Platformy Technologiczne

Jarosław Kuchta

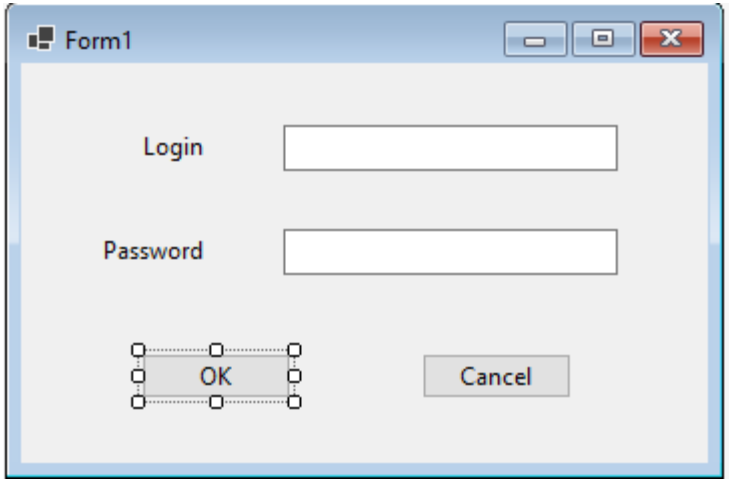
Windows Presentation Foundation

WF – WPF – UWP – XAMARIN

- Windows Forms
  - .NET Framework 2.0
  - proste komponenty GUI
  - korzysta z natywnych wywołań Win32
  - obecnie w trybie utrzymywania
- Windows Presentation Foundation (WPF)
  - .NET Framework 3.0
  - deklaratywny język XAML+C#
  - dwukierunkowa integracja z Windows Forms (ElementHost, WindowsFormsHost)
- Universal Windows Platform
  - Windows 10
  - uniwersalna platforma wykonania dla komputerów, konsoli Xbox, urządzeń mobilnych i aplikacji Internet of Things (np. HoloLens)
  - interfejs dotykowy
  - czy następca WPF?
- Xamarin
  - oparty o Mono (a nie .NET Framework)
  - natywna platforma GUI dla aplikacji Windows, Android, iOS

Windows Forms Design

Projektowanie wizualne



Code-behind

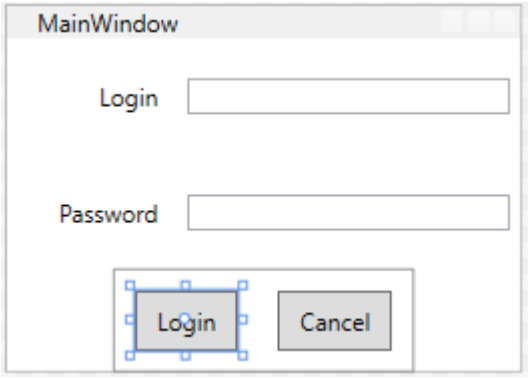
```
public partial class Form1 : Form
 {

   private void OkButton_Click
         (object sender, EventArgs e)
   {

   }
 }
```

Kod generowany (layout)

```
partial class Form1
 {
   #region Windows Form Designer generated code

   private void InitializeComponent()
   {
     this.OkButton = new System.Windows.Forms.Button();
     this.CancelButton = new System.Windows.Forms.Button();
     ...
     //
     // OKbutton
     //
     this.OkButton.Location = new System.Drawing.Point(60, 145);
     this.OkButton.Name = "button1";
     this.OkButton.Size = new System.Drawing.Size(75, 23);
     this.OkButton.TabIndex = 0;
     this.OkButton.Text = "OK";
     this.OkButton.UseVisualStyleBackColor = true;
     this.OkButton.Click += new System.EventHandler(this.OkButton_Click);
     ...
   }

   #endregion

   private System.Windows.Forms.Button OKButton;
   private System.Windows.Forms.Button CancelButton;
   private System.Windows.Forms.Label label1;
   private System.Windows.Forms.Label label2;
   private System.Windows.Forms.TextBox textBox1;
   private System.Windows.Forms.TextBox textBox2;
 }
}
```

WPF Design

Projektowanie wizualne



XAML

```
<Window x:Class="WpfApp1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="181.75" Width="256.125">
  <Grid>
    <Grid.RowDefinitions>…</Grid.RowDefinitions>
    <Grid Grid.Row="0">
      <Grid.ColumnDefinitions>…</Grid.ColumnDefinitions>
      <Grid.RowDefinitions>…</Grid.RowDefinitions>
      <Label VerticalAlignment="Center"
             Margin="5" …>Login</Label>
      <Label VerticalAlignment="Center"
             Margin="5" …>Password</Label>
      <TextBox VerticalAlignment="Center"
               Margin="5" …></TextBox>
      <TextBox VerticalAlignment="Center"
               Margin="5" …></TextBox>
    </Grid>
    <StackPanel Grid.Row="1"
                Orientation="Horizontal" …>
      <Button Margin="10"
              Padding="10,5">Login</Button>
      <Button Margin="10"
              Padding="10,5">Cancel</Button>
    </StackPanel>
  </Grid>
</Window>
```

Code-behind

```
public partial class MainWindow : Window
 {

   private void LoginButton_Click
        (object sender, RoutedEventArgs e)
   {

   }
 }
```

