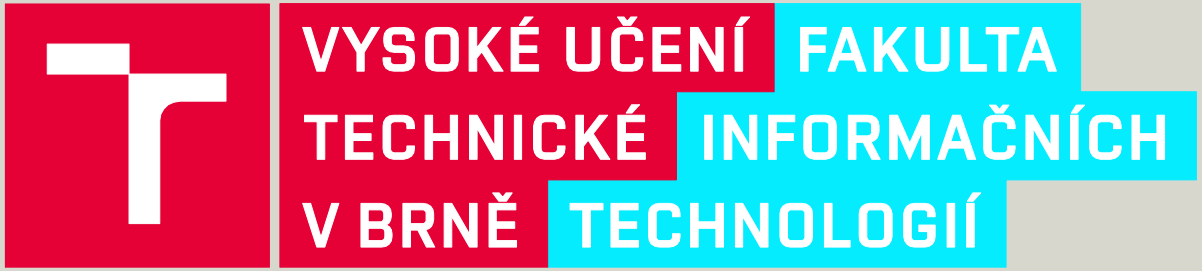
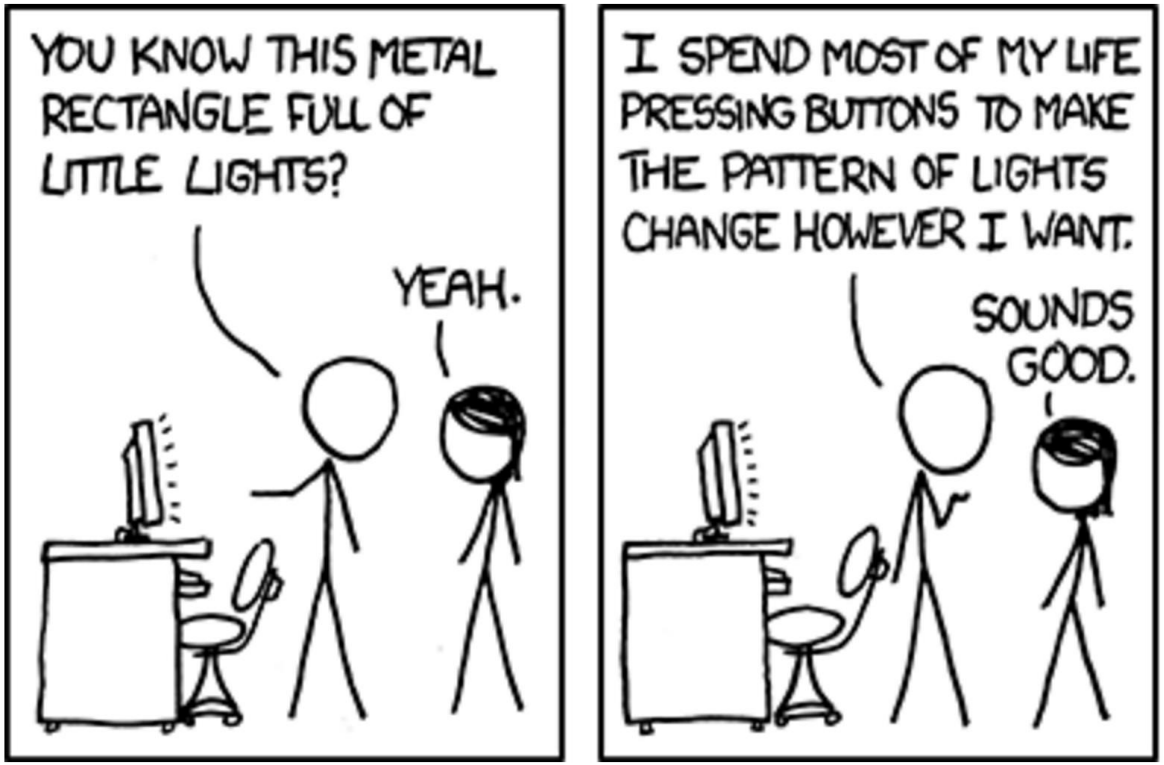


WPF – Desktopové aplikace

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Trocha teorie (snad) nikoho nezabije!



Co to je WPF?

