.NET Multi-platform App UI (.NET MAUI) (2)

Programowanie Wizualne

Paweł Wojciechowski

Instytut Informatyki, Politechniki Poznańskiej

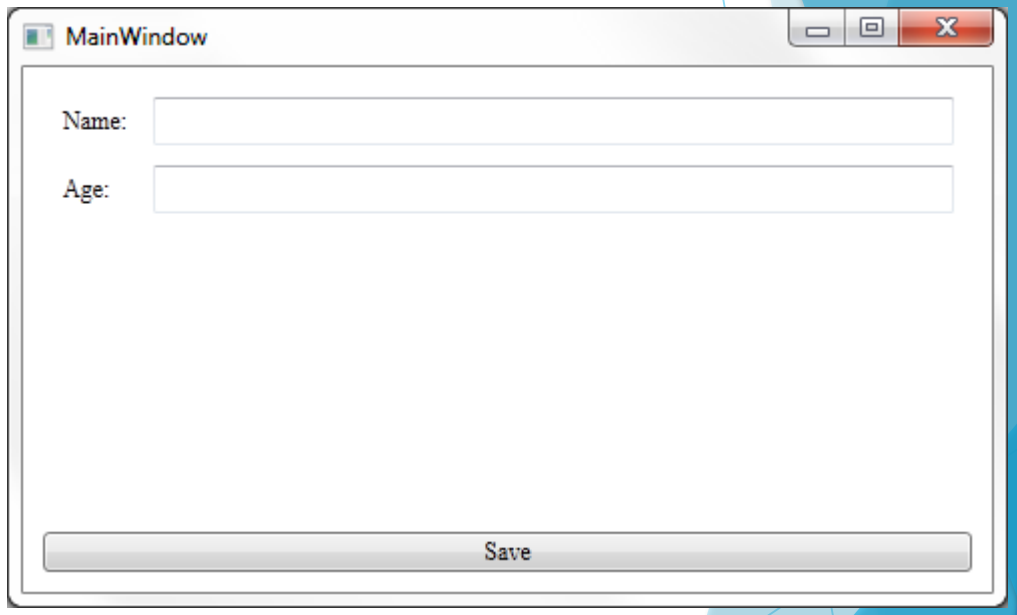
2023

Wiązanie (binding) danych

Ręczna synchronizacja danych – bez wiązania

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

}
```

```
<TextBox Name="txtName"
         TextChanged="txtName_TextChanged">
</TextBox>

<TextBox Name="txtAge"
         TextChanged="txtName_TextChanged">
</TextBox>

<Button Name="SaveButton"
        Click="SaveButton_Click">Save
</Button>
```

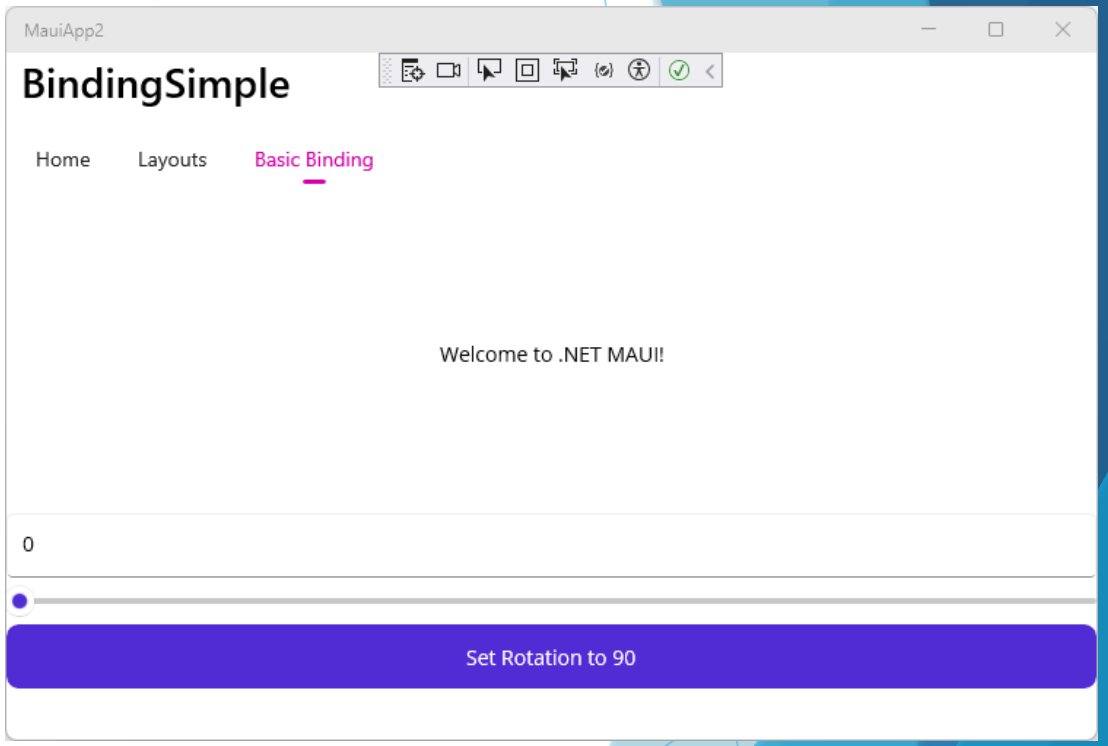


Wiązanie danych

- jest to związek, który mówi skąd dane powinny zostać pobrane (obiekt źródłowy) aby ustawić właściwość w obiekcie docelowym
- właściwość docelowa jest zawsze *BindableProperty*
- najprostszym zastosowaniem jest wiązanie dwóch takich właściwości
- realizacja wiązania poprzez klasę `Binding` z pakietu `Microsoft.Maui.Controls`
- niepoprawne wiązanie nie generuje wyjątków

Proste wiązanie właściwości