Windows Forms – WPF porównanie kontrolek

Kontrolki kompatybilne

- Label
- TextBox, RichTextBox
- Button
- CheckBox
- ComboBox
- GroupBox
- RadioButton
- ListBox
- ListView
- TreeView
- …

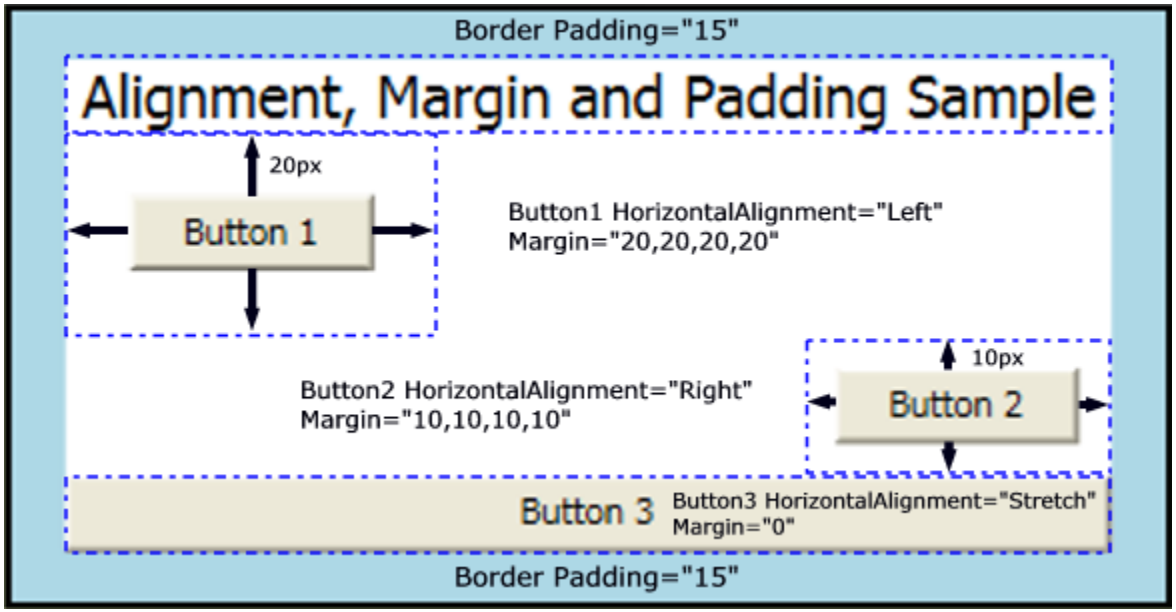
Tylko WPF

- Grid
- DockPanel
- Canvas
- Rectangle, Ellipse
- …

Tylko Forms

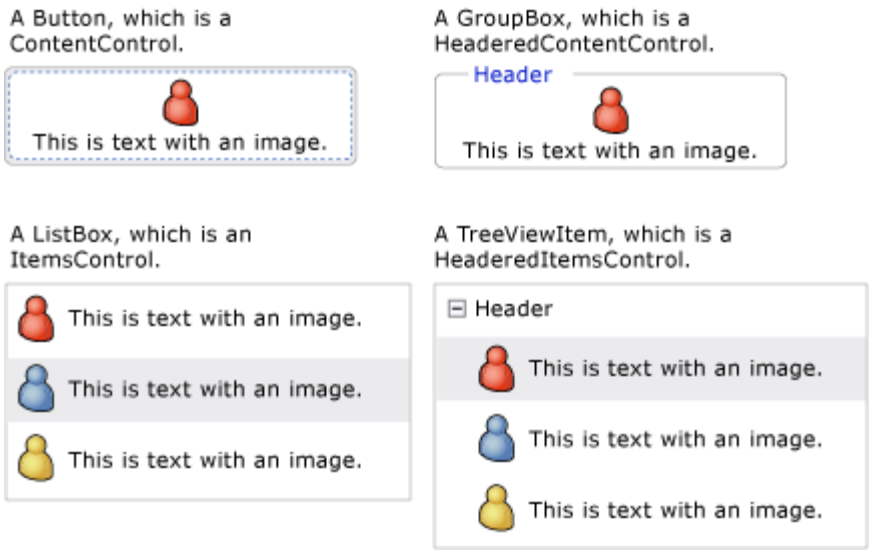
- PropertyGrid
- AxHost
- …

Model wizualny kontrolki



Model zawartości WPF

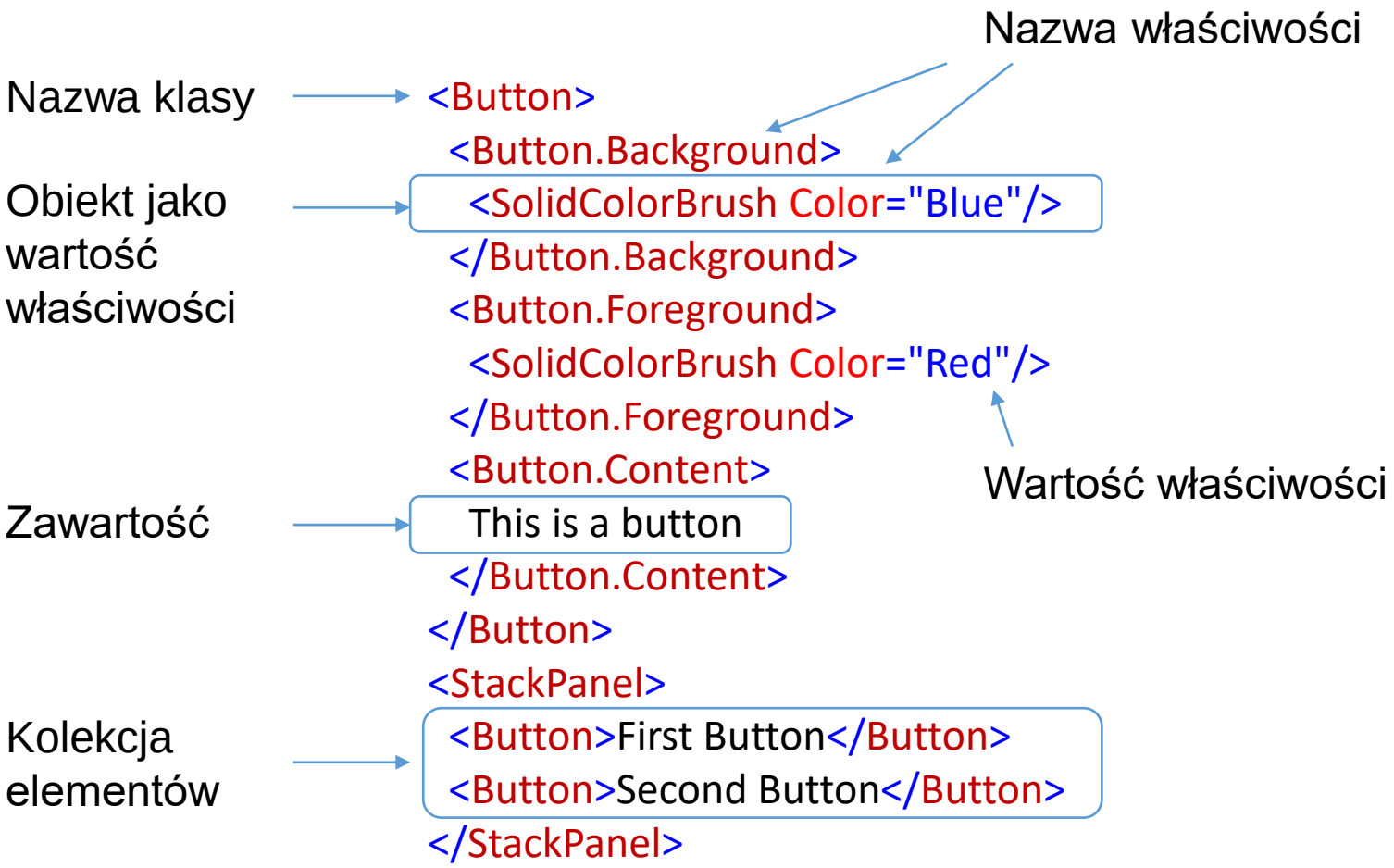
- ContentControl
- HeaderedContentControl
- ItemsControl
- HeaderedItemsControl



Przegląd XAML

- eXtensible Application Markup Language
- oparty o XML
- język deklaratywny
- do deklaracji kontrolek i zasobów
- zorientowany obiektowo
- zintegrowany z kodem .NET Framework
- zintegrowany z kodem aplikacji
- zapewnia powiązania danych

Składnia obiektowa XAML



Przestrzenie nazw XAML

```
<UserControl x:Class="PritiGraph.Views.TokenView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:views="clr-namespace:PritiGraph.Views"