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Windows Presentation Foundation (WPF)

- „Nový” grafický systém pro Windows
- Zašťiťuje tvorbu „rich-media” aplikací
- Přináší čistou separaci rolí mezi **UI** (XAML) a **business logic** (C#, VB.NET)
- Ovlivněn technologiemi jako je *HTML*, *CSS*, *Flash*
- Hardwarová akcelerace

WPF - Vektorová grafika

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

- Všechny aplikace využívající WPF jsou **Direct3D**
  - **Direct3D** (součást **DirectX**) se používá v grafických aplikacích cílících na výkon
  - Vykreslení akcelerováno grafickou kartou
  - *Výsledek*: vysoce kvalitní „rich-media UI”
- Vektorová grafika zajišťuje přiblížení, změnu rozměrů bez ztráty kvality
- WPF implementuje
  - float point system logických pixelů
  - 32-bit ARGB barevné spektrum

WPF - Textový model

- WPF umožňuje využít rozsáhlou podporu *typografických* a *text renderujících* funkcí
- Vytváří mezinárodní fonty z kompozitních fontů
- WPF renderovací engine využívá *ClearType* technologii
  - Použití před-renderovaných textů uložených ve *video paměti*
  - *Architektura WPF není závislá na rozlišení*

WPF - Animace

- WPF podporuje **časované animace**
  - Časovače jsou *inicializované* a *manažované* v řežii WPF
  - Změny na obrazovce jsou *koordinované* použitím **storyboard**
- Animace mohou bý iniciovány
  - Pomocí *externích událostí*
  - Včetně *vstupů od uživatele*
- Animace může být *definována na úrovni objektů* přímo v rámci XAML

WPF – Audio a Video

- WPF přináší podporu zakomponování audia a videa do UI
- Podpora **audia** je tvořena tenkou vrstvou nad existující *Win32* a *WMP* funkcionalitou
- Podpora **videa** nativně umožňuje použití pormátů *WMV*, *MPEG* a podmnožinu *AVI*
- Vztahy mezi videm a animacemi jsou podporovány
  - Využíváme oboje k vytvoření dynamičnosti obsahu
  - Animace mohou být *synchronizovány s medii*

WPF – Styly a Šablony 1/2

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

- Ve WPF je **styl** sadou vlastností aplikovaných na obsah za účelem *docílení změny v renderování*
  - Konceptuálně stejné jako CSS
  - Např. změna fontu u tlačítka – *Button control*
- Požití převážně pro *standardizaci vlastností* nastavených u jednotlivých prvků
- WPF styly obsahují *specifické vlastnosti* pro tvorbu apliakceí
  - Např. možnost *provedení různých vizuálních efektů* na základě akce provedené uživatelem

WPF – Styly a Šablony 2/2

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

- **Šablona** ve WPF umožňuje *kompletní změny v UI* jakýchkoliv WPF prvků
- Dostupné šablony
  - *ControlTemplate* – sdileni vzhledu napric UI Controls
  - *ItemsPanelTemplate* – vzhled panelu, napr. ListBox
  - *DataTemplate* – vhled objektů v panelech
  - *HierarchicalDataTemplate* - vzhled objektů v panelech s hierarchickou strukturou, napr. TreeView

WPF – Commandy 1/2

- **Command** je abstraktnější a volně spojovanou (*loosely-coupled*) verzí událostí (*events*)
  - Např. *Copy, cut, paste, save*, atd...
- WPF podpora commandů umožňuje šetření kódu, který je nutné napsat
- Přináší flexibilitu změnit UI bez rozbití „back-end“ logiky
- Commandy mají *action, source, target* a *binding*

WPF – Commandy 2/2

- *Proč používat commandy?*
  - WPF definuje řadu předpřipravených commandů
  - Commandy obsahují automatickou poporu pro vstupní akce
  - Některé WPF komponenty mají vestavěno chování spjaté s různými commandy
    - Např. Button
  - Efekt => čistý kód, nepoužívá se *Code-behind*
- Commandy jsou určeny pro:
  - Vykonaní akce
  - Ověření, zdali je akci možné vykonat

Deklarace UI pomocí XAML

- **XAML = eXtensible Application Markup Language**
- *Deklarativní jazyk* – říká **Co**, neříká **Jak**
- Popisuje primárně chování a integraci UI komponent
- Formát pro serializaci hierarchie objektů
- *.NET namespaces* reprezentuje pomocí XML namespaces
- Typicky propojen s *Code-behind* třídou