```
<VerticalStackLayout>

    <Label
        x:Name="WelcomeLabel"
        Text="Welcome to .NET MAUI!"
        VerticalOptions="Center"
        HorizontalOptions="Center"
        Rotation="{Binding Source={x:Reference rotationSlider},
                           Path=Value }"
        Padding="100"
        />
    <Entry Text="{Binding Source={x:Reference rotationSlider},
                          Path=Value}"/>
    <Slider x:Name="rotationSlider"  Minimum="0" Maximum="360" />
    <Button Text="Set Rotation to 90" Clicked="Button_Clicked"/>
</VerticalStackLayout>
```



Alternatywnie można utworzyć to samo wiązanie z poziomu kodu C#:

```
public BindingSimple()
{
   InitializeComponent();
   Binding binding = new Binding();
   binding.Source = rotationSlider;
   binding.Path = "Value";
   WelcomeLabel.SetBinding(Slider.RotationProperty, binding);
}
```

Tryby wiązania

„Ręczne” ustawienie wartości - zerwanie wiązania!

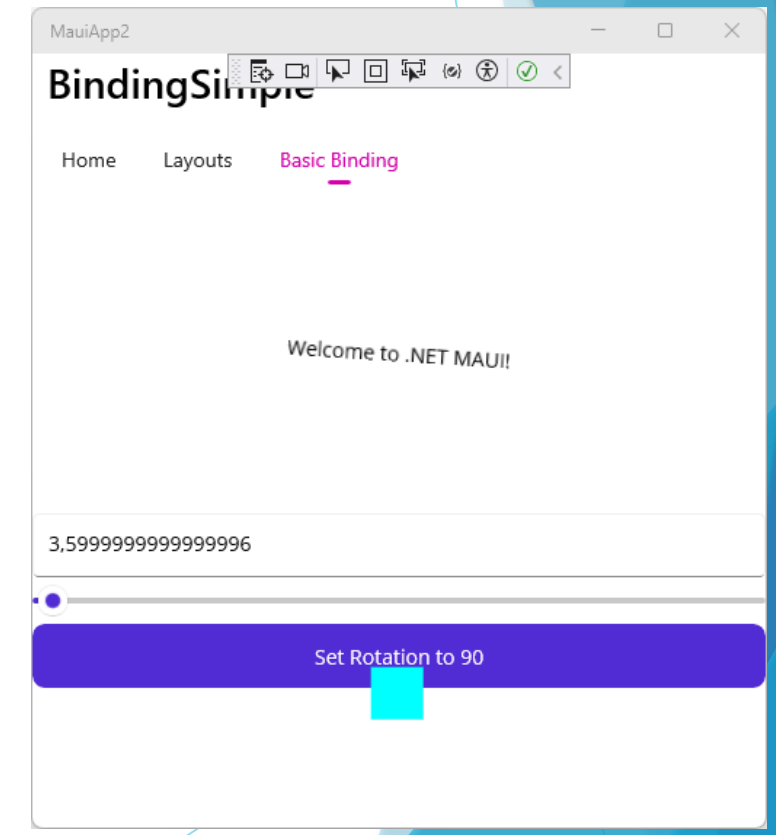
```
private void Button_Clicked(object sender, EventArgs e)
{
    //rotationSlider.Value = 90;
    WelcomeLabel.Rotation = 90;
}
```

Ustawienie wiązania dwukierunkowego:

```
Rotation="{Binding Source={x:Reference rotationSlider},
                   Path=Value,
                   Mode=TwoWay}"
```

A co będzie, gdy dane będą sprzeczne?