    xmlns:local="clr-namespace:PritiGraph"
    mc:Ignorable="d"
    d:DesignHeight="30" d:DesignWidth="50">
```

Standardowe

z własnego kodu (code-behind)

Deklarowanie zasobów w XAML

Zasoby lokalne

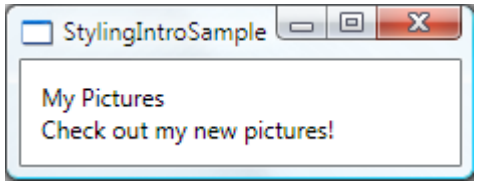
```
<UserControl.Resources>
   <local:ColorBrushConverter x:Key="RedBlackColors"/>
</UserControl.Resources>
```

Słownik zasobów

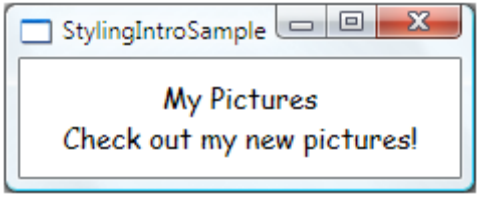
```
<ResourceDictionary
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:views="clr-namespace:PritiGraph.Views"
        xmlns:local="clr-namespace:PritiGraph"
        xmlns:vm="clr-namespace:PritiGraph"
        >
  <SolidColorBrush x:Key="TagFill" Color="Silver"/>
  <SolidColorBrush x:Key="ShapeFill" Color="WhiteSmoke"/>
  <views:ColorDictionary x:Key="RedBlackColors">
    <Color x:Key="False">Red</Color>
    <Color x:Key="True">Black</Color>
  </views:ColorDictionary>
  <local:ValidityBrushConverter x:Key="ValidityBrushConverter"
                ColorDictionary="{StaticResource RedBlackColors}"/>
```