WPF – vykreslení objektů

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

- WPF pracuje na nejnižší úrovni s **tvary** (shapes) nikoliv s pixely
  - Tvary jsou vektorové vyjádřeny a mohou být lehce manipulovány
- Vývojář vytvoří tvar a nechá WPF, aby jej vykreslilo tou *nejoptimalizovanější* cestou
  - WPF přináší řadu předpřipravených tvarů jako *line, rectangle, ellipse, path* a jine
  - Objekty tvarů mohou být použity uvnitř panelů (*panels*) a dále uvnitř většiny WPF komponent

XAML - základy

- XAML je založený na XML
- Určený pro deklaraci a inicializaci .NET objektů
  - Použití XAML v WPF pro *snadnou ruční editaci člověkem*
  - Využit pro **separaci** *UI* od *C# kódu*
- XAML obsahuje hierarchii elementů reprezentující vizuální objekty
  - Tyto objekty nazýváme *user interface elemnts* neboli *UI elementes*

Elementy a Atributy - Vlastnosti objektů

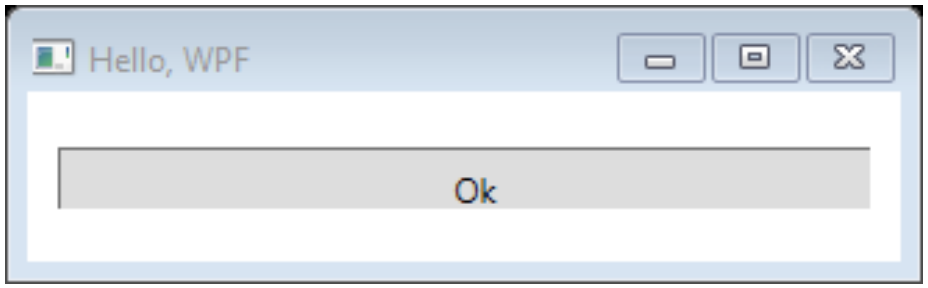
- UI elementy společnou podmnožinu vlastností a funkcí
  - Např. *Width, Height, Cursor* a *Tag* vlastnosti (*property*)
- Deklarování XML **elementu** v XAML
  - Ekvivalentní k instancializaci objektu přes bezparametrový konstruktor
- Nastavení **atributu** na objektu elementu
  - Ekvivalentní k přiřazení do vlastnosti daného jména

- **Atribut** – jednoduché propertie
- **Element** – UI Elementy, složitější propertie – instanciované třídy
- Výchozí vlastnost

XAML – základy - ukázka

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

```
<Window x:Class="HelloWorld.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:HelloWorld"
        mc:Ignorable="d"
        Title="Hello, WPF"  Height="100" Width="330">
    <Grid>
        <Button x:Name="OkButton" Content="Ok" Height="30" Width="300"
    </Grid>
</Window>
```



XAML – ukázka - vysvětlení

Window/UserControl/… - z čeho tato třída dědí
x:Class - třída s *Code-behindem*
xmlns:x - speciální namespace pro účely XAMLu (povinný)
xmlns:d –vlastnosti použité v designeru, ignorují se **(mc:Ignorable)**
xmlns - namespace s vestavěnými komponentami v WPF

- Kořenový element Window deklaruje *partial class*
- *Width, Height, Title* jsou XAML vlastnosti (*properties*)
- Element *Button* deklaruje prvek tlačítko

Propertie elementy

- Ne všechny **vlastnosti** (*propertie*) musí nabývat pouze hodnot typu string
  - Některé property mohou obsahovat instance objektů
- XAML obsahuje syntaxi pro nastavení hodnot *komplexních vlastností* nazvanou *propertie elements*
  - Ve formě elementu **TypeName.PropertyName** obsaženém uvnitř **TypeName** elementu

```
<Grid>
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="1*"/>
      <ColumnDefinition Width="2*"/>
   </Grid.ColumnDefinitions>
</Grid>
```

UI deklarativně vs imperativně

- V WPF může být každý element vytvořen buď deklaradivně nebo programově
  - Není *žádný rozdíl* ve vykonávání či rychlosti vykonávání
    - Instanciování elementu z *Code-behind* jde **proti** myšlence XAML
      - Stejný přístup jako u Windows Forms