```
<BoxView x:Name="AquaBox" Color="Aqua" HeightRequest="10" WidthRequest="10"
     ScaleX="{Binding Source={x:Reference rotationSlider},
                      Path=Value,Mode=TwoWay}"
     ScaleY="{Binding Source={x:Reference rotationSlider},
                      Path=Value,Mode=TwoWay}"></BoxView>

private void Button_Clicked(object sender, EventArgs e)
{
   WelcomeLabel.Rotation = 90;
   AquaBox.ScaleX = 2;
   AquaBox.ScaleY = 1;
}
```



Tryby wiązania

`BindingMode`

- `OneWay` - jednostronne od źródła do celu
- `TwoWay` - dwustronne
- `OneTime` – jednorazowe od źródła do celu ale tylko gdy zmienia się BindingContext
- `OneWayToSource` – jednostronne od celu do źródła
- `Default` - domyślne – zależne od właściwości

Wiązanie do obiektów nie będących elementami wizualnymi

- dane muszą być publicznymi właściwościami obiektu
- dopuszczalne są następujące rozwiązania:
  - Source – wiązanie do obiektu
  - RelativeSource – pozwala wiązać właściwości do właściwości obiektów w hierarchii komponentów
  - BindingContext – wiązanie do właściwości `BindingContext` komponentu – źródłem zostaje pierwszy element w hierarchii który ma wartość różną od `null`

Wiązanie typu *Source*

```
namespace MauiApp2;

public partial class BindingSimple : ContentPage
{
    public double DefaultRotation { get; set; } = 45.0;

    public BindingSimple()
    {
        InitializeComponent();

    }
}
```

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace=MauiApp2"
             x:Class="MauiApp2.BindingSimple"
             x:Name="MyPage"
             Title="BindingSimple">
    <VerticalStackLayout>

        <Label
            x:Name="WelcomeLabel"
            Text="Welcome to .NET MAUI!"
            VerticalOptions="Center"
            HorizontalOptions="Center"
            Rotation="{Binding Source={x:Reference MyPage}, Path=DefaultRotation}"
            Padding="100"
            />
```

Wiązania typu RelativeSource

Tryb wyszukiwania:

- Self – wiązanie do innej właściwości tego samego obiektu
- FindAncestor – wiązanie do rodzica. Podaje się AncestorType – czyli jakiego typu rodzica szukamy, a następnie opcjonalnie AncestorLevel w celu pominięcia zadanej liczby wystąpień tego typu
- TemplatedParent – używane w szablonach

Wiązania typu `RelativeSource` (2)

- Wiązanie do siebie samego albo elementu nadrzędnego na nieznanym poziomie w hierarchii
- Wiązanie tego typu wykorzystuje obiekt klasy `RelativeSource`

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:MauiApp2"
             x:Class="MauiApp2.BindingSimple"
             x:Name="MyPage"
             Title="BindingSimple">

    <VerticalStackLayout>

        <Label Text="{Binding Source={x:RelativeSource Mode=FindAncestor,
                                                       AncestorType={x:Type ContentPage}},
                              Path=Title,
                              StringFormat='{0}'}">
        </Label>


    </VerticalStackLayout>
</ContentPage>
```

Wiązania typu BindingContext

```
<ContentPage.Resources>
    <p:Person x:Key="PersonKowalski" Id="1" Name="Jan Kowalski" Title="Pan" ></p:Person>
</ContentPage.Resources>
```

```
 <HorizontalStackLayout>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Id}"></Label>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Title}"></Label>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Name}"></Label>
 </HorizontalStackLayout>
```

```
<HorizontalStackLayout BindingContext="{Binding Source={StaticResource Key=PersonKowalski}}">
    <Label Text="{Binding Id}"></Label>
    <Label Text="{Binding Title}"></Label>
    <Label Text="{Binding Name}"></Label>
</HorizontalStackLayout>
```

Dane są wyszukiwane w hierarchii komponentów do czasu napotkania właściwości BindingContext różnej od null.

Konwersja powiązanych danych

- Można wyróżnić dwa rodzaje konwersji:
  - **formatowanie tekstów** – gdy dane są w postaci tekstowej – zazwyczaj zawierają liczby lub daty
  - **konwertery wartości** (*value converters*) – pozwalające na konwersję danych dowolnego typu

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:MauiApp2"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"


 <Label Text="{Binding Source={x:Static sys:DateTime.Now},StringFormat='Data: {0:yyyy-MMMM-dd}'}"></Label>
```

Konwertery wartości

```
namespace MauiApp2
{
    public class BoolToColorConverter : IValueConverter
    {
        public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
        {
            bool val = (bool)value;

            if ( val == true)
                return new Color(0, 255, 0);

            else
                return new Color(255, 0, 0);
        }

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

```
<ContentPage.Resources>
    <p:BoolToColorConverter x:Key="BTCConverter"/>
</ContentPage.Resources>


<CheckBox x:Name="ColorCheckBox"/>

 <Label  Text="To jest tekst" TextColor="{Binding Source={x:Reference ColorCheckBox},
                                                  Path=IsChecked,
                                                  Converter={StaticResource BTCConverter}}">
```

Konwertery wartości (2)