Użycie zasobów lokalnych

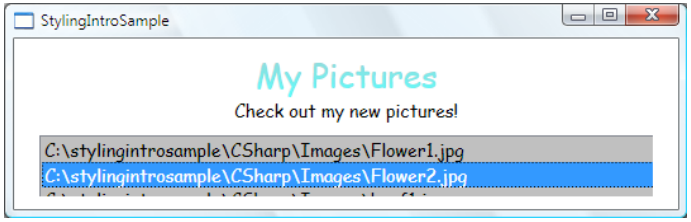
Stylizacja




```
<Window.Resources>
  <!--A Style that affects all TextBlocks-->
  <Style TargetType="TextBlock">
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="FontFamily" Value="Comic Sans MS"/>
    <Setter Property="FontSize" Value="14"/>
  </Style>
</Window.Resources>
```



Szablony prezentacji danych




```
<ListView.ItemsPanel>
  <StackPanel Orientation="Horizontal"/>
</ListView.ItemsPanel>
<ListView.ItemTemplate>
  <DataTemplate DataType="{x:Type local:Photo}">
  <Border Margin="3">
    <Image Source="{Binding Source}" MaxHeight="75"/>
  </Border>
 </DataTemplate>
</ListView.ItemTemplate>
```

Wyzwalacze (Triggers)


```
<Style TargetType="ListBoxItem">
  <Setter Property="Opacity" Value="0.5" />
  <Setter Property="MaxHeight" Value="75" />
  <Style.Triggers>
    <Trigger Property="IsSelected" Value="True">
      <Setter Property="Opacity" Value="1.0" />
    </Trigger>
```

Obsługa zdarzeń

Deklaracja w XAML

```
<Button Content="A Button"
        Click="Button_Clicked" />
```

Przypisanie w code-behind

```
button.Click += Button_Clicked;
```

Metoda obsługi

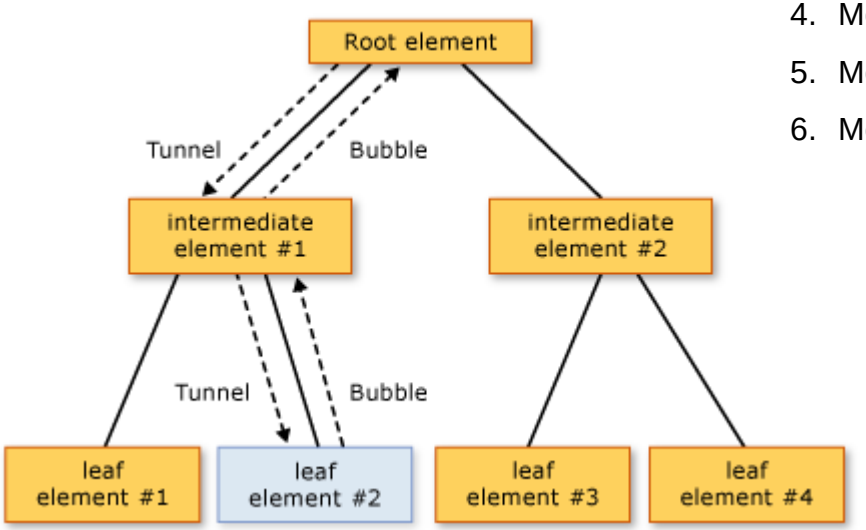
```
private void Button_Clicked(object sender, RoutedEventArgs e)
{
  Debug.WriteLine("Button clicked!");
}
```

Wyzwalacze zdarzeń i animacje

```
<EventTrigger RoutedEvent="Mouse.MouseEnter">
  <EventTrigger.Actions>
    <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
            Duration="0:0:0.2"
            Storyboard.TargetProperty="MaxHeight"
            To="90"  />
        </Storyboard>
    </BeginStoryboard>
  </EventTrigger.Actions>
</EventTrigger>
<EventTrigger RoutedEvent="Mouse.MouseLeave">
  <EventTrigger.Actions>
    <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
            Duration="0:0:1"
            Storyboard.TargetProperty="MaxHeight"  />
        </Storyboard>
    </BeginStoryboard>
  </EventTrigger.Actions>
</EventTrigger>
```