MainWindows.xaml:

```
<Button Content="Click ME!" />
```

MainWindow.xaml.cs:

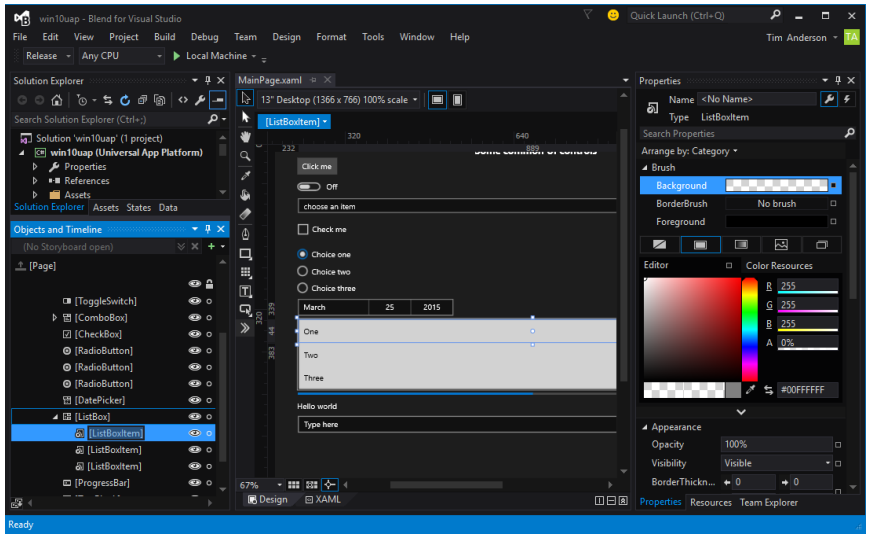
```
var button = new Button();
button.Content = "Click ME!";
```

Deklarativni UI – Idea 1/2

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

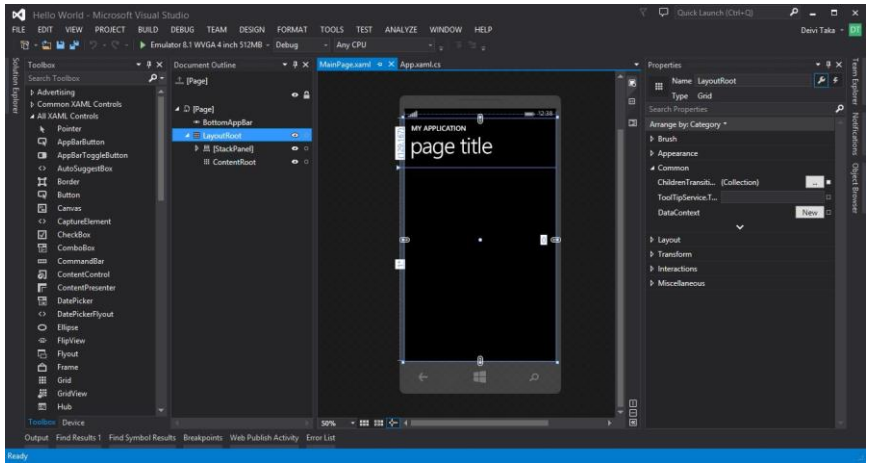
Designér, návrhář UI

- používá *Expression Blend*
- edituje XAML soubory



Vývojář

- používá *Visual Studio*
- pracuje s *code-behind*



V praxi se to často překrývá

Deklarativni UI – Idea 2/2

- *Co se stane když, se nepoužívá XAML v WPF?*
  - Ztrácí se myšlenka separace zájmů – *separation of concerns.*
  - *Designér* a *programátor* **nemohou** zaráz pracovat na stejném souboru.

