Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

Microsoft

Julian Myrcha
Institute of Computer Science
26.02.2025

Jak to było do tej pory:

- **User32** - wygląd kontrolek (przycisk, pole tekstowe, okienko)
- **GDI/GDI+** - rysowanie i pisanie

Co to oznaczało:

- Wykorzystanie bezpośrednie kłopotliwe (Charles Petzold)
- Wydajność słaba (interfejs został opracowany w początkach windows, kiedy system był 16 bitowy a monitory wyświetlały kolorowy obraz z 4 elementowej palety)
- Wszystkie technologie przed WPF to tak naprawdę nakładki ...
- Wydajny jest DirectX, bo pisano w nim gry
- Ale co z tego, skoro jest zbyt skomplikowany do aplikacji biznesowych

- wykorzystuje `User32` w bardzo ograniczonym zakresie
- rysowane wszystkiego w DirectX
- Przy starcie sprawdza kartę graficzną:
  - **tier-0** - DirectX 9.0 lub brak karty, rysujemy programowo
  - **tier-1** - DirectX =9.0, częściowe wsparcie
  - **tier-2** - DirectX =9.0, duże wsparcie

XAML - (**Ex**tensible **A**pplication **M**arkup **L**anguage)

Wymiarowanie
jakie jest DPI:

```
[DPI] = sqrt(1920^2+1080^2)/17 cali=129.58dpi
```

czyli

```
[Physical Unit Size] = [Device-Independent Unit Size] x [System DPI]
= 1/96 inch x 130 dpi = 1.35 pixels
```

czyli okno 300x200 ma rozmiar **405x270** pikseli

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Windows Presentation Framework to wynik projektu firmy Microsoft, którego celem było stworzenie nowej warstwy prezentacji opartej na języku opisu graficznego o nazwie kodowej Avalon
- WPF jest ograniczony do systemu Windows
- Technologie z WPF są wykorzystywane w innych technologiach (np. Xamarin, który bazuje na Mono)
- Microsoft tworzy Maui
- Avalonia to niezależny projekt open source WPF przeznaczony do nowoczesnych aplikacji
  - macOS, Linux i Windows z natywnym wsparciem dla x86 i x64, a także ARM
- Avalonia nie jest bezpośrednim portem WPF, choć ma wspólne kluczowe koncepcje
  - używa XAML
  - używa arkuszy styli CSS
  - używa własnych kolekcji

- Uruchom polecenie w wierszu poleceń:

```
dotnet new -i Avalonia.Templates
The following template packages will be installed:
   Avalonia.Templates

Success: Avalonia.Templates::0.10.18.7 installed the following templates:
Template Name       Short Name                 Language  Tags
------------------  -------------------------  --------  ----------------------------------------
Avalonia .NET C...  avalonia.app               [C#],F#   Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia .NET C...  avalonia.mvvm              [C#],F#   Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia Cross ...  avalonia.xplat             [C#],F#   Desktop/Xaml/Avalonia/Web/Mobile
Avalonia Resour...  avalonia.resource                    Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia Styles     avalonia.styles                      Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia Templa...  avalonia.templatedcontrol  [C#],F#   Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia UserCo...  avalonia.usercontrol       [C#],F#   Desktop/Xaml/Avalonia/Windows/Linux/macOS
Avalonia Window     avalonia.window            [C#],F#   Desktop/Xaml/Avalonia/Windows/Linux/macOS
```

- Uruchom polecenie w wierszu poleceń:
- W rider zainstaluj `File-Settings-Plugin`:
  - AvaloniaReader
  - JavaFx Runtime for plugins
- Avalonia umożliwia tworzenie aplikacji przy użyciu języka znaczników XAML i kodu C# (lub innego języka .NET)

- Uruchom polecenie w wierszu poleceń:
- W rider zainstaluj `File-Settings-Plugin`:
  - AvaloniaReader
  - JavaFx Runtime for plugins
- Avalonia umożliwia tworzenie aplikacji przy użyciu języka znaczników XAML i kodu C# (lub innego języka .NET)

- Uruchom polecenie w wierszu poleceń:
- W rider zainstaluj `File-Settings-Plugin`:
  - AvaloniaReader
  - JavaFx Runtime for plugins
- Avalonia umożliwia tworzenie aplikacji przy użyciu języka znaczników XAML i kodu C# (lub innego języka .NET)

- Uruchom polecenie w wierszu poleceń:
- W rider zainstaluj `File-Settings-Plugin`:
  - AvaloniaReader
  - JavaFx Runtime for plugins
- Avalonia umożliwia tworzenie aplikacji przy użyciu języka znaczników XAML i kodu C# (lub innego języka .NET)

- Większość kodu reaguje na coś, co uległo zmianie
  - C# events, callbacks, delegates, lambdas, observables, notifications oraz bindings

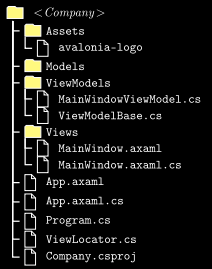
- Większość kodu reaguje na coś, co uległo zmianie
  - C# events, callbacks, delegates, lambdas, observables, notifications oraz bindings

- utworzenie projektu

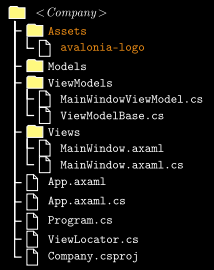
```
jmy@uzen:~/Private/2023L/WPF
The template "Avalonia .NET Core MVVM App" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /home/jmy/Private/2023L/WPF/Company/Company.csproj...
  Determining projects to restore...
  Restored /home/jmy/Private/2023L/WPF/Company/Company.csproj (in 11.38 sec).
Restore succeeded.
```

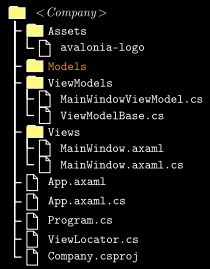
- utworzenie projektu
  - utworzone pliki



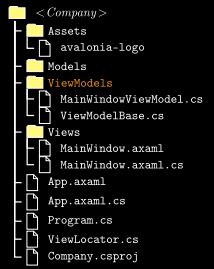
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.