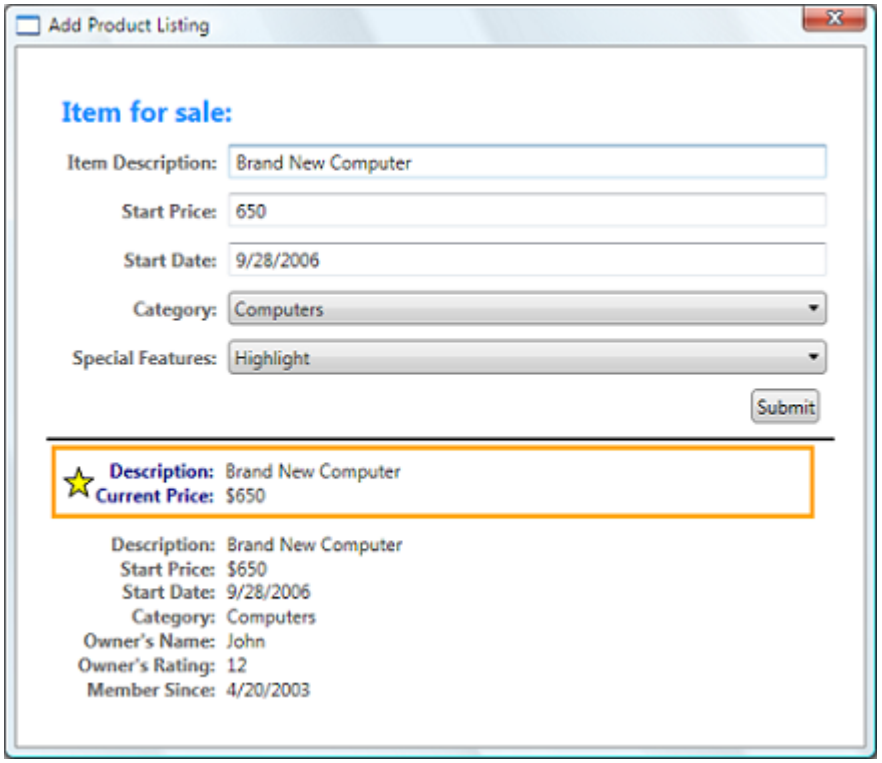
Routed Events

- Bubbling
- Direct
- Tunneling



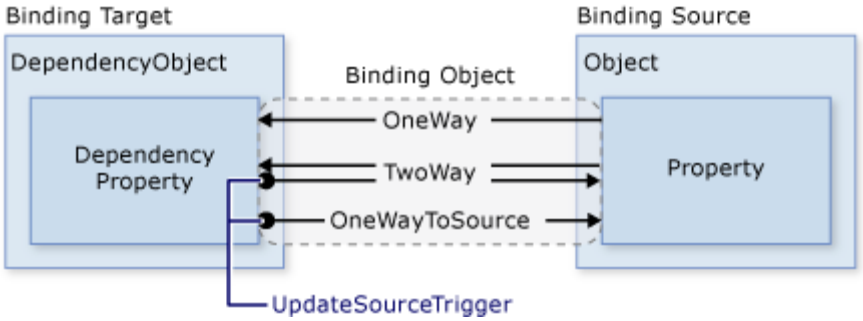
1. PreviewMouseDown (tunnel) na Root element.
2. PreviewMouseDown (tunnel) na intermediate element #1.
3. PreviewMouseDown (tunnel) na leaf element #2.
4. MouseDown (bubble) na leaf element #2.
5. MouseDown (bubble) na intermediate element #1.
6. MouseDown (bubble) na Root element.

Data Binding




```
<ListBox Name="Master" Grid.Row="2" Grid.ColumnSpan="3" Margin="8"
    ItemsSource="{Binding Source={StaticResource listingDataView}}">


<ContentControl Name="Detail" Grid.Row="3" Grid.ColumnSpan="3"
    Content="{Binding Source={StaticResource listingDataView}}"
    ContentTemplate="{StaticResource detailsProductListingTemplate}"
    Margin="9,0,0,0"/>
```

Kierunki wiązania



Obiekty zależne (Dependency Objects)

UserControl jest pochodna od DependencyObject

```
public partial class UserControl1 : UserControl {

   public UserControl1() {
      InitializeComponent();
   }
```

Deklaracja właściwości klasy

```
   public static readonly DependencyProperty SetTextProperty =
      DependencyProperty.Register("SetText", typeof(string),
        typeof(UserControl1), new PropertyMetadata
                  ("", new PropertyChangedCallback(OnSetTextChanged)));
```

Deklaracja właściwości instancji

```
   public string SetText {
      get { return (string)GetValue(SetTextProperty); }
      set { SetValue(SetTextProperty, value); }
   }
```

Obsługa zmiany właściwości klasy

```
   private static void OnSetTextChanged(DependencyObject d,
      DependencyPropertyChangedEventArgs e) {
      UserControl1 UserControl1Control = d as UserControl1;
      UserControl1Control.OnSetTextChanged(e);
   }
```

Obsługa zmiany właściwości instancji

```
   private void OnSetTextChanged(DependencyPropertyChangedEventArgs e) {
      tbTest.Text = e.NewValue.ToString();
   }
}
```

Wiązania do klas POCO (Plain Old CLR Objects)

Klasa musi implementować interfejs INotifyPropertyChanged

Interfejs INotifyPropertyChanged wymaga implementacji zdarzenia PropertyChanged

Metoda RaisePropertyChanged wywołuje PropertyChanged

Pole prywatne przechowuje wartość

Getter po prostu zwraca pole

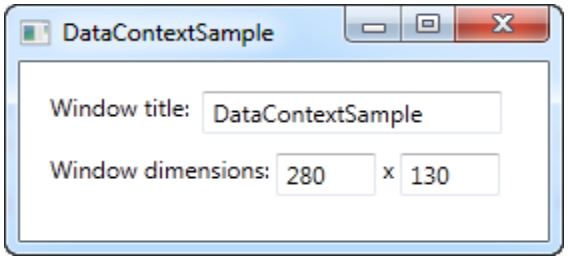
Setter sprawdza czy wartość jest różna od pola i tylko wówczas zmienia pole oraz wywołuje PropertyChanged przez metodę RaisePropertyChanged

```
public class Category : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;

  void RaisePropertyChanged(string prop)
  {
    if (PropertyChanged != null)
     PropertyChanged(this, new PropertyChangedEventArgs(prop));
  }

  string _CategoryId;
  public string CategoryId
  {
    get { return _CategoryId; }
    set
    {
      if (_CategoryId != value)
      {
        _CategoryId = value;
        RaisePropertyChanged("CategoryId");
      }
    }
  }
}
```