```
public class ValueToColorConverter:IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {

        double val = (double)value;
        if (val < (double) parameter)
        {
            return new SolidColorBrush(Colors.Red);
        }
        else
        {
            return new SolidColorBrush(Colors.Black);
        }
    }

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

Konwertery wartości (3)

```
<Button Name="SetScaleButton" Click="SetScaleButton_Click"  >
     <Button.Foreground>
          <Binding ElementName="ScaleSlider" Path="Value" Converter="{StaticResource ColorConverter}">

               <Binding.ConverterParameter>
                    <sys:Double>
                        0.5
                    </sys:Double>
               </Binding.ConverterParameter>

          </Binding>
      </Button.Foreground>
      Set Scale = 1
</Button>
```

Konwertery wartości (4)

```
public class ValueToColorConverter : IValueConverter
{
    public double MinWarningThreshold { get; set; }
    public double MaxWarningThreshold { get; set; }

    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {

        double val = (double)value;
        if (val < MinWarningThreshold || val > MaxWarningThreshold)
        {
            return new Color(0, 0, 0);
        }
        else
        {
            return new Color(255, 0, 0);
        }
    }

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

Konwertery wartości (4)

```
<Label Text="Jakiś tam tekst">
    <Label.TextColor>
        <Binding Source="{x:Reference rotationSlider}" Path="Value">
            <Binding.Converter>
                <p:ValueToColorConverter MinWarningThreshold="100" MaxWarningThreshold="200"/>
            </Binding.Converter>
        </Binding>
    </Label.TextColor>
</Label>
```

Konwertery wielu wiązań (*multibinding*)

```
public class ValueToColorConverter:IMultiValueConverter
   {
       public double MinWarningThreshold { get; set; }
       public double MaxWarningThreshold { get; set; }

       public object Convert(object[] values, Type targetType, object parameter, System.Globalization.CultureInfo culture)
       {
           bool isChecked = (bool)values[0];
           double val = (double)values[1];
           if ( isChecked && ( val < MinWarningThreshold || val > MaxWarningThreshold))
           {
               return new SolidColorBrush(Colors.Red);
           }
           else
           {
               return new SolidColorBrush(Colors.Black);
           }

       }

       public object[] ConvertBack(object value, Type[] targetTypes, object parameter, System.Globalization.CultureInfo culture)
       {
           throw new NotImplementedException();
       }
   }
```

Konwertery wielu wiązań (*multibinding*)

```
<Button Name="SetScaleButton" Click="SetScaleButton_Click"  >
     <Button.Foreground>
         <MultiBinding >
             <MultiBinding.Converter>
                 <local:ValueToColorConverter MinWarningThreshold="0.5" MaxWarningThreshold="1.5"/>
             </MultiBinding.Converter>
             <Binding ElementName="IsScaleWarningEnabled" Path="IsChecked"/>
             <Binding ElementName="ScaleSlider" Path="Value"/>
         </MultiBinding>
     </Button.Foreground>

     Set Scale = 1</Button>

  <CheckBox Name="IsScaleWarningEnabled"  Content="scale warning"/>
```

Bindable Property

BindableProperty

- Praktycznie całe .NET MAUI wykorzystuje obiekty typu BindableProperty (zamiast np. pól różnego typu) do działania.
- Są nimi wszystkie (?) właściwości kontrolek
- Korzyści:
  - Wsparcie wiązania danych (data binding)
  - Obsługa styli
  - Obsługa szablonów
  - Wartości dziedziczone od rodziców
  - Wartości domyślne właściwości
  - Walidacja wartości właściwości
  - Mechanizm wywołania zwrotnego (callback) monitorujący zmiany wartości właściwości

BindableProperty (2)

- Obiekty w aplikacji .NET MAUI powinny używać właściwości BindableProperty (zamiast normalnych) wtedy, gdy:
  - mają mieć możliwość bycia poprawnym celem (target) wiązania danych
  - ma istnieć możliwość ustawienia wartości tej właściwości poprzez style
  - mają posiadać wartości domyślne, które są inne niż wartość domyślna dla typu
  - wartość właściwości ma być walidowana
  - ma być możliwość monitorowania zmian właściwości
- Wszystkie właściwości BindableProperty są definiowane jako statyczne `public static readonly BindableProperty`
- W WPF-ie podobną rolę pełniły DependencyProperty, ale tamten mechanizm był nawet silniejszy

Tworzenie właściwości BindableProperty

- Utworzenie nowej właściwości składa się z dwóch kroków:
  - Utworzenie instancji BindableProperty wykorzystując jedną z przeciążonych metod BindableProperty.Create
  - Zdefiniowanie poprawnych metod dostępu do właściwości
- BP musi być utworzona w głównym wątku (tzw. UI thread)
- Obiekt, który zawiera BP musi dziedziczyć po BindableObject (najwyższy obiekt w hierarchii)
- Tworzenie właściwości BP ma sens głównie (tylko?) w przypadku projektowania własnych kontrolek

Tworzenie właściwości BindableProperty (2)

- Tworząc nową BP trzeba podać minimalnie następujące parametry:
  - nazwę właściwości – z założenia nazwa obiektu BP jest nazwą właściwości z dodanym sufiksem Property
  - typ właściwości
  - typ obiektu właściciela
  - wartość domyślną właściwości zwracaną wtedy, gdy nie została ustawiona inna wartość
- Wartość domyślna zostanie również nadana po wywołaniu metody ClearValue()