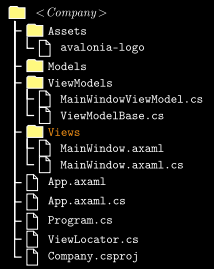
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele



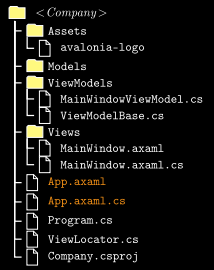
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele
  ViewModels - viewmodele okienek aplikacji



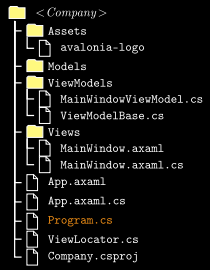
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele
  ViewModels - viewmodele okienek aplikacji
  Views - okienka aplikacji



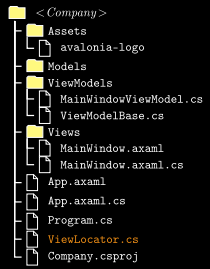
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele
  ViewModels - viewmodele okienek aplikacji
  Views - okienka aplikacji
  App.axaml - plik, w którym zostaną umieszczone style i szablony XAML, które będą miały zastosowanie do całej aplikacji



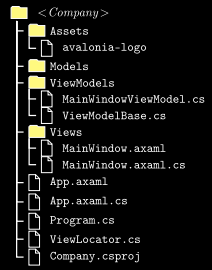
- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele
  ViewModels - viewmodele okienek aplikacji
  Views - okienka aplikacji
  App.axaml - plik, w którym zostaną umieszczone style i szablony XAML, które będą miały zastosowanie do całej aplikacji
  Program.cs - plik startowy, konfiguracja opcji programu



- utworzenie projektu
  Assets - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  Models - nasze modele
  ViewModels - viewmodele okienek aplikacji
  Views - okienka aplikacji
  App.axaml - plik, w którym zostaną umieszczone style i szablony XAML, które będą miały zastosowanie do całej aplikacji
  Program.cs - plik startowy, konfiguracja opcji programu
  ViewLocator.cs - wyszukiwanie widoków dla viewmodeli



- utworzenie projektu
  - **Assets** - Pliki umieszczone w tym katalogu zostaną automatycznie uwzględnione jako zasoby w aplikacji.
  - **Models** - nasze modele
  - **ViewModels** - viewmodele okienek aplikacji
  - **Views** - okienka aplikacji
  - **App.axaml** - plik, w którym zostaną umieszczone style i szablony XAML, które będą miały zastosowanie do całej aplikacji
  - **Program.cs** - plik startowy, konfiguracja opcji programu
  - **ViewLocator.cs** - wyszukiwanie widoków dla viewmodeli



format `*.axaml` jest taki sam jak `*.xaml` - został wprowadzony z powodu nieprawidłowego obsługiwania w Visual Studio

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="using:Company.ViewModels"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        x:Class="Company.Views.MainWindow"
        x:DataType="vm:MainWindowViewModel"
        Icon="/Assets/avalonia-logo.ico"
        Title="Company">
    <Design.DataContext>
        <!-- This only sets the DataContext for the previewer in an IDE -->
        <vm:MainWindowViewModel/>
    </Design.DataContext>
    <TextBlock Text="{Binding Greeting}" HorizontalAlignment="Center"
                                         VerticalAlignment="Center"/>
</Window>
```

```
using Avalonia.Controls;

namespace Company.Views;

public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

```xml
<Application xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="using:Company"
             x:Class="Company.App">

    <Application.DataTemplates>
        <local:ViewLocator/>
    </Application.DataTemplates>

    <Application.Styles>
        <FluentTheme Mode="Light"/>
    </Application.Styles>
</Application>
```

```
using Avalonia;
using Avalonia.Controls.ApplicationLifetimes;
using Avalonia.Markup.Xaml;
using Company.ViewModels;
using Company.Views;
namespace Company;
public partial class App : Application {
    public override void Initialize() {
        AvaloniaXamlLoader.Load(this);
    }
    public override void OnFrameworkInitializationCompleted() {
        if (ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop) {
            desktop.MainWindow = new MainWindow {
                DataContext = new MainWindowViewModel(),
            };
        }
        base.OnFrameworkInitializationCompleted();
    }
}
```

```
using ReactiveUI;

namespace Company.ViewModels;

public class ViewModelBase : ReactiveObject
{
}
```

```
namespace Company.ViewModels;

public class MainWindowViewModel : ViewModelBase
{
    public string Greeting => "Welcome to Avalonia!";
}
```

```cs
using Avalonia;
using Avalonia.ReactiveUI;
using System;

namespace Company;
class Program
{
    [STAThread]
    public static void Main(string[] args) => BuildAvaloniaApp()
        .StartWithClassicDesktopLifetime(args);

    // Avalonia configuration, don't remove; also used by visual designer.
    public static AppBuilder BuildAvaloniaApp()
        => AppBuilder.Configure<App>()
            .UsePlatformDetect()
            .LogToTrace()
            .UseReactiveUI();
}
```

```cs
using System;
using Avalonia.Controls;
using Avalonia.Controls.Templates;
using Company.ViewModels;
namespace Company;
public class ViewLocator : IDataTemplate {
    public IControl Build(object data) {
        var name = data.GetType().FullName!.Replace("ViewModel", "View");
        var type = Type.GetType(name);
        if (type != null) {
            return (Control)Activator.CreateInstance(type)!;
        }
        return new TextBlock { Text = "Not Found: " + name };
    }
    public bool Match(object data) {
        return data is ViewModelBase;
    }
}
```

