Object-oriented programming - lab in .NET environment

Lecture 07

Themes

- Resources
- Animations
- Styles
- *Localization
- *User controls

Resources

- They give us the ability to save data for a particular control, window or application
- Any .NET objects can be saved

```
<Window x:Class="Predavanje6.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:Predavanje6"
        xmlns:clr="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <clr:String x:Key="cs">Server=.;Database=AdventureWorksOBP;Uid=sa;Pwd=SQL</clr:String>
    </Window.Resources>
    <Grid>
        <TextBlock Text="{StaticResource cs}" />
    </Grid>
</Window>
```

Resources

- We can define them at the element, window or application level
- When defining, we set the attribute **`x:Key`**
  - In XAML:
    - As attribute value - we use extension `StaticResource`
      - The value is loaded after the XAML is loaded, and changing the value during the lifetime of the application will not manifest in the application
    - `DynamicResource`
      - The value is loaded when an attempt is made to retrieve it, and a change in the value during the lifetime of the application will be manifested in the application
    - As a new element - `StaticResourceExtension`
  - In the code we use a method `FindResource()` defined on each element

Examples of resource usage

```xml
<Window x:Class="Predavanje6.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:Predavanje6"
        xmlns:clr="clr-namespace:System;assembly=mscorlib"
        Background="{DynamicResource AppBackground}"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <clr:String x:Key="cs">Server=.;Database=AdventureWorksOBP;Uid=sa;Pwd=SQL</clr:String>

        <x:Array x:Key="cbOpcije" Type="clr:String">
            <clr:String>Opcija 1</clr:String>
            <clr:String>Opcija 2</clr:String>
            <clr:String>Opcija 3</clr:String>
        </x:Array>

        <LinearGradientBrush x:Key="AppBackground" StartPoint="0.5,0" EndPoint="0.5, 1">
            <GradientStop Color="DarkBlue" Offset="0"/>
            <GradientStop Color="LightBlue" Offset="1"/>
        </LinearGradientBrush>
    </Window.Resources>
    <Grid>
        <TextBlock Foreground="White" Margin="10">
            <TextBlock.Text>
                <StaticResource ResourceKey="cs"/>
            </TextBlock.Text>
        </TextBlock>
        <ComboBox ItemsSource="{StaticResource cbOpcije}" Width="100" Height="20"/>
    </Grid>
</Window>
```

Animations

- We can animate any dependent property for which there is a corresponding animation type (there are over 40 built-in animation types)
- WPF has three built-in animation types:
  - Interpolation
    - We define the initial and final value of the dependent property and the duration of the animation
  - Keyframe animations
    - We define key points that WPF then connects for a given duration
  - Path-based animations
    - They mostly serve to move the element along the given path (*advanced animations)

Embedded animations

- According to the data type of the dependent property that we want to animate, we must choose the appropriate animation
- WPF comes with a number of built-in animations:
  - `DoubleAnimation`
  - `ColorAnimation`
  - `Int32Animation`
  - `PointAnimation`
  - `ThicknessAnimation`
  - …