```
public static readonly BindableProperty IsExpandedProperty =
        BindableProperty.Create ( "IsExpanded",
                                  typeof(bool),
                                  typeof(Expander),
                                  false);
```

Tworzenie właściwości BindableProperty (3)

- Opcjonalnie można podać jeszcze następujące atrybuty:
  - domyślny tryb wiązania danych (BindingMode)
  - delegat – funkcja walidująca dane
  - delegat – funkcja wywoływana w gdy wartość właściwości została zmieniona
  - delegat – funkcja wywoływana gdy wartość będzie zmieniona
  - delegat – funkcja CoerceValue
  - delegat Func służący do inicjalizacji wartości domyślnej właściwości
- Tworzenie metod dostępowych – wykorzystanie metod GetValue i SetValue

```
public bool IsExpanded
{
        get => (bool)GetValue(IsExpandedProperty);
        set => SetValue(IsExpandedProperty, value);
}
```

Tworzenie właściwości BindableProperty (4) - PropertyChanged

```
public static readonly BindableProperty IsExpandedProperty =
    BindableProperty.Create(nameof(IsExpanded),
    typeof(bool),
    typeof(Expander),
    false,
    propertyChanged: OnIsExpandedChanged);
...

    static void OnIsExpandedChanged(BindableObject bindable,
                object oldValue,
                object newValue)
    {
        // Property changed implementation goes here
    }
```

Tworzenie właściwości BindableProperty (5) - validation

- Sprawdzenie poprawności ustawianej wartości
- Zwrócenie false powoduje zgłoszenie wyjątku

```
public static readonly BindableProperty AngleProperty =
        BindableProperty.Create("Angle",
                                typeof(double),
                                typeof(MainPage),
                                0.0,
                                validateValue: IsValidValue);


static bool IsValidValue(BindableObject view, object value)
{
    double result;
    double.TryParse(value.ToString(), out result);
    return (result >= 0 && result <= 360);
}
```

Tworzenie właściwości BindableProperty (6) - coerce

- metoda próbuje naprawić wartości właściwości – wykorzystanie np. przy zależności jednej właściwości od innej

```
public static readonly BindableProperty AngleProperty =
        BindableProperty.Create("Angle", typeof(double), typeof(MainPage), 0.0, coerceValue: CoerceAngle);

public static readonly BindableProperty MaximumAngleProperty =
        BindableProperty.Create("MaximumAngle", typeof(double), typeof(MainPage), 360.0, propertyChanged: ForceCoerceValue);


static object CoerceAngle(BindableObject bindable, object value)
{
    MainPage page = bindable as MainPage;
    double input = (double)value;

    if (input > page.MaximumAngle)
              input = page.MaximumAngle;
    return input;
}

static void ForceCoerceValue(BindableObject bindable, object oldValue, object newValue)
{
    bindable.CoerceValue(AngleProperty);
}
```

Tworzenie właściwości BindableProperty (7) - podsumowanie

- W przypadku DependencyProperty z WPF wartości przechowywały nadaną wartość (jako *desired*) a końcowa wartość wynikała z nałożonych innych ograniczeń: styli, triggerów, działania funkcji Coerce itd. Natomiast wartość przypisana była pamiętana. W przypadku .NET MAUI tak nie jest
- W .NET MAUI znacząco uproszczono zdarzenia, co pewnie pośrednio wynika z działania platform docelowych.
- Sposób działania BindableProperty ma znaczenie dla zrozumienia jak całe .NET MAUI działa, natomiast definiowanie własnych właściwości tego typu w większości przypadków ogranicza się do tworzenia własnych komponentów

Wiązanie kolekcji

- Trochę więcej pracy wymaga powiązanie kolekcji do elementu kontrolnego
- Dwa podstawowe komponenty do wyświetlania zawartości kolekcji to: ListView i CollectionView

```
namespace MauiApp1
{
    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Title { get; set; }