- treść jest zazwyczaj określana za pomocą tagów ...
- ale może być wartością atrybutu Content

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="using:Company.ViewModels"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        x:Class="Company.Views.MainWindow"
        x:DataType="vm:MainWindowViewModel"
        Icon="/Assets/avalonia-logo.ico"
        Title="Company">
        Balbinka!
</Window>
```

- treść jest zazwyczaj określana za pomocą tagów ...
- ale może być wartością atrybutu Content

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="using:Company.ViewModels"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
        x:Class="Company.Views.MainWindow"
        x:DataType="vm:MainWindowViewModel"
        Icon="/Assets/avalonia-logo.ico"
        Title="Company"
        Content="Balbinka!">
</Window>
```

- utwórz kontrolkę użytkownika: `Company-add-avalonia user control` i wprowadź `EmployeeListView`

lub w linii komend:

```
dotnet new avalonia.usercontrol -o Views -n EmployeeListView  --namespace Company.Views
```

- utwórz kontrolkę użytkownika: `Company-add-avalonia user control` i wprowadź `EmployeeListView` Company/EmployeeListView.axaml

```
<UserControl xmlns="https://github.com/avaloniaui"
     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"
     mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
     x:Class="Company.EmployeeListView">
    Welcome to Avalonia!
</UserControl>
```

- utwórz kontrolkę użytkownika: `Company-add-avalonia user control` i wprowadź `EmployeeListView`

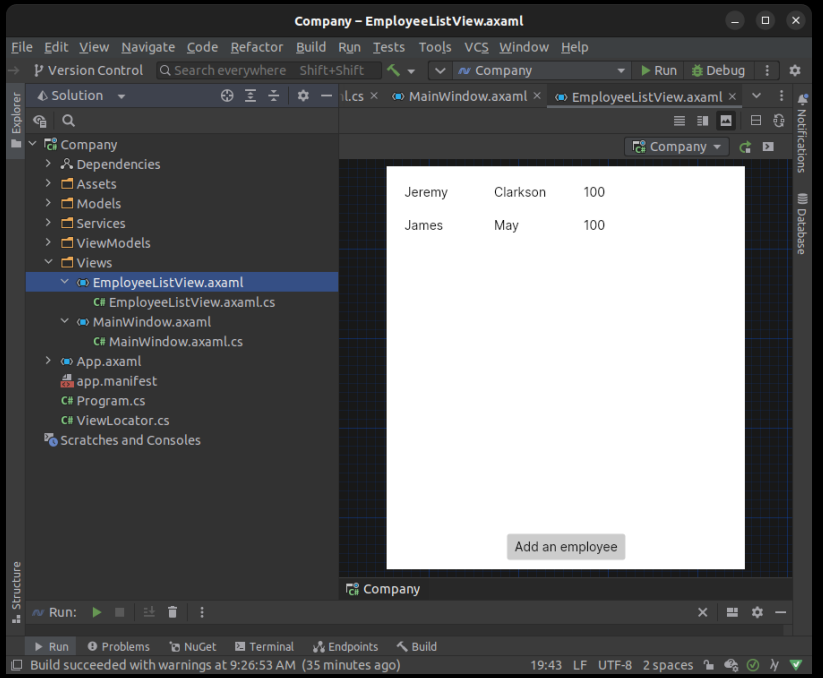
Company/EmployeeListView.axaml.cs

```
using Avalonia;
using Avalonia.Controls;
using Avalonia.Markup.Xaml;
namespace Company;
public partial class EmployeeListView : UserControl
{
    public EmployeeListView()
    {
        InitializeComponent();
    }
    private void InitializeComponent()
    {
        AvaloniaXamlLoader.Load(this);
    }
}
```