- *Co se stane když, vytvoříte objekt deklarativně?*
  - Objekt se instanciuje voláním *bezparametrového konstruktoru*.
  - Všechna magie se odehraje v *InitializeComponent();*

Silverlight

- Silverlight je *cross-platform*, *cross-browser plugin*, který obsahuje technologii založenou na WPF
  - *Rich Internet Application* (RIA) platforma
  - Obsahuje XAML a **podmnozinu** WPF
  - Umoznruje „rich-media“ funkcionalitu jako je video, vektorová grafika a animace
- Silverlight a WPF sdílí stejnou XAML prezentační vrstvu
  - Obě technologie jsou si velmi podobné, ale *Silverlight* je **limintovaný** v mnoha aspektech.
- **Deprecated** - konec podpory Silverlight 5, Rijen 2021

WPF - prakticky

Hierarchie tříd

- *System.Object*
- *System.Windows.DependencyObject*
  - podpora dependency properties
- *System.Windows.UIElement*
  - metody pro vykreslování
- *System.Windows.FrameworkElement*
  - podpora pro databinding, styly atd.
- *System.Windows.Controls.Control*
  - Bázoví třída pro definici *UI elements*

Panely - layout

- jediné komponenty, které mohou mít víc potomků
- slouží k rozmístění prvků na ploše

Tendence ve WPF

- vektorová grafika
- přizpůsobení UI velikosti plochy
- "gumový layout"

System.Object
System.Windows.Threading.DispatcherObject
System.Windows.DependencyObject
System.Windows.Media.Visual
System.Windows.UIElement
System.Windows.FrameworkElement
System.Windows.Controls.Panel

Panely - layout

Canvas

- absolutní pozicování v pixelech
- vlastnosti *Canvas.Top, Canvas.Left*

Grid

- tabulkový layout
- vlastnosti *Grid.Row, Grid.Column, Grid.RowSpan, Grid.ColumnSpan*

StackPanel

- komponenty vedle sebe nebo pod sebou
- Vlastnost *StackPanel.Orientation*

WrapPanel

- komponenty vedle sebe a pak pod sebou, ci opacne
- Vlastnost *WrapPanel.Orientation*

Content Controls

- Jen jeden potomek
- Border
  - rámeček a pozadí okolo obsahu
- Button
- Label
- CheckBox, RadioButton
- ScrollViewer
  - pokud se obsah nevejde, objeví se posuvníky

System.Object
System.Windows.Threading.DispatcherObject
System.Windows.DependencyObject
System.Windows.Media.Visual
System.Windows.UIElement
System.Windows.FrameworkElement
System.Windows.Controls.Control
System.Windows.Controls.ContentControl

Vlastnosti pro pozicování

- *Width, MinWidth, MaxWidth*
- *Height, MinHeight, MaxHeight*
- *HorizontalAlignment, VerticalAlignment*
  - zarovnání v rámci rodičovského elementu
- *HorizontalContentAlignment, VerticalContentAlignment*
  - zarovnání vnitřního obsahu
- *Margin, Padding*
  - vnější a vnitřní okraj

System.Object
System.Windows.Threading.DispatcherObject
System.Windows.DependencyObject
System.Windows.Media.Visual
System.Windows.UIElement
System.Windows.FrameworkElement