    }
}
```

```
MauiApp1.Person
MauiApp1.Person
MauiApp1.Person
MauiApp1.Person
```

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.SimpleCollectionView"
             xmlns:local="clr-namespace:MauiApp1"
             Title="SimpleCollectionView">
    <ContentPage.Resources>
        <x:Array x:Key="PeopleTable" Type="{x:Type local:Person}">
            <local:Person Id="1" Name="Jan Kowalski" Title="inż"/>
            <local:Person Title="dr"   Id="2" Name="Jurek" ></local:Person>
            <local:Person Title="prof" Id="3" Name="Jarek" ></local:Person>
            <local:Person Title="prof" Id="4" Name="Anna" ></local:Person>
        </x:Array>
    </ContentPage.Resources>
    <VerticalStackLayout>
        <ListView ItemsSource="{StaticResource PeopleTable}"/>
    </VerticalStackLayout>
</ContentPage>
```

Wiązanie kolekcji (2)

- Domyślnie wywoływana jest metoda ToString()

```
public class Person
{
    …
    public override string ToString()
    {
        return Title+" "+Name;
    }

}
```

```
inż Jan Kowalski
dr Jurek
prof Jarek
prof Anna
```

- Można również zdefiniować szablon dla poszczególnych elementów

```
<ListView ItemsSource="{StaticResource PeopleTable}" >
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <Border Stroke="Blue" StrokeThickness="3">
                    <HorizontalStackLayout Spacing="10">
                        <Image Source="dotnet_bot.png"  WidthRequest="50"/>
                        <Label Text="{Binding Title}" FontSize="20" TextColor="Red"></Label>
                        <Label Text="{Binding Name}" TextColor="Black"></Label>
                    </HorizontalStackLayout>
                </Border>
            </ViewCell>
        </DataTemplate>

    </ListView.ItemTemplate>
</ListView>
```

Przeniesienie szablonu danych do zasobów

```
<ContentPage.Resources>
    <x:Array x:Key="PeopleTable" Type="{x:Type local:Person}">
        <local:Person Id="1" Name="Jan Kowalski" Title="inż"/>
        <local:Person Title="dr"   Id="2" Name="Jurek" ></local:Person>
        <local:Person Title="prof" Id="3" Name="Jarek" ></local:Person>
        <local:Person Title="prof" Id="4" Name="Anna" ></local:Person>
    </x:Array>
    <DataTemplate x:Key="PersonListTemplate">
        <ViewCell>
            <Border Stroke="Blue" StrokeThickness="3">
                <HorizontalStackLayout Spacing="10">
                    <Image Source="dotnet_bot.png"  WidthRequest="50"/>
                    <Label Text="{Binding Title}" FontSize="20" TextColor="Red"></Label>
                    <Label Text="{Binding Name}" TextColor="Black"></Label>
                </HorizontalStackLayout>
            </Border>
        </ViewCell>
    </DataTemplate>
</ContentPage.Resources>

<VerticalStackLayout>
    <ListView ItemsSource="{Binding Source={x:Reference CollViewPage}, Path=People }"
              ItemTemplate="{StaticResource PersonListTemplate}" />
 </VerticalStackLayout>
```

Wiązanie kolekcji (3)

- Kolekcja może być właściwością klasy. Załóżmy, że w klasie głównej widoku mamy następującą właściwość:

```
public partial class PropertyChanged : ContentPage
{

    public List<Person> People { get; set; } =
    new List<Person>() {
            new Person() { Id = 0, Title= "dr", Name ="Jurek" },
            new Person() { Id = 1, Title= "inż", Name ="Mamoń" },
            new Person() { Id = 2, Title= "prof", Name ="Jarek" },
            new Person() { Id = 3, Title= "prof", Name ="Anna" },
    };
…
```

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.PropertyChanged"
             x:Name="CollViewPage"
             Title="PropertyChagned">
<VerticalStackLayout>
        <ListView ItemsSource="{Binding Source={x:Reference CollViewPage}, Path=People }" >
…
```

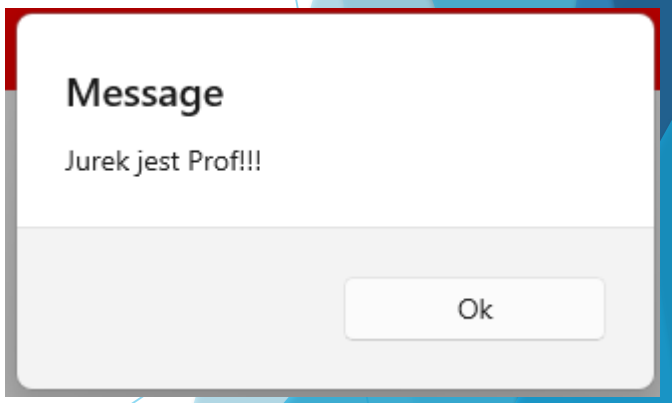
```
private void Button_Clicked(object sender, EventArgs e)
{
    People[0].Title = "Prof!!!";
    DisplayAlert("Message", People[0].Name + " jest " + People[0].Title , "Ok");
```

Wiązanie kolekcji (4) – modyfikowanie obiektów

- A co się stanie, gdy zmodyfikujemy właściwości któregoś z obiektów?

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp1.PropertyChanged"
             x:Name="CollViewPage"
             Title="PropertyChagned">
<VerticalStackLayout>
        <ListView ItemsSource="{Binding Source={x:Reference CollViewPage}, Path=People }" >
…
        <Button Text="Change Person" Clicked="Button_Clicked"   />
```

```
private void Button_Clicked(object sender, EventArgs e)
{
    People[0].Title = "Prof!!!";
    DisplayAlert("Message", People[0].Name + " jest " + People[0].Title , "Ok");
}
```