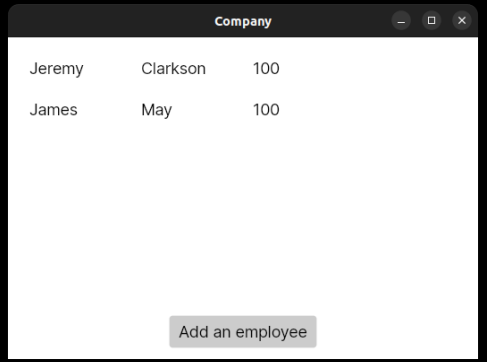
`mc:Ignorable="d"` informuje silnik XAML, że wpisy zaczynające się od d: można zignorować

```
<UserControl xmlns="https://github.com/avaloniaui"
     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"
     mc:Ignorable="d" d:DesignWidth="400" d:DesignHeight="450"
     x:Class="Company.Views.EmployeeListView">
    <DockPanel>
        <Button DockPanel.Dock="Bottom"
                HorizontalAlignment="Center">
            Add an employee
        </Button>
        <StackPanel
          <StackPanel Orientation="Horizontal" Margin="10">
            <TextBlock Width="100" Text="Jeremy"></TextBlock>
            <TextBlock Width="100" Text="Clarkson"></TextBlock>
            <TextBlock Width="100" Text="100"></TextBlock>
          </StackPanel>
          <StackPanel Orientation="Horizontal" Margin="10">
            <TextBlock Width="100" Text="James"></TextBlock>
            <TextBlock Width="100" Text="May"></TextBlock>
            <TextBlock Width="100" Text="100"></TextBlock>
          </StackPanel>
        </StackPanel>
    </DockPanel>
</UserControl>
```

w rederze jest podgląd



wyświetlenie w oknie głównym




```
<Window xmlns="https://github.com/avaloniaui"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="using:Company.ViewModels"
    xmlns:views="clr-namespace:Company.Views"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
    x:Class="Company.Views.MainWindow"
    x:DataType="vm:MainWindowViewModel"
    Icon="/Assets/avalonia-logo.ico"
    Title="Company">

    <Design.DataContext>
        <vm:MainWindowViewModel/>
    </Design.DataContext>
    <views:EmployeeListView></views:EmployeeListView>
</Window>

```

- utworzenie modelu
- utworzenie bazy testowej (mockup)
- utworzenie ViewModelu
  - nasz widok jest używany przez MainWindow, więc nasz ViewModel powinien być częścią MainWindowViewModel

```
namespace Company.Models;

public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Salary { get; set; }
}
```

- utworzenie modelu
- utworzenie bazy testowej (mockup)
- utworzenie ViewModelu
  - nasz widok jest używany przez MainWindow, więc nasz ViewModel powinien być częścią MainWindowViewModel

```
using Company.Models;
using System.Collections.Generic;
namespace Company.Services;
public class Database
{
  public IEnumerable<Employee> GetItems() => new[]
  {
    new Employee { FirstName = "Jeremy", LastName = "Clarkson"},
    new Employee { FirstName = "James", LastName = "May", Salary = 100},
    new Employee { FirstName = "Richard", LastName = "Hammond" },
  };
}
```

- utworzenie modelu
- utworzenie bazy testowej (mockup)
- utworzenie ViewModelu
  - nasz widok jest używany przez MainWindow, więc nasz ViewModel powinien być częścią MainWindowViewModel

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Company.Models;
namespace Company.ViewModels;
public class EmployeeListViewModel
{
  public EmployeeListViewModel(IEnumerable<Employee> items)
  {
    Items = new ObservableCollection<Employee>(items);
  }
  public ObservableCollection<Employee> Items { get; }
}
```

- utworzenie modelu
- utworzenie bazy testowej (mockup)
- utworzenie ViewModelu
  - nasz widok jest używany przez MainWindow, więc nasz ViewModel powinien być częścią MainWindowViewModel

```
using Company.Services;
namespace Company.ViewModels;
public class MainWindowViewModel : ViewModelBase
{
    public MainWindowViewModel(Database db)
    {
        List = new EmployeeListViewModel(db.GetItems());
    }
    public EmployeeListViewModel List { get; }
}
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
  - który został ustawiony w App.axaml.cs ...
  - który wskazuje na EmployeeListViewModel ...
  - który pochodzi od BaseViewModel
  - więc zwraca wartość true w funkcji dopasowania w ViewLocator
  - i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
  - EmployeeListViewModel ma właściwość Items, więc
  - EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
<Window xmlns="https://github.com/avaloniaui"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="using:Company.ViewModels"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

