük el az elemeket, amely egy **`UniformGrid`** segítségével jelenít meg gombokat (**`Button`**)
- a nézetmodell megadja a mező típusát (**`ColorFieldViewModel`**), amely tárolja a sor (**`Row`**), oszlop (**`Column`**), szín (**`Color`**) értékeket, valamint a végrehajtandó utasítást (**`FieldChangeCommand`**), amely paraméterben az egész mezőt megkapja, így a nézetmodell könnyen tudja módosítani a megfelelő elemeket

WPF erőforrások kezelése

Példa

Tervezés:



WPF erőforrások kezelése

Példa

*Megvalósítás (**`MainWindow.xaml`**):*

```
...
<GroupBox Margin="2" Header="Méret:" …>
   <StackPanel Orientation="Horizontal">
      <TextBlock Text="Sorok:" Margin="5" />
      <TextBox Text="{Binding RowCount}" … />
      <TextBlock Text="Oszlopok:" Margin="5" />
      <TextBox Text="{Binding ColumnCount}" … />
      <Button Name="_ChangeSizeButton"
         Command="{Binding ChangeSizeCommand}"
         Content="Méretváltás" Width="80" … />
   </StackPanel>
</GroupBox>
...
```


WPF erőforrások kezelése

Példa

*Megvalósítás (**`MainWindow.xaml`**):*

```
...
<ItemsControl.ItemTemplate>
   <DataTemplate> <!-- megadjuk, milyenek legyenek
                       az elemek -->
      <Button CommandParameter="{Binding}"
         Command="{Binding FieldChangeCommand}">
         <Button.Background>
             <SolidColorBrush
                   Color="{Binding Color}" />
         </Button.Background>
      </Button>
   </DataTemplate>
...
```


WPF erőforrások kezelése

Animációk

- A WPF támogatja animációk végrehajtását, amely lényegében függőségi tulajdonságok adott időn keresztül történő folyamatos módosítását jelenti
  - az animáció típusa megadja a módosítani szánt érték típusát (pl. **`DoubleAnimation`**, **`ColorAnimation`**, **`ThicknessAnimation`**, …)
  - az animációnál definiálnunk kell a kezdőállapotot (**`From`**), a végállapotot (**`To`**), valamint az időt (**`Duration`**)
  - az animáció rendelkezhet tetszőlegesen sok köztes állapottal (**`KeyFrame`**), amelyekre egyéni kritériumok és időkorlátok szabhatóak, valamint megadható az animáció módja (lineáris, diszkrét, spline)

WPF erőforrások kezelése

Animációk

- Az animációkat forgatókönyvekbe (`StoryBoard`) szervezzük
  - a forgatókönyvvel megadható a célobjektum (`Storyboard.Target`, `Storyboard.TargetName`), illetve a céltulajdonság (`Storyboard.TargetProperty`)
  - a céltulajdonság tetszőlegesen összetett lehet, pl.: `Opacity`, `Canvas.Left`, `(Control.Foreground).(SolidColorBrush.Color)`, `(Control.RenderStransform).(TransformGroup.Children[0]).(ScaleTransform.ScaleX)`
  - a forgatókönyvvel szabályozhatjuk a végrehajtást (`Start`, `Stop`) az ismétlődést (`RepeatBehavior`), gyorsulási és lassulási mértéket, esetleg visszajátszást (`AutoReverse`)

WPF erőforrások kezelése

Animációk

- Pl.:

```
<Storyboard Storyboard.TargetName="myButton"
   Duration="0:00:04">
   <!-- forgatókönyv, amely 4 másodpercig fut a
        myButton vezérlőre -->
   <DoubleAnimation From="1" To="0"
      Storyboard.TargetPropety="Opacity" />
      <!-- áttetszővé tesszük -->
   <DoubleAnimation From="100" To="200"
      Storyboard.TargetPropety="Canvas.Left" />
      <!-- eltoljuk jobbra -->
   …
</Storyboard>
```


WPF erőforrások kezelése

Animációk végrehajtása

- Animációk végrehajthatóak kódban, valamint a felületen *trigger*ek segítségével
  - a trigger valamilyen esemény (`EventTrigger`), vagy értékváltozás (`DataTrigger`) hatására képes animációt futtatni (`BeginAnimation`), vagy tulajdonságot beállítani (`Setter`)
  - elhelyezhetőek stílusban, vezérlőben, sablonban, pl.:

```
<Button.Triggers>
   <EventTrigger RoutedEvent="MouseEnter">
      <!-- MouseEnter eseményre fut le -->
      <BeginStoryboard Storyboard="…" />
         <!-- animáció futtatása -->
   </EventTrigger> …
```


WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely véletlenszerű színre állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- adjunk animációt a gombokhoz, amelyben az egér felülhúzására (`MouseEnter`) a gomb elhalványul és összemegy, majd visszaalakul eredeti formájára
- ehhez 3 animáció szükséges (áttetszőség és a két méret)
- a relatív méretezés érdekében a gomboknak a transzformációját (`RenderTransform`) animáljuk, így annak összetett elérési útvonala lesz (pl. (`(Control.RenderTransform).(ScaleTransform.ScaleX)`))

WPF erőforrások kezelése

Példa

Megvalósítás (`MainWindow.xaml`):

```
…
<Window.Resources>
   <Storyboard x:Key="fieldSizeStoryboard"
      Duration="0:0:2" AutoReverse="True">
      <!-- animáció a mezőkre -->
      <DoubleAnimation
         Storyboard.TargetProperty="Opacity"
         From="1" To="0"/>
      <DoubleAnimation Storyboard.TargetProperty="
         (Control.RenderTransform).
         (ScaleTransform.ScaleX)" From="1"
         To="0.5" />
…
```


WPF erőforrások kezelése

Példa

Megvalósítás (`MainWindow.xaml`):

```
…
<DataTemplate>
   <Button … >
      …
      <Button.Triggers>
         <!-- eseményre történő animálás -->
         <EventTrigger RoutedEvent="MouseEnter">
            <BeginStoryboard
               Storyboard="{StaticResource
                     fieldSizeStoryboard}" />
         </EventTrigger>
      </Button.Triggers>
      …
```

WPF erőforrások kezelése

Megjelenítés befolyásolás

- A triggerek akkor is hasznosak, ha a megjelenítést akarjuk szabályozni a nézetmodell adatai alapján, pl.:

```
<Style TargetType="Button">
   <!-- stílus gombokra -->
   <Style.Triggers>
      <!-- a szín adatkötés hatására változik -->
      <DataTrigger Binding="{Binding FieldText}"
                   Value="">
         <!-- ha nincs szöveg megadva -->
         <Setter Property="Background"
                 Value="Gray" />
         <!-- a gomb szürke lesz -->
         </DataTrigger>
      </Style.Triggers>
```


WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely három szín között (piros, fehér, zöld) állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a színt a nézet adja meg, így a nézetmodell nem adhat vissza konkrét színt, csak egy sorszámot (0 és 2 között), amely alapján a szín állítható (`ColorNumber`)
- a színt trigger segítségével állítjuk a nézetben, a gomb stílusában, amely az érték függvényében színezi a gombot, (a gomb emellett animálódik, így `DataTrigger` és `EventTrigger` is hatni fog a vezérlőre)
- a triggereket az ablak erőforrásaként megadott stílusban hozzuk létre

WPF erőforrások kezelése

Példa

Megvalósítás (`MainWindow.xaml`):

```
…
<Style x:Key="buttonStyle" TargetType="Button">
   <Style.Triggers>
      <!-- a színezés a nézetmodellben lévő adat
           függvényében fog változni -->
      <DataTrigger Binding="{Binding ColorNumber}"
                   Value="0">
         <Setter Property="Background"
                 Value="Green" />
      </DataTrigger>
      …
</Style>
…
```


WPF erőforrások kezelése

Példa

Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- javítsuk a megjelenítést azáltal, hogy karakterek helyett grafikus alakzatokat (`Line`, `Ellipse`, `Rectangle`) jelenítünk meg a nézetben
  - a karakterek hatására változnak az elemek `DataTrigger` segítségével (amely a lehetséges `Player` értékeket figyeli)
- ugyanakkor továbbra is gombokat jelenítünk meg (amely kattintható), de felüldefiniáljuk a sablont (`Template`) egy egyedi felépítéssel (`ControlTemplate`), így a gomb megjelenése teljesen más lesz

WPF erőforrások kezelése

Példa

Megvalósítás (`TicTacToeWindow.xaml`):

```
<Style.Triggers>
   <DataTrigger Binding="{Binding Player}"
                Value="O">
   <Setter Property="Template">
      <!-- a gomb sablonját cserélgetjük -->
      <Setter.Value>
         <ControlTemplate>
            <Canvas Background="White">
               <Ellipse … />
            </Canvas>
         </ControlTemplate>
      </Setter.Value>
   …
```