- Wpisy na liście nie ulegają zmianie

Informowanie o zmianie stanu obiektu - INotifyPropertyChanged

- Interfejs INotifyPropertyChanged (System.ComponentModel) ma za zadanie informować o zmianach, które zaszły w obiektcie
- Ma tylko jedną składową

```
event System.ComponentModel.PropertyChangedEventHandler? PropertyChanged;
```

```
public class Person:INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    private string title;

    public string Title
    {
        get => title;
        set
        {
            title = value;
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Title)));
        }
    }
    public int Id { get; set; }
    public string Name { get; set; }

    public override string ToString() => Title+" "+Name;
}
```

Oczywiście WSZYSTKIE właściwości powinny informować o zmianach!!!

INotifyPropertyChanged – inna implementacja

```
public class Person:INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    private string title;

    public string Title
    {
        get => title;
        set
        {
            title = value;
            NotifyPropertyChanged(nameof(Title));
            //NotifyPropertyChanged();
        }
    }
    public int Id { get; set; }
    public string Name { get; set; }

    private void NotifyPropertyChanged([CallerMemberName] string propertyName = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    public override string ToString() => Title+" "+Name;
}
```

To wywołanie (bez parametru) też by zadziałało bo jest to

Zmiany w kolekcjach danych

- A co się stanie, jeśli dodatkowo oprócz zmiany wartości właściwości jednego z obiektów, dodamy (albo usuniemy) obiekt do (z) listy?

```
private void Button_Clicked(object sender, EventArgs e)
{
    People[0].Title = "Prof!!!";
    DisplayAlert("Message", People[0].Name + " jest " + People[0].Title , "Ok");
    People.Add(new Person() { Id = 4, Title = "mgr", Name = "Halina" });
}
```

- Oczywiście zmiany nie będą widoczne
- Kolekcje powinny również informować o zmianach -> typ ObservableCollection<T> (System.Collections.ObjectModel) – implementuje ona interfejs INotifyCollectionChanged

Zmiany w kolekcjach danych – ObservableCollection<T>

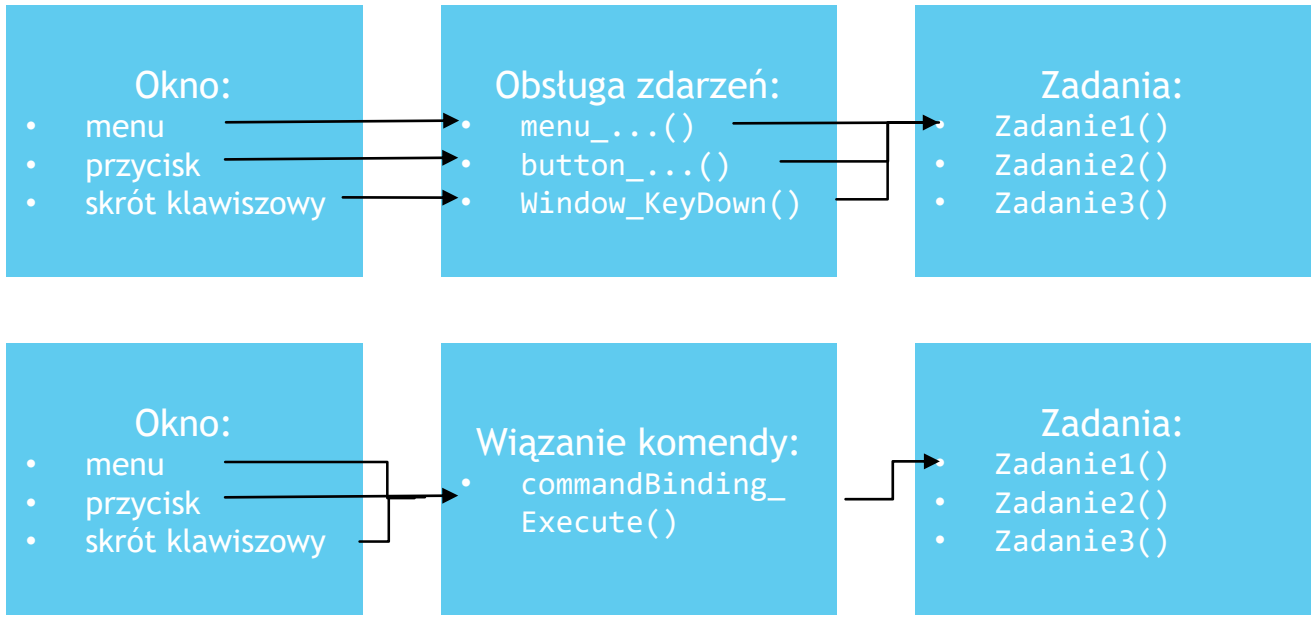
```
    private ObservableCollection<Person> people;
    public ObservableCollection<Person> People
    {
        get => people;
        //set => people = value;
    }

    public PropertyChanged()
    {

        people = new ObservableCollection<Person>()
        {
            new Person() { Id = 0, Title= "dr", Name ="Jurek" },
            new Person() { Id = 1, Title= "inż", Name ="Mamoń" },
            new Person() { Id = 2, Title= "prof", Name ="Jarek" },
            new Person() { Id = 3, Title= "prof", Name ="Anna" },

        };
        InitializeComponent();
    }

    private void Button_Clicked(object sender, EventArgs e)
    {
        People[0].Title = "Prof!!!";
        DisplayAlert("Message", People[0].Name + " jest " + People[0].Title , "Ok");
        People.Add(new Person() { Id = 4, Title = "mgr", Name = "Halina" });
    }
}
```