Kontrolka jako Data Context

```
<StackPanel Margin="15">
     <WrapPanel>
       <TextBlock Text="Window title:  " />
       <TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}" Width="150" />
     </WrapPanel>
     <WrapPanel Margin="0,10,0,0">
     <TextBlock Text="Window dimensions: " />
     <TextBox Text="{Binding Width}" Width="50" />
     <TextBlock Text=" x " />
      <TextBox Text="{Binding Height}" Width="50" />
  </WrapPanel>
</StackPanel>
```

```
public partial class DataContextSample : Window
{
  public DataContextSample()
  {
    InitializeComponent();
    this.DataContext = this;
  }
}
```



Items Source (1/3)

Deklaracja klasy danych →

```
public class User
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Mail { get; set; }
 }
```

Inicjacja listy danych →

ustawienie ItemsSource dla ListView →

```
public ListViewItemTemplateSample()
{
    InitializeComponent();
    List<User> items = new List<User>();
    items.Add(new User() { Name = "John Doe", Age = 42, Mail = "john@doe-family.com" });
    items.Add(new User() { Name = "Jane Doe", Age = 39, Mail = "jane@doe-family.com" });
    items.Add(new User() { Name = "Sammy Doe", Age = 13, Mail = "sammy.doe@gmail.com" });
    lvDataBinding.ItemsSource = items;
}
```

Uwaga: Ponieważ klasa User nie implementuje interfejsu INotifyPropertyChanged, a metoda ListViewItemTemplateSample używa "zwykłej" listy, a nie ObservableCollection, więc żadne zmiany danych nie będą pokazywane w widoku (patrz dalej przykład MVVM).

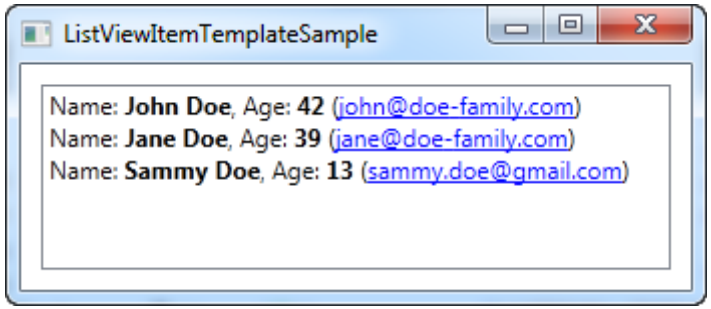
Items Source (2/3)

ListView z deklaracją DataTemplate :

```
<ListView Margin="10" Name="lvDataBinding">
 <ListView.ItemTemplate>
  <DataTemplate>
   <WrapPanel>
    <TextBlock Text="Name: " />
    <TextBlock Text="{Binding Name}" FontWeight="Bold" />
    <TextBlock Text=", " />
    <TextBlock Text="Age: " />
    <TextBlock Text="{Binding Age}" FontWeight="Bold" />
    <TextBlock Text=" (" />
    <TextBlock Text="{Binding Mail}" TextDecorations="Underline" Foreground="Blue" Cursor="Hand" />
    <TextBlock Text=")" />
   </WrapPanel>
  </DataTemplate>
 </ListView.ItemTemplate>
</ListView>
```

Items Source (3/3)

Efekt:

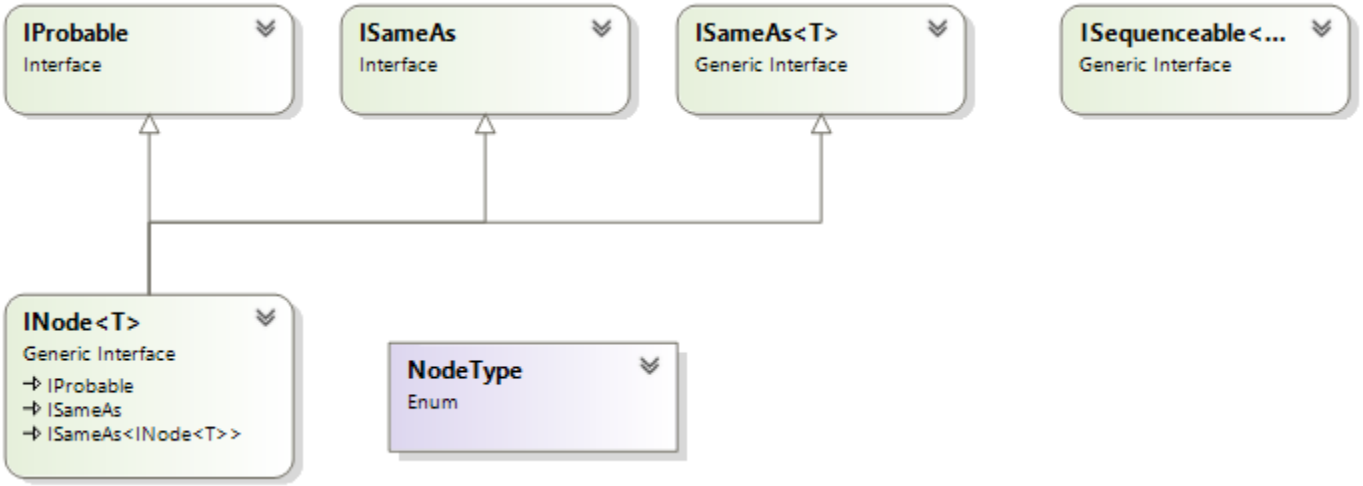


Model – View – ViewModel (MVVM)