Formátovaný text

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

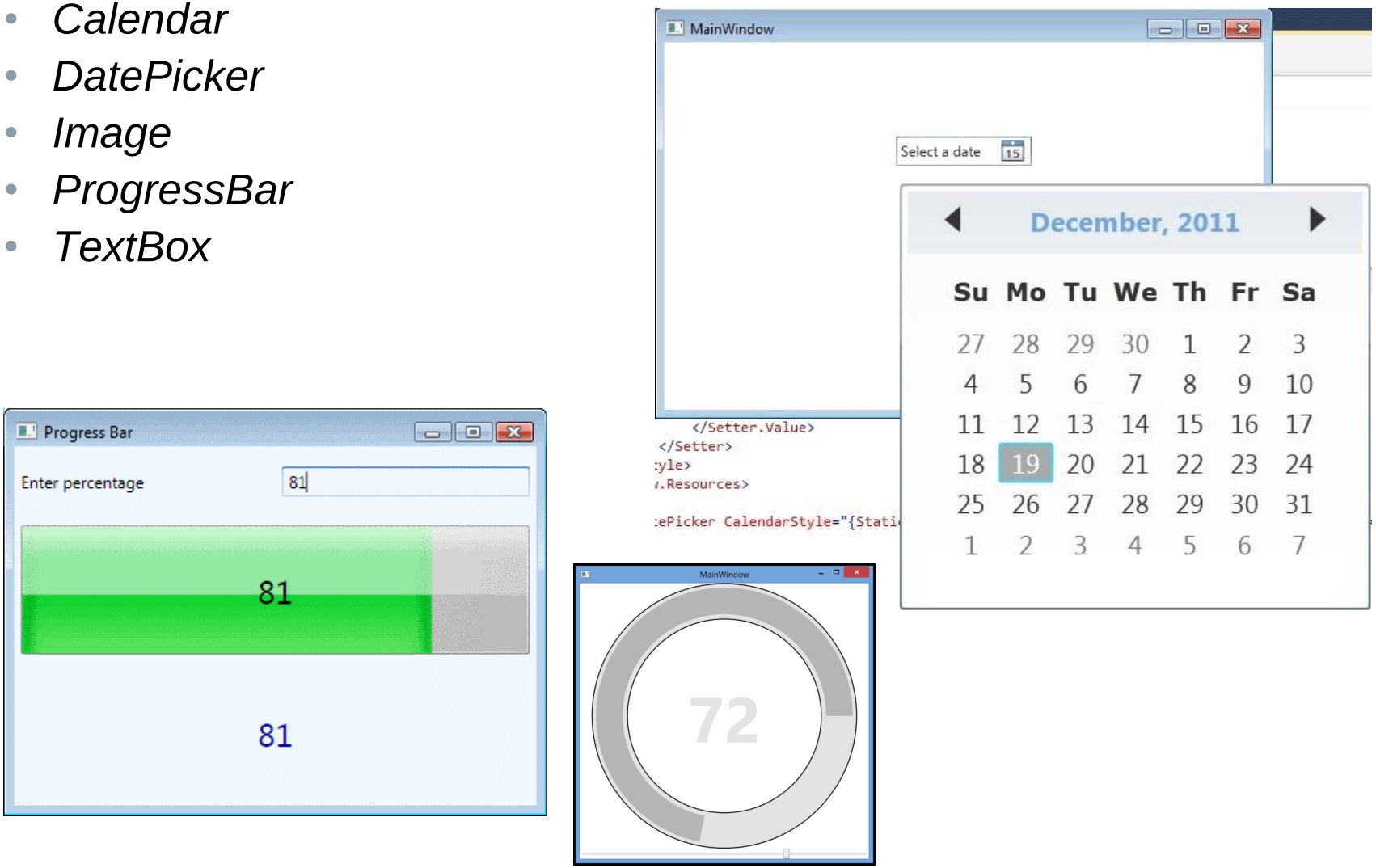
TextBlock

- vlastnost *TextWrapping*
- uvnitř
  - Run
    - *FontWeight, FontSize, Foreground*…
  - LineBreak, Span, Hyperlink, Bold, Italic, Underline

```
<TextBlock>
Sample text with <Bold>bold</Bold>, <Italic>italic</Italic>
and <Underline>underlined</Underline> words.
Username: <Run FontWeight="Bold" Text="{Binding UserName}"/>
</TextBlock>
```

Další komponenty

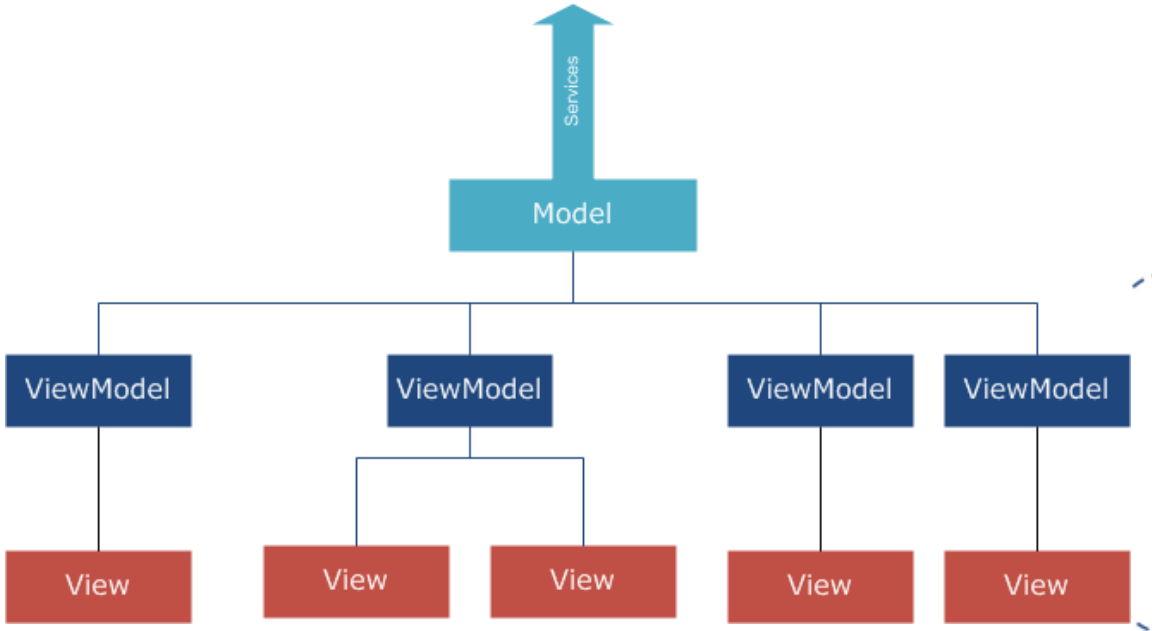
- *Calendar*
- *DatePicker*
- *Image*
- *ProgressBar*
- *TextBox*