    ↪  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006
    mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"
    x:Class="Company.Views.MainWindow"
    x:DataType="vm:MainWindowViewModel"
    Icon="/Assets/avalonia-logo.ico"
    Title="Company"
    Content="{Binding List}">
</Window>
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
// App.axaml.cs
public override void OnFrameworkInitializationCompleted()
{
  if (ApplicationLifetime is
  ↪  IClassicDesktopStyleApplicationLifetime desktop)
  {
      var db = new Database();
      desktop.MainWindow = new MainWindow
      {
          DataContext = new MainWindowViewModel(db),
      };
  }
  base.OnFrameworkInitializationCompleted();
}
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
1  using Company.Services;
2  namespace Company.ViewModels;
3  public class MainWindowViewModel : ViewModelBase
4  {
5    public MainWindowViewModel(Database db)
6    {
7      List=new EmployeeListViewModel(db.GetItems());
8    }
9    public EmployeeListViewModel List { get; }
10 }
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
1  using System.Collections.Generic;
2  using System.Collections.ObjectModel;
3  using Company.Models;
4  namespace Company.ViewModels;
5
6  public class EmployeeListViewModel: ViewModelBase
7  {
8    public EmployeeListViewModel(IEnumerable<Employee> items)
9    {
10     Items = new ObservableCollection<Employee>(items);
11   }
12   public ObservableCollection<Employee> Items { get; }
13 }
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
1  using System;
2  using Avalonia.Controls;
3  using Avalonia.Controls.Templates;
4  using Company.ViewModels;
5  namespace Company;
6  public class ViewLocator : IDataTemplate {
7    public IControl Build(object data) {
8      var name=data.GetType().FullName!.Replace("ViewModel","View");
9      var type = Type.GetType(name);
10     if (type != null) {
11         return (Control)Activator.CreateInstance(type)!;
12     }
13     return new TextBlock { Text = "Not Found: " + name };
14   }
15   public bool Match(object data) {
16     return data is ViewModelBase;
17   }
18 }
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
1  using System;
2  using Avalonia.Controls;
3  using Avalonia.Controls.Templates;
4  using Company.ViewModels;
5  namespace Company;
6  public class ViewLocator : IDataTemplate {
7    public IControl Build(object data) {
8      var name=data.GetType().FullName!.Replace("ViewModel","View");
9      var type = Type.GetType(name);
10     if (type != null) {
11         return (Control)Activator.CreateInstance(type)!;
12     }
13     return new TextBlock { Text = "Not Found: " + name };
14   }
15   public bool Match(object data) {
16     return data is ViewModelBase;
17   }
18 }
```

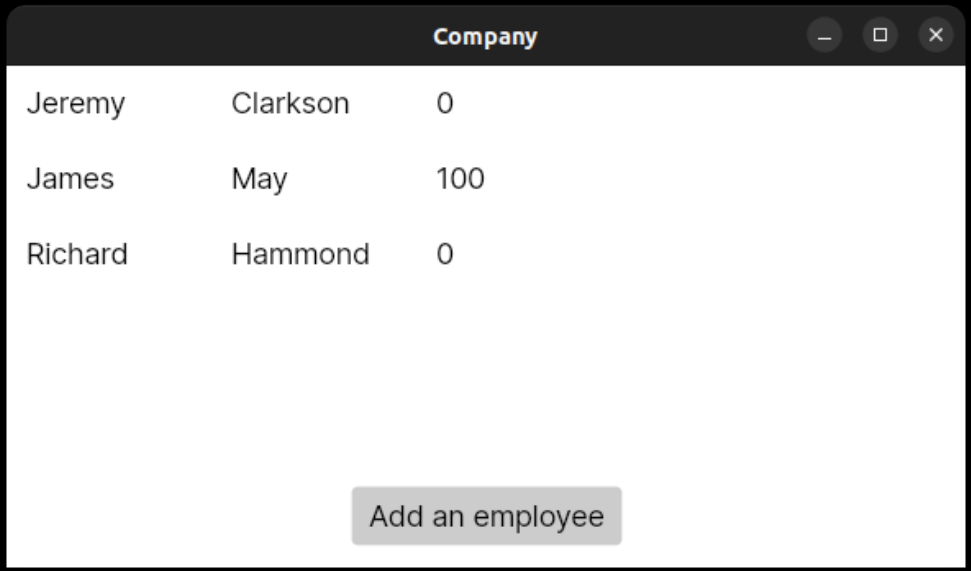
- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
1  using System.Collections.Generic;
2  using System.Collections.ObjectModel;
3  using Company.Models;
4  namespace Company.ViewModels;
5
6  public class EmployeeListViewModel: ViewModelBase
7  {
8    public EmployeeListViewModel(IEnumerable<Employee> items)
9    {
10     Items = new ObservableCollection<Employee>(items);
11   }
12   public ObservableCollection<Employee> Items { get; }
13 }
```