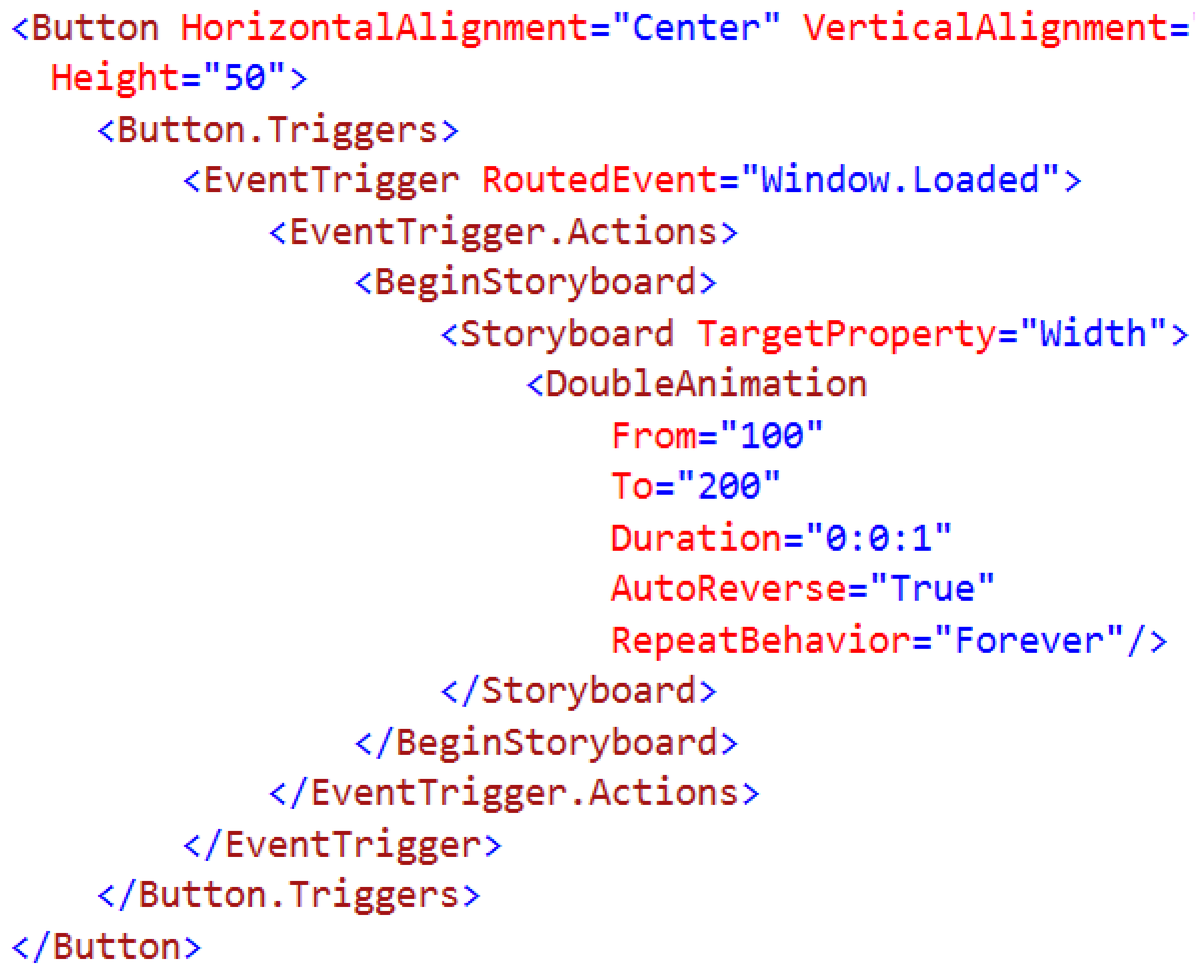
Defining animation

- Each animation supports the following properties:
  - **`Storyboard.TargetProperty`**: the dependent property we're animating
  - **`From`**: initial value of the dependent property we are animating
  - **`To`**: the final value of the dependent property we are animating
  - **`Duration`**: animation duration in h:m:s.ms format
  - **`AutoReverse`**: the animation also takes place backwards or not
  - **`RepeatBehavior`**: how many times the animation should be repeated (1x, 2x, ..., Forever)

Storyboard accommodation

- Animations are always defined within `Storyboard` object
- `Storyboard` defines a dependent property to be animated
- `BeginStoryboard` defines when the animation starts
  - It is usually triggered after some event
    - We put it as a content of `EventTrigger`
    - `EventTrigger` can be placed inside `Triggers` collections of any element

An example of a simple animation



```
<Button HorizontalAlignment="Center" VerticalAlignment=
  Height="50">
    <Button.Triggers>
        <EventTrigger RoutedEvent="Window.Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard TargetProperty="Width">
                        <DoubleAnimation
                            From="100"
                            To="200"
                            Duration="0:0:1"
                            AutoReverse="True"
                            RepeatBehavior="Forever"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Examples of more complex animations

```xml
<Button HorizontalAlignment="Center" VerticalAlignment="Top" Content="Klikni me" Width="100"
  Height="50">
    <Button.Background>
        <LinearGradientBrush StartPoint="0, 0.5" EndPoint="1, 0.5">
            <GradientStop Offset="0" Color="DarkBlue"/>
            <GradientStop Offset="0" Color="White"/>
            <GradientStop Offset="1" Color="DarkBlue"/>
        </LinearGradientBrush>
    </Button.Background>
    <Button.Triggers>
        <EventTrigger RoutedEvent="Button.Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard TargetProperty="Background.GradientStops[1].Color">
                        <ColorAnimation
                            From="White"
                            To="Yellow"
                            Duration="0:0:1"
                            AutoReverse="True" RepeatBehavior="Forever"/>
                    </Storyboard>
                </BeginStoryboard>
                <BeginStoryboard>
                    <Storyboard TargetProperty="Background.GradientStops[1].Offset">
                        <DoubleAnimation
                            To="1"
                            Duration="0:0:1"
                            AutoReverse="True" RepeatBehavior="Forever"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Styles