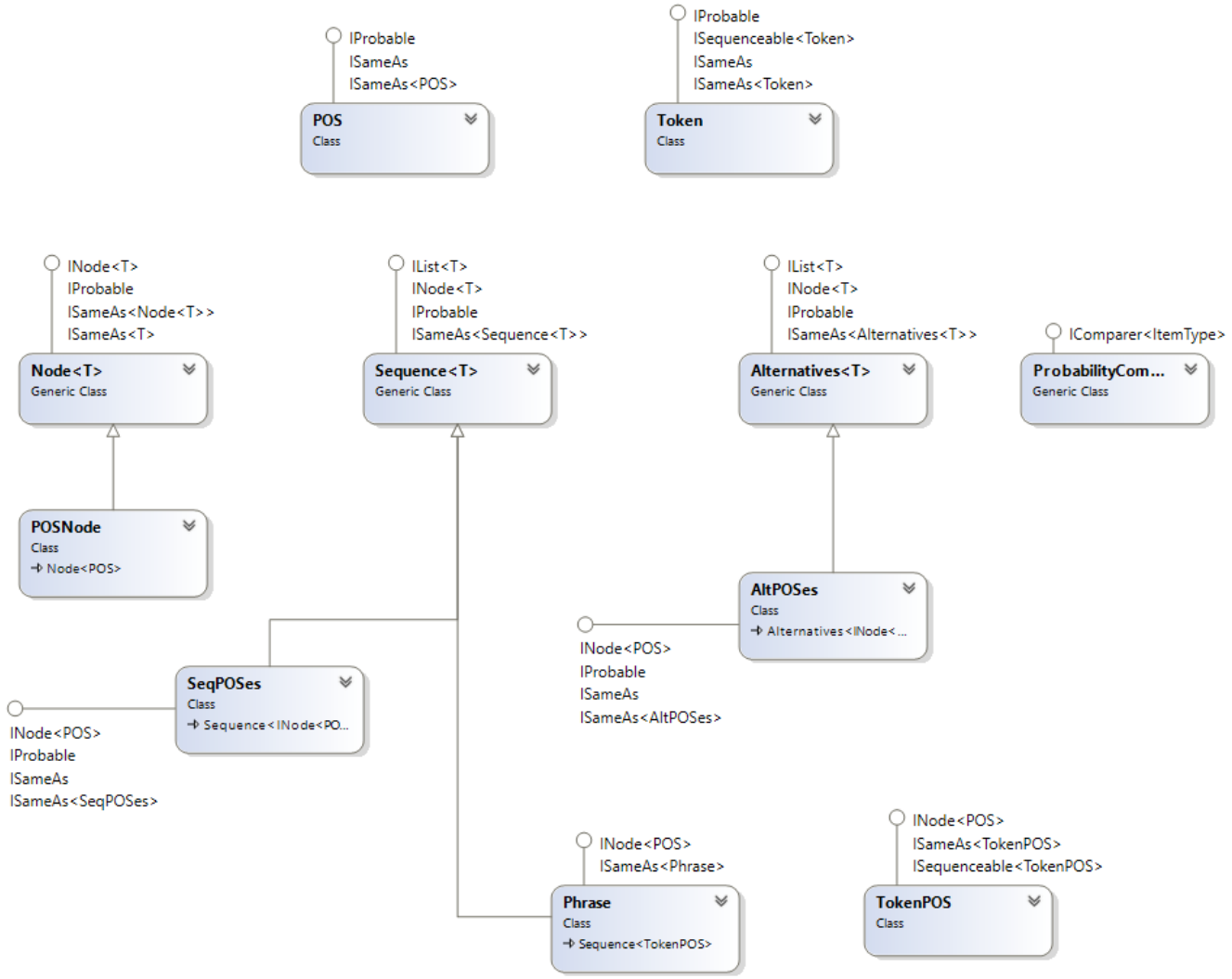
- Powiązanie modeli danych z widokiem
- Odświeża widok po zmianie danych
- Korzysta z interfejsów
- Różne klasy danych
- Abstrakcyjne i konkretne modele widoku
- Różne szablony danych

Jak powiązać różne modele danych

Użycie interfejsów



Klasy danych implementują interfejsy



Abstrakcyjny model widoku

```
public abstract class ViewModel : IViewModel
{
  public abstract bool IsValid { get; }

  public event PropertyChangedEventHandler PropertyChanged;

  public void NotifyPropertyChanged(string propertyName)
  {
    if (PropertyChanged != null)
      PropertyChanged.Invoke(this, new PropertyChangedEventArgs(propertyName));
  }
}
```

Konkretny model widoku (1/2)

```
public class POSesViewModel : ViewModel
{
  public POSesViewModel(INode<POS> poses)
  {
    Items = new ObservableCollection<IViewModel>();
    if (poses.NodeType==NLP.NodeType.Alternative)
      Orientation=Orientation.Vertical;
    else
      Orientation=Orientation.Horizontal;
    if (poses is Phrase seqPOSes)
      foreach (var item in seqPOSes)
        Items.Add(ViewModelFactory.CreateViewModel(item));
    else
      Items.Add(ViewModelFactory.CreateViewModel(poses));
  }
```