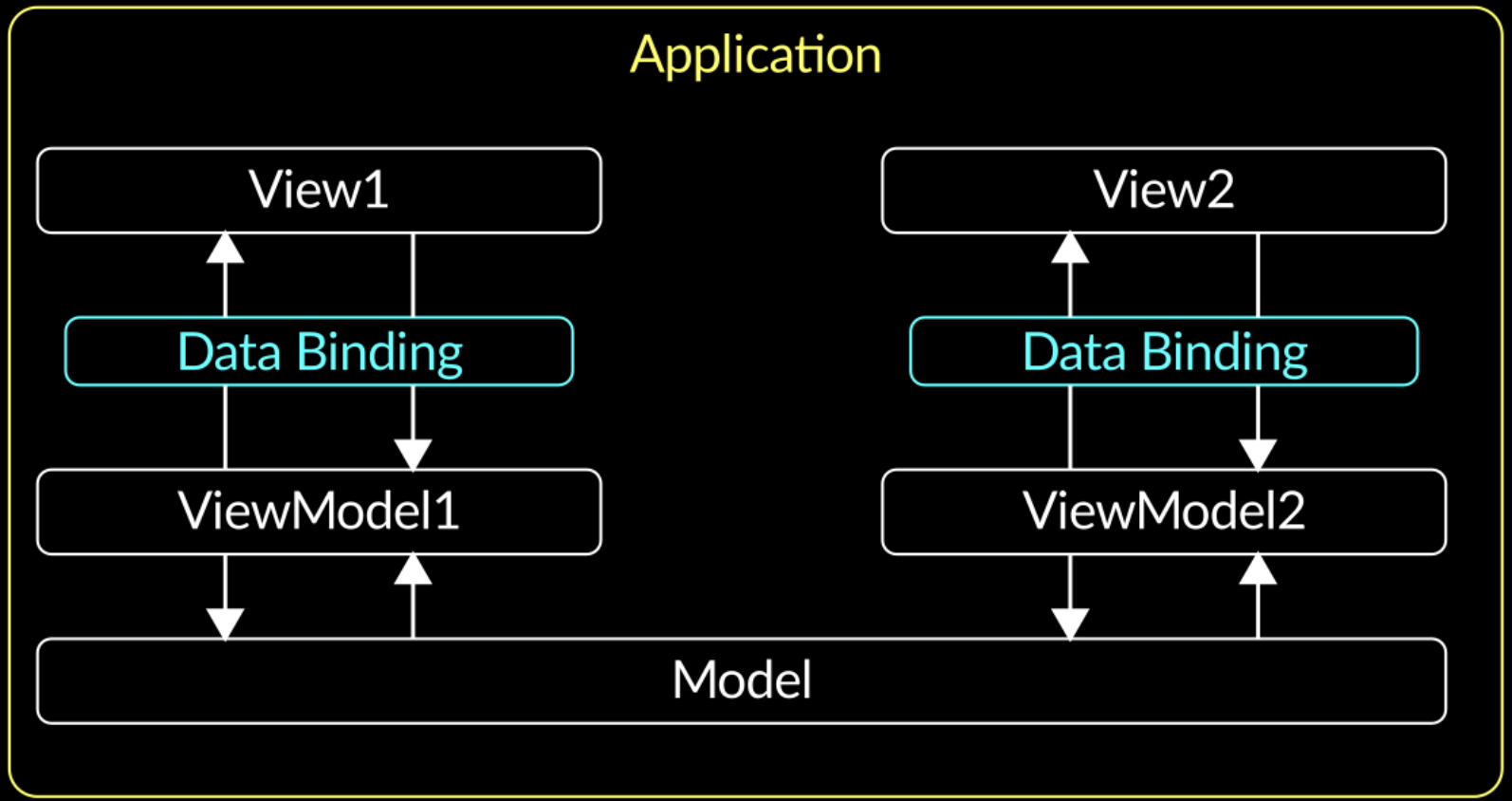
- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych

```
<UserControl xmlns="https://github.com/avaloniaui"
             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-compatibi
            mc:Ignorable="d" d:DesignWidth="400"
             d:DesignHeight="450"
            x:Class="Company.Views.EmployeeListView">
  <DockPanel>
    <Button DockPanel.Dock="Bottom" HorizontalAlignment="Center"
     Margin="10">
          Add an employee
    </Button>
    <ItemsControl Items="{Binding Items}">
      <ItemsControl.ItemTemplate>
        <DataTemplate>
          <StackPanel Orientation="Horizontal" Margin="10">
            <TextBlock Width="100" Text="{Binding
             FirstName}"></TextBlock>
            <TextBlock Width="100" Text="{Binding
             LastName}"></TextBlock>
            <TextBlock Width="100" Text="{Binding
             Salary}"></TextBlock>
          </StackPanel>
        </DataTemplate>
      </ItemsControl.ItemTemplate>
    </ItemsControl>
  </DockPanel>
</UserControl>
```

- Wiążemy właściwość List z `MainWindowViewModel` ...
- który został ustawiony w App.axaml.cs ...
- który wskazuje na EmployeeListViewModel ...
- który pochodzi od BaseViewModel
- więc zwraca wartość true w funkcji dopasowania w ViewLocator
- i w rezultacie EmployeeListView jest zwracany z `DataContext` ustawionym na EmployeeListViewModel
- EmployeeListViewModel ma właściwość Items, więc
- EmployeeListView używa `DataContext` do uzyskania wyświetlanych danych



- Model-View-Presenter (Martin Fowler)
- Model-View-ViewModel (John Gossma)



```
1  namespace Wpf2015Z {
2    public class Contact: INotifyPropertyChanged {
3      private string lastName;
4      public string LastName {
5        get { return lastName; }
6        set {
7          lastName = value;
8          OnPropertyChanged("LastName");
9        }
10     }
11     public event PropertyChangedEventHandler PropertyChanged;
12     // Wysyla event do WPF poprzez data-binding o zmianie
13     private void OnPropertyChanged(string propertyName) {
14       var handler = PropertyChanged;
15       if (handler != null) {
16         handler(this, new PropertyChangedEventArgs(propertyName));
17       }
18     }
19   }
20 }
```

plik ContactManagerViewModel.cs (ModelView)

```
1 namespace Wpf2015Z {
2   public class ContactManagerViewModel:ObservableCollection<Contact> {
3     public ContactManagerViewModel() {
4       Add(new Contact { LastName = "Kowalski"});
5     }
6   }
7 }
```

- Aby widzieć właściwości, należy pokazać namespace c# (u nas vm)
- Wiązanie robimy za pomocą {Binding ..
- Jeżeli właściwość nie jest bezpośrednio, to {Binding Path=..

```
<Window x:Class="Wpf2015Z.MainWindow"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:vm="clr-namespace:Wpf2015Z"
   Title="MainWindow" Height="350" Width="525">
  <StackPanel>
    <TextBox  Text="{Binding LastName}" />
    <Button Content="Button"  Click="Button_Click" />
  </StackPanel>
</Window>
```

- przykład nie jest mvvm, bo przycisk dla prostoty nie jest zrobiony komendą

- Przypisanie kontekstu do obiektu ModelView
- W kodzie obsługi nic nie wiemy, kto i jak zmienia właściwość LastName

```
namespace Wpf2015Z {
  public partial class MainWindow : Window {
    public MainWindow() {
      DataContext = new Contact();
      InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e) {
      MessageBox.Show((DataContext as Contact).LastName,
        "Potwierdz:", MessageBoxButton.YesNo, MessageBoxImage.Question);
    }
  }
}
```