- A style contains a series of settings that apply to the desired elements
  - The goal is to standardize the appearance of the elements and enable simple changes
  - They can be defined at the control, window, or application level
- A style usually consists of:
  - Setters that set the values of dependent properties
  - Triggers that react to an event and trigger animations (`EventTrigger`) or set properties (`Trigger`)

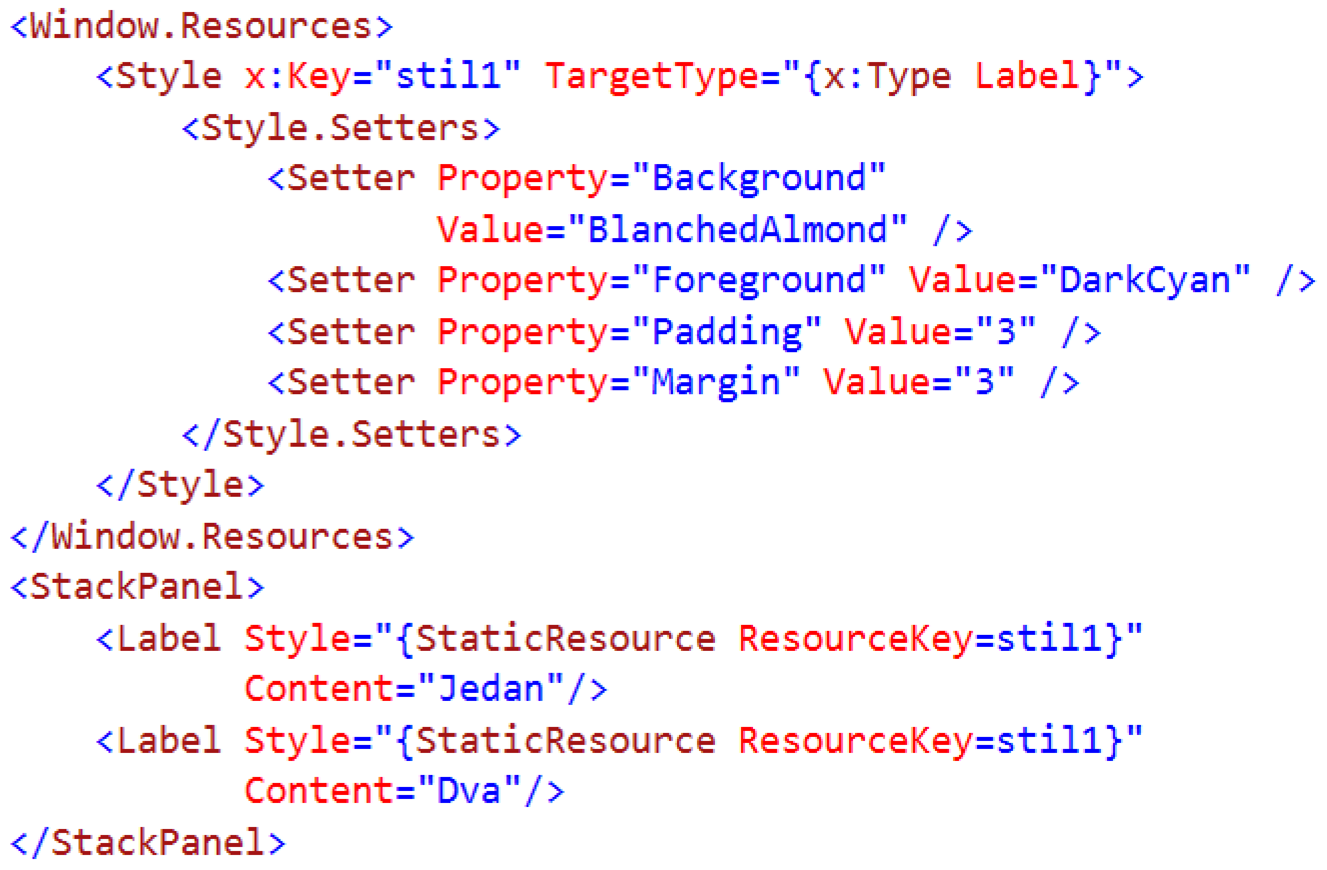
Applying style to *child* elements

- It applies to all *child* elements (not only direct)

```
<StackPanel Margin="10">
    <StackPanel.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="Foreground" Value="Blue" />
            <Setter Property="FontSize" Value="21" />
        </Style>
    </StackPanel.Resources>
    <TextBlock>Prva linija teksta</TextBlock>
    <TextBlock>Druga linija teksta</TextBlock>
    <TextBlock Foreground="Red">
        Svojstvo elementa ima veći prioritet od primijenjenog stila
    </TextBlock>
</StackPanel>
```

Apply style by name

- Style is usually defined as a resource
    - **`x:Key`**– by defining a key, the style must be explicitly applied to the individual control
    - Most often, the **TargetType** is also defined on the corresponding element type



```
<Window.Resources>
    <Style x:Key="stil1" TargetType="{x:Type Label}">
        <Style.Setters>
            <Setter Property="Background"
                    Value="BlanchedAlmond" />
            <Setter Property="Foreground" Value="DarkCyan" />
            <Setter Property="Padding" Value="3" />
            <Setter Property="Margin" Value="3" />
        </Style.Setters>
    </Style>
</Window.Resources>
<StackPanel>
    <Label Style="{StaticResource ResourceKey=stil1}"
           Content="Jedan"/>
    <Label Style="{StaticResource ResourceKey=stil1}"
           Content="Dva"/>
</StackPanel>
```

Apply style by type

- If we omit **`x:Key`** when defining the style, it refers to all elements of the defined type

```
<Window.Resources>
    <Style TargetType="{x:Type Label}">
        <Style.Setters>
            <Setter Property="Background"
                    Value="BlanchedAlmond" />
            <Setter Property="Foreground" Value="DarkCyan" />
            <Setter Property="Padding" Value="3" />
            <Setter Property="Margin" Value="3" />
        </Style.Setters>
    </Style>
</Window.Resources>