A co by się stało, gdybyśmy podmienili całą ObservableCollection ?

System komend (command)

Komendy - idea

- Funkcjonalność aplikacji może zostać podzielona na zadania.
- Każde z zadań może zostać wykonane na wiele sposobów



Komendy – zalety i wady

- Podstawowe zalety:
  - oddelegowanie zdarzeń do odpowiednich komend
  - synchronizacja stanu kontrolek ze stanem związanej komendy
  - część kontrolek obsługuje komendy automatycznie
- Największe ograniczenia (wady)
  - brak mechanizmów śledzenia historii wywoływania komend
  - brak mechanizmu cofnięcia wykonania komendy (*Undo*)
  - ograniczone możliwości stanu komendy

Interfejs ICommand

```
public interface ICommand
{
    void Execute(object parameter);
    bool CanExecute(object parameter);

    event EventHandler CanExecuteChanged;
}
```

- `Execute` – zawiera logikę komendy
- `CanExecute` – stan komendy = wartość `true` jeśli komendę można wywołać i `false` w przeciwnym przypadku
- `CanExecuteChanged` – zdarzenie wywoływane gdy zmieni się stan komendy – jest to sygnał dla wszystkich elementów kontrolnych używających komendy, żeby sprawdziły jej stan przez wywołanie metody `CanExecute()`

Model MVVM (Model-View-ViewModel)

MVVM (Model-View-ViewModel)

- Wzorzec Model-View-ViewModel jest wzorcem bazującym na MVC (Model-View-Contoller)
- Pozwala odizolować logikę aplikacji od interfejsów użytkownika i danych

Model

- obejmuje logikę biznesową i dane
- z reguły zawiera kod operujący na danych (dostęp do nich, przechowywanie itp.)
- często ta warstwa będzie oddzielnym serwisem/klasą
- powinna być na tyle niezależna, żeby można było jej użyć w innym miejscu – np. do budowy innego rodzaju interfejsu
- nie modyfikuje bezpośrednio ani widoku ani modelu widoku

Widok (View)

Cechy charakterystyczne:

- zawiera elementy interfejsu (Control, UserControl), szablony danych i style
- połączenie z warstwą ViewModel poprzez właściwość DataContext
- elementy kontrolne są powiązane z właściwościami i komendami ViewModelu
- w widoku może konwertować dane oraz można dołączać elementy walidacji danych
- definiuje i obsługuje pewne zachowania związane z prezentacją danych (np. animacje)
- zawartość powinna być (w miarę możliwości) zawarta w kodzie XAML, może jednak zawierać kod, który jest trudny/niemożliwy do realizacji w XAMLu albo taki, który wymaga bezpośredniego odwołania się do elementów interfejsu użytkownika

Widok modelu (ViewModel)

- nie ma referencji do widoku (View)
- definiuje właściwości i komendy, które mogą być związane z interfejsami użytkownika w widoku
- informuje o zmianach stanu danych (INotifyPropertyChanged/INotifyCollectionChanged)
- koordynuje interakcję widoku z danymi
- może łączyć wiele modeli (relacja jeden do wielu)
- definiuje dodatkowe właściwości, których nie ma model
- może zawierać dodatkowe sprawdzanie poprawności danych
- może udostępniać model bezpośrednio do widoku lub może go specjalnie opakowywać (np. w zależności od tego, czy klasa modelu implementuje interfejs INotifyPropertyChanged)
- może definiować stany logiczne prezentowane użytkownikowi