- Czyli nic nie stoi na przeszkodzie, aby pisać ładne testy

- `#region` służy do sterowania Visual Studio - można zwinąć część pliku
- kawałek klasy służy tylko w czasie uruchamiania
- najważniejsza jest implementacja metody `OnPropertyChanged`
- każdy kto podłącza się do właściwości tej klasy za pomocą wiązania wie, że wystawia ona zdarzenie `PropertyChanged`

```
1  namespace Calc.ViewModel {
2  public abstract class ViewModelBase
3      : INotifyPropertyChanged, IDisposable {
4  #region Constructor
5  protected ViewModelBase() { }
6  #endregion // Constructor
7
8  #region Debugging Aides
9  /// <summary>
10 /// metoda nie wystepuje w wersji reklease projektu
11 /// </summary>
12 [Conditional("DEBUG")]
13 [DebuggerStepThrough]
14 public void VerifyPropertyName(string propertyName) {
15   // Verify that the property name matches a real,
16   // public, instance property on this object.
17   if (TypeDescriptor.GetProperties(this)[propertyName] == null) {
18     string msg = "Invalid property name: " + propertyName;
19
20     if (this.ThrowOnInvalidPropertyName)
21       throw new Exception(msg);
22     else
23       Debug.Fail(msg);
24   }
25 }
26 protected virtual bool ThrowOnInvalidPropertyName{ get; private set;}
27 #endregion // Debugging Aides
28
29 #region INotifyPropertyChanged Members
```

```
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged(string propertyName) {
  this.VerifyPropertyName(propertyName);

  PropertyChangedEventHandler handler = this.PropertyChanged;
  if (handler != null) {
    var e = new PropertyChangedEventArgs(propertyName);
    handler(this, e);
  }
}

#endregion // INotifyPropertyChanged Members
#region IDisposable Members
public void Dispose() {
  this.OnDispose();
}
protected virtual void OnDispose() {
}
#endregion // IDisposable Members
}
}
```

- właściwości muszą informować, że się zmieniły
- podłączenie modelu też zewnętrzne - aby można było w czasie testowania przypiąć coś innego
- podłączenie do właściwości modelu też można zewnętrznie (u mnie sznurek)

```
namespace Calc.ViewModel {
  public class MainWindowViewModel : ViewModelBase {
    public ICommand ButtonCommand { get; set ;}
    public string Display {
      get {
        return m_calculator.Display;
      }
      set {
        if (value == m_calculator.Display)
          return;
        m_calculator.Display = value;
        base.OnPropertyChanged("Display");
      }
    }
    private Model.Calculator m_calculator;
    public  Model.Calculator model {
        set {
            m_calulator = value;
        }
    }
    public MainWindowViewModel() {
      ButtonCommand = new RelayCommand(
          new Action<object>(delegate(object obj) {
        m_calculator.Process(obj as string);
        Display = m_calculator.Display;
      }));
    }
  }
}
```

Przykład:

```
// Program.cs
namespace SampleConsole {
  class Program {
    static void Main(string[] args) {
      Worker worker = new Worker();
      worker.work("NTR");
      Console.ReadKey();
    }
  }
}
```

- Tworzymy obiekt klasy `Worker`
- W konstruktorze klasy `Worker` tworzone są obiekty pomocnicze

Dependency Injection (2)

```
// worker.cs
namespace SampleConsole {
  class Worker {
    Helper helper;
    public Worker(IHelper helper) {
      this.helper = new Helper();
    }
    public void work(string msg) {
      Console.WriteLine("starting hard work for {0}: ", msg);
      helper.process(msg);
    }
  }
}
```

```
// helper.cs
namespace SampleConsole {
  class Helper {
    public void process(string msg) {
      Console.WriteLine("I am too lazy");
    }
  }
}
```

Problemy

- Klasa `Worker` ściśle powiązana z klasą `Helper`
- W dużych aplikacjach powstaje stosowa struktura zależności, trudna do testowania
- Gdyby dało się oduczyć klasy `Worker` od implementacji klasy `Helper`
- Najlepiej wsadzić odpowiedni interfejs:

```
1 namespace SampleConsole {
2   interface IHelper {
3     void process(string msg);
4   }
5 }
```

Przykład:

```
// Program.cs
namespace SampleConsole {
  class Program {
    static void Main(string[] args) {
      Helper help = new Helper();
      Worker worker = new Worker(help);
      worker.work("NTR");
      Console.ReadKey();
    }
  }
}
```

- To my tworzymy obiekty pomocnicze
- Klasa worker dostaje od nas środowisko, zna tylko interfejs

Dependency Injection (5)


```
// worker.cs
namespace SampleConsole {
  class Worker {
    IHelper helper;
    public Worker(IHelper helper) {
      this.helper = helper;
    }
    public void work(string msg) {
      Console.WriteLine("starting hard work for {0}: ", msg);
      helper.process(msg);
    }
  }
}
```

```
// helper.cs
namespace SampleConsole {
  class Helper : IHelper{
    public void process(string msg) {
      Console.WriteLine("I am too lazy");
    }
  }
}
```

Przykład:

```
// Program.cs
namespace SampleConsole {
  class Program {
    static void Main(string[] args) {
      Helper help = new Helper();
      Worker worker = new Worker{ helper=help };
      worker.work("NTR");
      Console.ReadKey();
    }
  }
}
```