<StackPanel>
    <Label Content="Jedan"/>
    <Label Content="Dva"/>
</StackPanel>
```

Applying multiple styles

- One style can be based on another style and thus achieve the effect of inheritance

```
<Window.Resources>
    <Style x:Key="s1" TargetType="{x:Type Label}">
        <Setter Property="Background"
                Value="BlanchedAlmond" />
        <Setter Property="Foreground" Value="DarkCyan" />
        <Setter Property="Padding" Value="3" />
        <Setter Property="Margin" Value="3" />
    </Style>
    <Style x:Key="s2"
           TargetType="{x:Type Label}"
           BasedOn="{StaticResource s1}">
        <Setter Property="FontSize" Value="25" />
    </Style>
</Window.Resources>
<StackPanel>
    <Label Style="{StaticResource ResourceKey=s1}"
           Content="Jedan"/>
    <Label Style="{StaticResource ResourceKey=s2}"
           Content="Dva"/>
</StackPanel>
```

Applying style triggers

- The style often also contains triggers (collection `Triggers`)
    - Object **`Trigger`** allows setting an array of values when the default property takes on the default value

```
<StackPanel>
    <Button>
        <Button.Style>
            <Style TargetType="Button">
                <Setter Property="Foreground" Value="Blue"/>
                <Setter Property="Content" Value="Ja sam gumb"/>
                <Style.Triggers>
                    <Trigger Property="IsMouseOver" Value="True">
                        <Setter Property="Foreground" Value="Green"/>
                        <Setter Property="Content" Value="Klikni me" />
                    </Trigger>
                </Style.Triggers>
            </Style>
        </Button.Style>
    </Button>
</StackPanel>
```

Applying style triggers

- **`EventTrigger`** enables the use of animations when a given event occurs

```
<Style TargetType="{x:Type Label}">
    <Style.Setters>
        <Setter Property="Background" Value="BlanchedAlmond" />
    </Style.Setters>
    <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Background" Value="BurlyWood" />
        </Trigger>
        <EventTrigger RoutedEvent="MouseLeave">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation To="5" AutoReverse="True"
                                     Storyboard.TargetProperty="FontSize"
                                     Duration="0:0:0.3" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Style.Triggers>
</Style>
```