Konkretny model widoku (2/2) - Observable Collection

```
public ObservableCollection<IViewModel> Items { get; set; }

 public override bool IsValid => Items.Where(item => !item.IsValid).FirstOrDefault()==null;

 public Orientation Orientation
 {
  get { return _Orientation; }
  set
  {
   if (_Orientation!=value)
   {
    _Orientation=value;
    NotifyPropertyChanged("Orientation");
   }
  }
 }
 private Orientation _Orientation;
```

Value Converter

```
public class ValidityBrushConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
  {
    Color color = ColorDictionary[value.ToString()];
    return new SolidColorBrush { Color = color };
  }

  public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
  {
    throw new NotImplementedException();
  }

  public ColorDictionary ColorDictionary { get; set; }
}
```

ResourceDictionary

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:views="clr-namespace:PritiGraph.Views"
                    xmlns:local="clr-namespace:PritiGraph"
                    xmlns:vm="clr-namespace:PritiGraph"
                    >
  <SolidColorBrush x:Key="TagFill" Color="Silver"/>
  <SolidColorBrush x:Key="ShapeFill" Color="WhiteSmoke"/>
  <views:ColorDictionary x:Key="RedBlackColors">
    <Color x:Key="False">Red</Color>
    <Color x:Key="True">Black</Color>
  </views:ColorDictionary>
  <local:ValidityBrushConverter x:Key="ValidityBrushConverter"
                   ColorDictionary="{StaticResource RedBlackColors}"/>
  <views:ColorDictionary x:Key="RedGrayColors">
    <Color x:Key="False">Red</Color>
    <Color x:Key="True">LightGray</Color>
  </views:ColorDictionary>
  <local:ValidityBrushConverter x:Key="ValidityBorderConverter"
                   ColorDictionary="{StaticResource RedGrayColors}"/>
```

Szablony danych powiązane z typami danych

```
<DataTemplate
   x:Key="POSViewModelTemplate"
   DataType="{x:Type vm:POSViewModel}">
   <Grid MinHeight="10" MinWidth="10" Margin="{StaticResource ListViewItemMargin}">
     <views:POSView
               LeftFill="{StaticResource TagFill}"
               RightFill="{StaticResource ShapeFill}"
               LeftFillDictionary="{StaticResource POSTagColors}"/>
   </Grid>
 </DataTemplate>

 <DataTemplate
   x:Key="AltPOSesViewModelTemplate"
   DataType="{x:Type vm:AltPOSesViewModel}">
   <Grid MinHeight="10" MinWidth="10" Margin="{StaticResource ListViewItemMargin}">
     <views:AltPOSView/>
   </Grid>
 </DataTemplate>
...
```

Dynamiczny wybór szablonu

```
public class DynamicTemplateSelector : DataTemplateSelector
{
  public static readonly DependencyProperty TemplatesProperty =
  DependencyProperty.RegisterAttached("Templates", typeof(TemplateCollection), typeof(DynamicTemplateSelector),
  new FrameworkPropertyMetadata(new TemplateCollection(), FrameworkPropertyMetadataOptions.Inherits));

  public static TemplateCollection GetTemplates(UIElement element)
  {
    return (TemplateCollection)element.GetValue(TemplatesProperty);
  }

   public static void SetTemplates(UIElement element, TemplateCollection collection)
  {
    element.SetValue(TemplatesProperty, collection);
  }

   public override System.Windows.DataTemplate SelectTemplate(object item, System.Windows.DependencyObject container)
  {
     …
  }
 }
```

Selektor szablonu elementu

```
<local:DynamicTemplateSelector x:Key="MyTemplateSelector"/>

 <Style x:Key="MyListStyle" TargetType="ListView">
  <Setter Property="ItemTemplateSelector" Value="{StaticResource MyTemplateSelector}"/>
  <Setter Property="local:DynamicTemplateSelector.Templates">
   <Setter.Value>
    <local:TemplateCollection>
     <local:Template Value="{x:Type vm:POSViewModel}"
                     DataTemplate="{StaticResource POSViewModelTemplate}"/>
     <local:Template Value="{x:Type vm:AltPOSesViewModel}"
                     DataTemplate="{StaticResource AltPOSesViewModelTemplate}"/>
...
    </local:TemplateCollection>
   </Setter.Value>
  </Setter>
```

Przypisanie szablonu

Inne settery właściwości

```
<Setter Property="ItemsPanel">
    <Setter.Value>
      <ItemsPanelTemplate>
        <StackPanel Orientation="{Binding Orientation}"></StackPanel>
      </ItemsPanelTemplate>
    </Setter.Value>
  </Setter>
  <Setter Property="VerticalContentAlignment" Value="Top"/>
  <Setter Property="HorizontalContentAlignment" Value="Center"/>
  <Setter Property="BorderBrush" Value="{Binding IsValid, Converter={StaticResource ValidityBorderConverter}}"/>
  <Setter Property="Margin" Value="0,5,0,0" />
</Style>
```

Użycie konwertera wartości

Deklaracja widoku

```
<UserControl x:Class="PritiGraph.Views.PhrasesView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:views="clr-namespace:PritiGraph.Views"
             xmlns:vm="clr-namespace:PritiGraph"
             mc:Ignorable="d"
             d:DesignHeight="100" d:DesignWidth="100">

  <ListView x:Name="PhrasesListView"
```

Użycie stylu → `Style="{StaticResource MyListStyle}"`

Powiązanie ze źródłem → `ItemsSource="{Binding Items}">`

```
  </ListView>
</UserControl>
```

Efekt końcowy