Objektno orijentirano programiranje - praktikum u .NET okolini

Predavanje 05

Windows Presentation Foundation

- **WPF** je primarna Microsoftova tehnologija za stvaranje grafičkog korisničkog sučelja (GUI)
- Glavni cilj:
  - Odvojiti korisničko sučelje od programske logike
- Osnovne značajke WPF-a:
  - Naglasak na vizualnoj komponenti aplikacije
  - Deklarativno programiranje (**XAML** - Extensible Application Markup Language)
    - Koristi se za opis korisničkog sučelja na deklarativan način
    - Glavni cilj je olakšati suradnju programera s ekspertima iz drugih polja (npr. UI dizajneri)
  - Neovisnost o rezoluciji
  - Hardverska akceleracija (za iscrtavanje koristi DirectX)
  - Prilagodljivost

Struktura inicijalnog WPF projekta

- Dependencies
- AssemblyInfo.cs
- **App.xaml** - ***App.xaml.cs*** – deklarativno opisuje što pokreće Main + događaji na aplikativnoj razini

```
<Application x:Class="Primjer.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:Primjer"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

- **MainWindow.xaml** - ***MainWindow.xaml.cs*** – korisničko sučelje i događaji na razini prozora

Deklarativno i proceduralno

- *Gotovo* sve što se može napraviti XAML-om može se napraviti i s preferiranim .NET proceduralnim jezikom
- Kao što je to učinjeno s konfiguracijom Maina, paradigma se nastavlja na izgradnju GUI-ja
- XAML (*object element*):

```
<Button
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Click="Button_Click"
    Content="Button" />
```

- C#:

```
Button b = new Button();
b.Content = "Button";
b.Click += Button_Click;
```

- Definiranje atributa (*property attribute* ili *event attribute*) je identično definiranju svojstva, odnosno događaja na objektu

Imenski prostori

- Naziv elementa (npr. Button) je naziv klase – ali iz kojeg imenskog prostora?
- Mapiranje XAML imenskih prostora s .NET imenskim prostorima ugrađeno je unutar WPF sklopa (*assembly*)
- Ishodišni element XAML datoteke mora definirati najmanje jedan (podrazumijevani) imenski prostor kako bi definirao sebe i ostale elemente djecu
- Uključuje niz .NET imenskih prostora u kojima se nalaze sve osnovne WPF klase

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

Imenski prostori

- XAML datoteke koriste imenski prostor s prefiksom x

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

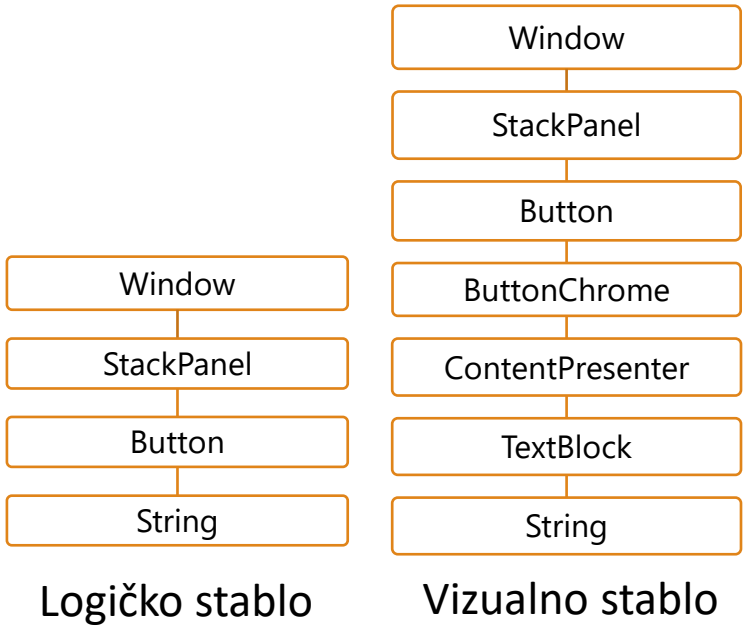
  - Uključuje niz .NET imenskih prostora u kojima se nalaze sve osnovne XAML klase
  - *clr-namespace*: deklariran unutar assembly `xmlns:local="clr-namespace:Primjer"`
- **Odnos XAML-a i *code-behinda***
  - Segment XAML dokumenta: `<Window x:Class="WpfApplication1.MainWindow"> </Window>`
    - Rekli smo da želimo instancu klase `MainWindow` koja nasljeđuje `Window`
  - Segment *code-behind* dokumenta:

```
namespace WpfApplication1
{
    public partial class MainWindow : Window
```

Logička i vizualna stabla

- **Logičko stablo**: skup elemenata definiranih u XAML-u
- **Vizualno stablo**: proširena verzija logičkog stabla u kojoj se svaki element proširi u sastavne dijelove
- Primjerice, ako imamo XAML:

```
<StackPanel>
    <Button Padding="3"
            Margin="3"
            Content="Gumb"/>
</StackPanel>
```



Elementi kao svojstva (*property elements*)

- Jedna od osnovnih značajki WPF-a je **kompozitnost kontrola**, npr. sadržaj gumba ne mora biti samo tekst:

```
Rectangle r = new Rectangle();
r.Width = 50;
r.Height = 50;
r.Fill = Brushes.Black;

Button b = new Button();
b.Content = r;
```

- Isto je moguće napraviti u XAML-u korištenjem elemenata kao svojstva (*property elements*):

```
<Button x:Name="btn">
    <Button.Content>
        <Rectangle Fill="Black" Width="50" Height="50" />
    </Button.Content>
</Button>
```

Elementi kao svojstva (*property elements*)

- Svojstvo `Content` je umjesto atributa postavljeno pomoću XAML elementa
- Unutar `Button.Content` točka je ono što radi razliku između elementa kao objekta i elementa kao svojstva
- Uvijek su u formatu `ClassName.PropertyName`

```
<Button x:Name="btn">
    <Button.Content>
        <Rectangle Fill="Black" Width="50" Height="50" />
    </Button.Content>
</Button>
```

Elementi kao svojstva (*property elements*)

- Mogu se koristiti i kod definiranja jednostavnog sadržaja:

```
<Button Background="Aqua" Content="Klikni me"/>

<!-- ili -->

<Button>
    <Button.Background>
        Aqua
    </Button.Background>
    <Button.Content>
        Klikni me
    </Button.Content>
</Button>
```

Pretvarači tipova (*type converters*)

- Iz prethodnog primjera može se zaključiti da se svojstva čije su vrijednosti različitog tipa od `string` ili `object` postavljaju korištenjem `string` vrijednosti
- To je moguće zbog implicitnog pretvaranja u odgovarajući tip korištenjem *type convertera*
- WPF nudi konvertere za većinu uobičajenih tipova (`Brush`, `Color`, `Font`, ...)
  - To su klase koje nasljeđuju `TypeConverter` (`BrushConverter`, `ColorConverter`, `FontConverter`, ...)
- Bez *type convertera* morali bi koristiti elemente kao svojstva:

```
<Button>
    <Button.Background>
        <SolidColorBrush Color="Aqua" />
    </Button.Background>
</Button>
```

Pretvarači tipova (*type converters*)

- U prethodnom primjeru korišten je `Color` *type convetrer*
- Da ga nema, morali bi svojstvo definirati na sljedeći način:

```
<Button.Background>
    <SolidColorBrush>
        <SolidColorBrush.Color>
            <Color A="255" R="255" G="0" B="0" />
        </SolidColorBrush.Color>
    </SolidColorBrush>
</Button.Background>
```

- Navedeni način je moguće koristiti zato što postoji *type converter* koji može pretvoriti tip `string` u `byte` kojeg se očekuje na `A`, `R`, `G` i `B` vrijednostima

Označna proširenja (*markup extensions*)

- Predstavljaju XAML tehniku za dobivanje vrijednosti koje nisu primitivnog tipa niti specifičnog XAML tipa
  - npr. želimo promijeniti pozadinu kontrole u gradijentnu boju korištenjem string vrijednosti
- Kada je vrijednost atributa zatvorena unutar vitičastih zagrada, XAML parser tu vrijednost tretira kao označno proširenje (*markup extension*)
- Unutar `System.Windows.Markup` imenskog prostora (zato se koristi prefiks **x**) nalazi se nekoliko ugrađenih *markup extension* klasa (prema konvenciji sufiks *extension* može se maknuti iz naziva)

Označna proširenja (*markup extensions*)

```
<Button Background="{x:Null}"
        Height="{x:Static SystemParameters.IconHeight}"
        Content="Klikni me" />
```

- `NullExtension` omogućuje da `Background` svojstvo ima vrijednost `null` što inače nije podržano od `BrushConverter` klase
- `StaticExtension` klasa omogućuje korištenje vrijednosti statičkih svojstava objekata
- U primjeru je visina `Button` kontrole postavljena na vrijednost visine sistemskih ikona, a ona je dobivena iz statičke vrijednosti svojstva `IconHeight` na klasi `SystemParameters`

Kreiranje vlastitog označnog proširenja

- Klasa mora nasljeđivati `MarkupExtension`

```
public class MojExtension : MarkupExtension
{
    0 references
    public MojExtension() { }

    0 references
    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return "Pozdrav";
    }
}
```

- Prilikom korištenja vlastitog označnog proširenja mora se navesti imenski prostor

```
<Grid xmlns:prefiks="clr-namespace:WpfApplication1">
    <Label Content="{prefiks:Moj}" />
</Grid>
```

Kontrole s jednim djetetom

- Pojedinim WPF kontrolama možemo dodijeliti jedan objekt kao njihov sadržaj (*content controls*)
- Obično se sadržaj može dodijeliti kroz svojstvo `Content` ili kao dijete, primjerice:

```
<Button Content="Klikni me"/>
<!-- ili -->
<Button>
    Klikni me
</Button>

<Button>
    <Button.Content>
        <Rectangle Width="100" Height="100" Fill="Blue"/>
    </Button.Content>
</Button>
<!-- ili -->
<Button>
    <Rectangle Width="100" Height="100" Fill="Blue"/>
</Button>
```

Kontrole s više djece

- Pojedine WPF kontrole mogu imati više objekata kao sadržaj
  - Primjerice `ComboBox`, `ListBox`, `TabControl`, ...
  - Svaki objekt može biti neka kontrola ili neki drugi objekt
- Obično se sadržaj može dodijeliti kroz svojstvo `Items` ili kao više djece (`Items` je *content property* za npr. `ListBox`), primjerice:

```
<ListBox>
    <ListBox.Items>
        <ListBoxItem Content="Opcija 1"/>
        <ListBoxItem Content="Opcija 2"/>
    </ListBox.Items>
</ListBox>
<!-- ili -->
<ListBox>
    <ListBoxItem Content="Opcija 1"/>
    <ListBoxItem Content="Opcija 2"/>
</ListBox>
```

Pripojena svojstva (*attached property*)

- Pripojeno svojstvo je ovisno svojstvo kojemu se vrijednosti mogu dodjeljivati na klasama različitim od one gdje je definirano
- npr. želimo tip i veličinu fonta definirati na `StackPanel` klasi koja nema ta svojstva
  - Željena svojstva definirana su na `TextElement` klasi i mogu se dodijeliti putem pripojenih svojstava

```
<StackPanel
    TextElement.FontFamily="Arial"
    TextElement.FontSize="30">
    <Label Content="Pozdrav"/>
</StackPanel>

<Canvas>
    <Button Canvas.Top="20"
            Canvas.Left="20"
            Content="Klikni me"/>
</Canvas>
```