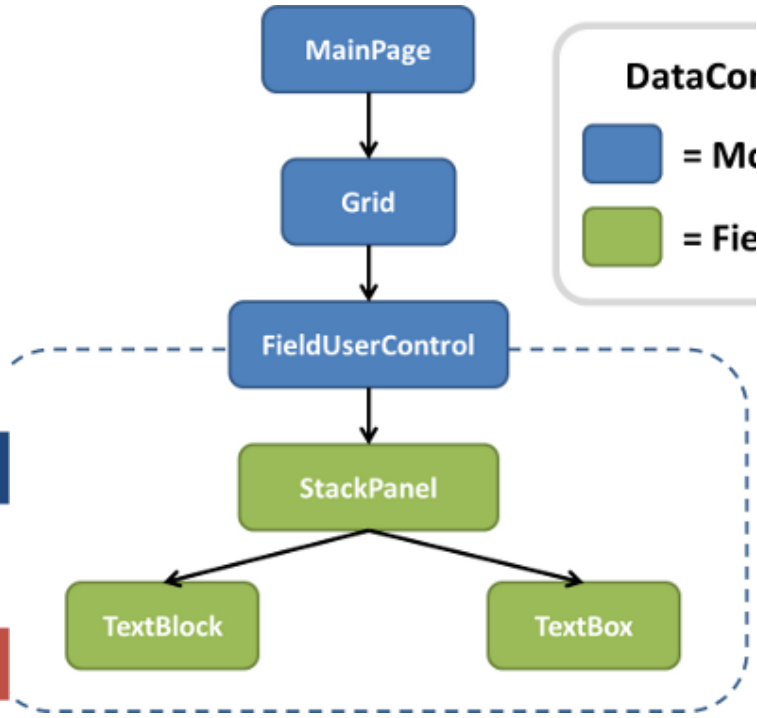


DataContext

- vlastnost třídy *FrameworkElement*
- pokud není nastaven, **přebírá se z rodiče** ve stromu komponent
- ideální prostředek pro *databinding*

- **typu object** – může obsahovat cokoliv





Typy bindingu

Proti ***aktuálnímu DataContextu***

- **{Binding}**
  - Aktuální DataContext

- **{Binding Name}**
  - Vlastnost (property) *Name* na aktuálním *DataContextu*

- **{Binding Name.Length}**
  - *DataContext.Name.Length*

Proti ***pojmenovanému elementu***

- vlastnost *x:Name*
- **{Binding Path=Text, ElementName=TextBox1}**
  - vlastnost Text objektu TextBox1

Směr bindingu

Vlastnost **Mode**

- **OneTime**
  - jen jednou na začátku
- **OneWay**
  - jedním směrem – ze zdroje do cíle
- **TwoWay**
  - obousměrně – změna v cíli změní i zdroj
- **OneWayToSource**
  - jedním směrem – od cíle ke zdroji
- **Default**
  - Uživatelsky editovatelné mají TwoWay, ostatní OneWay

Zdroj – vlastnost, na kterou bindujeme

Cíl – komponenta, která má *{Binding}*

Směr bindingu

OneWay a **TwoWay**

- reaguje na změny zdroje
- zdroj o nich musí dát nějak vědět
- objekt implementuje **INotifyPropertyChanged**
  - při změně vlastnosti vyvolá událost **PropertyChanged**

```
public class MainViewModel : INotifyPropertyChanged {
        private string _name;
        public event PropertyChangedEventHandler PropertyChanged;

        public string Name {
            get { return _name; }
            set {
                _name = value;
                this.OnPropertyChanged();
                this.OnPropertyChanged(nameof(FancyName));
            }
        }

        public string FancyName => $"***this.Name***";

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

Kolekce

Vlastnost objektu je kolekce

- položky jsou reprezentovány kolekcí vnitřních elementů
- System.Object
  System.Collections.*
- Implementují rozhraní IEnumerable
- Pro zobrazení změn musí kolekce implementovat rozhraní INotifyCollectionChanged

```
public class MainViewModel {
   public ObservableCollection<MenuItem> MenuItems { get; }
          = new ObservableCollection<MenuItem>();
}
```

ItemsControl

ComboBox
ListBox
TabControl
TreeView

…
System.Windows.Controls.Control
System.Windows.Controls.ItemsControl

ItemsControl

- Kolekce **Items**
  - obecné objekty, umístí se dovnitř
- Vlastnost **ItemsSource**
  - chce IEnumerable, pomocí něj vygeneruje položky
- Šablona **ItemTemplate**
  - definuje vzhled a obsah položky
    - DataContext je aktuální prvek kolekce

```
<ListBox ItemsSource="{Binding MenuItems}">
   <ListBox.ItemTemplate>  <DataTemplate>
         <StackPanel>
            <TextBlock Text="{Binding Title}" />
            <TextBlock Text="{Binding SubTitle}" />
         </StackPanel>
      </DataTemplate> </ListBox.ItemTemplate>
</ListBox>
```

ItemsControl – změna kolekce

Změna **ItemsSource**

- přiřadíme jinou kolekci
  - obsah se smaže a vygeneruje pro nová data
- změníme vlastnost prvku v kolekci
  - jen u dependency property nebo **INotifyPropertyChanged**
- přidáme nebo odebereme prvek v kolekci
  - kolekce musí implementovat **INotifyCollectionChanged**

ItemsControl - ListBox

- vlastnost **SelectedItem**
  - přímo objekt, který je bindovaný
- vlastnost **SelectedValuePath**
  - co se z objektu pouzije jako hodnota
    - *objekt.Vlastnost1.Vlastnost2* atd.
- vlastnost **SelectedValue**
  - hodnota dané položky

```
<ListBox
         ItemsSource="{Binding MenuItems}"
         SelectedItem="{Binding SelectedItem}"
         SelectedValue="{Binding SelectedTitle}"
         SelectedValuePath SelectedValuePath="@Title"
/>
```

INotifyCollectionChanged

ObservableCollection<T>

- implementuje toto rozhraní

Vlastní kolekce

- podporu přidáme

Existující kolekce

- např. napsat kolem ní "wrapper"

Commands

- Imeplementuje rozhrani ICommand
  - public interface ICommand
- Metody
  - Execute(Object) – definuje metodu, která se bude volat, když je command spuštěn
  - CanExecute(Object) – definuje metodu, která ověří, zdali je command může být spuštěn v aktuálním stavu
- Event
  - CanExecuteChanged – Událost je zavolána v případně, že se změní stav a command je možné vykonat

Commands - RelayCommand

RelayCommand – MVVMLight, Telerik

MyViewModel.cs:

```
private RelayCommand _myCommand;
public RelayCommand MyCommand => _myCommand ??
    (_myCommand = new RelayCommand(Execute,CanExecute);
private void Execute() {
        //...
}
private bool CanExecute() {
    return 1 == 1;
}
```

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ABCDEFGH

AAAA BBBB CCCC DDDD

abcd

Lorem ipsum dolores sit amet

abcd

Lorem ipsum dolores sit amet

abcd

Lorem ipsum dolores sit amet

abcd

Lorem ipsum dolores sit amet

abcd

Lore ipsum dolores

ABCD

Lorem ipsum dolores sit amet

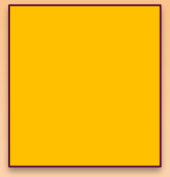
AAAA

BBBB

CCCC

DDDD

EEEE



ABCD

Lorem ipsum dolores sit amet Lorem ipsum dolores sit amet Lorem ipsum dolores sit amet.