- To my tworzymy obiekty pomocnicze
- Klasa `Worker` dostaje od nas środowisko, zna tylko interfejs
- Wykorzystujemy konstruktor bezparametrowy klasy `Worker`
- Zależności wystawiane jako właściwości do wypełnienia - nie ma psucia konstruktorów
- Boli to, że tak naprawdę dopuszczamy zapis jednorazowy
- Zamiast zabawy ręcznej lepiej użyć **NInject**

- konfiguracja w kodzie

```
class Program {
  static void Main(string[] args) {
    var kernel = new StandardKernel();
    kernel.Bind<IWorker>().To<Worker>()
     .InSingletonScope()
     .WithConstructorArgument("helper", kernel.Get<Helper>());
    var worker = kernel.Get<IWorker>();
    worker.work("NInject NTR");
    Console.ReadKey();
  }
}
```

- mamy możliwość określenia sposobu tworzenia obiektu;

  Transient - domyślne, tworzymy kolejny

  Singleton - jedna instancja w programie

  Thread - jedna instancja w w każdym wątku

  Request - jedna instancja w każdym nawrocie aplikacji webowej

- Atrybut `[Inject]` określa który konstruktor użyć

```
// worker.cs
namespace SampleConsole {
  class Worker {
    IHelper helper;
    [Inject]
    public Worker(IHelper helper) {
      this.helper = helper;
    }
    public void work(string msg) {
      Console.WriteLine("starting hard work for {0}: ", msg);
      helper.process(msg);
    }
  }
}
```

- Atrybut `[Inject]` określa którą funkcję użyć

```
// worker.cs
namespace SampleConsole {
  class Worker {
    IHelper helper;
    public Worker() {
    }
    [Inject]
    public setHelper(IHelper helper) {
      this.helper = helper;
    }
    public void work(string msg) {
      Console.WriteLine("starting hard work for {0}: ", msg);
      helper.process(msg);
    }
  }
}
```

- Atrybut `[Inject]` określa którą właściwość użyć

```
// worker.cs
namespace SampleConsole {
  class Worker {
    [Inject]
    IHelper helper {get ; set; }
    public Worker() {
    }
    public void work(string msg) {
      Console.WriteLine("starting hard work for {0}: ", msg);
      helper.process(msg);
    }
  }
}
```

- Możemy użyć pliku konfiguracyjnego

```
kernel.Load("registrations.xml");
//--------
<?xml version="1.0" encoding="utf-8" ?>
<module name="ntrModule">
  <bind service="NinjectConsole.IHelper, helper"
    to="NinjectConsole.Helper, helper" />
  <bind service="NinjectConsole.IWorker, Workers"
    to="NinjectConsole.Worker, Workers" />
</module>
```

- Ale jego możliwości są ograniczone

Położenie bazy danych podajemy w postaci **connection-stringa**. Najlepszym miejscem jest plik konfiguracyjny aplikacji (czyli **App.config**)

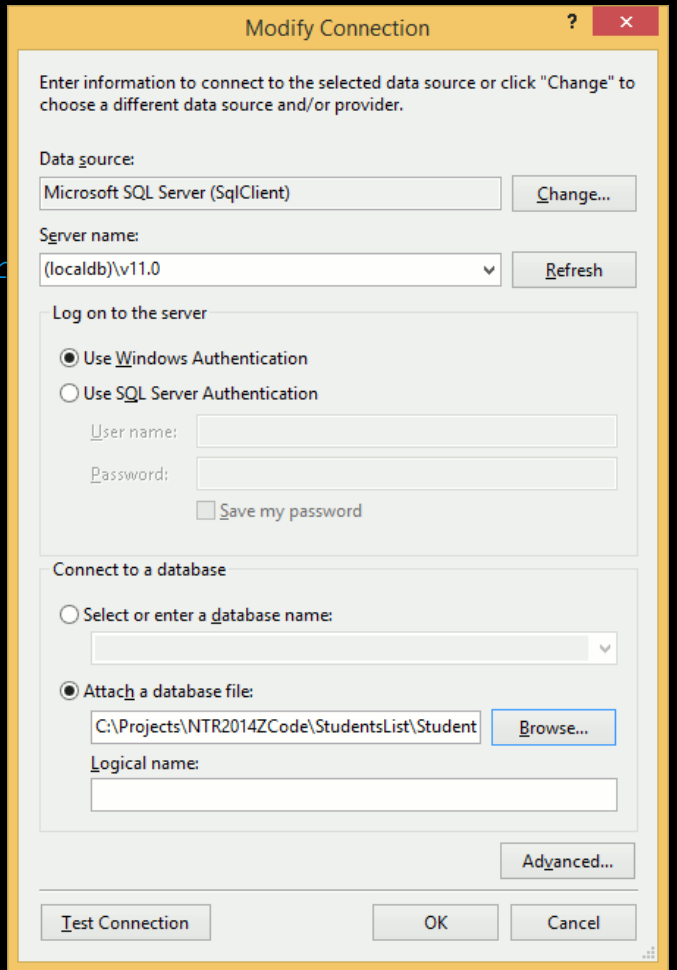
```
<connectionStrings>
  <add name="StudentsList.Model.StorageContext"
    connectionString="Data Source=(localdb)/v11.0;
    AttachDbFilename=C:/Projects/NTR2014ZCode/StudentsList
                    /StudentsList.Model.StorageContext.mdf;
    Initial Catalog=Model.StorageContext;Integrated Security=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

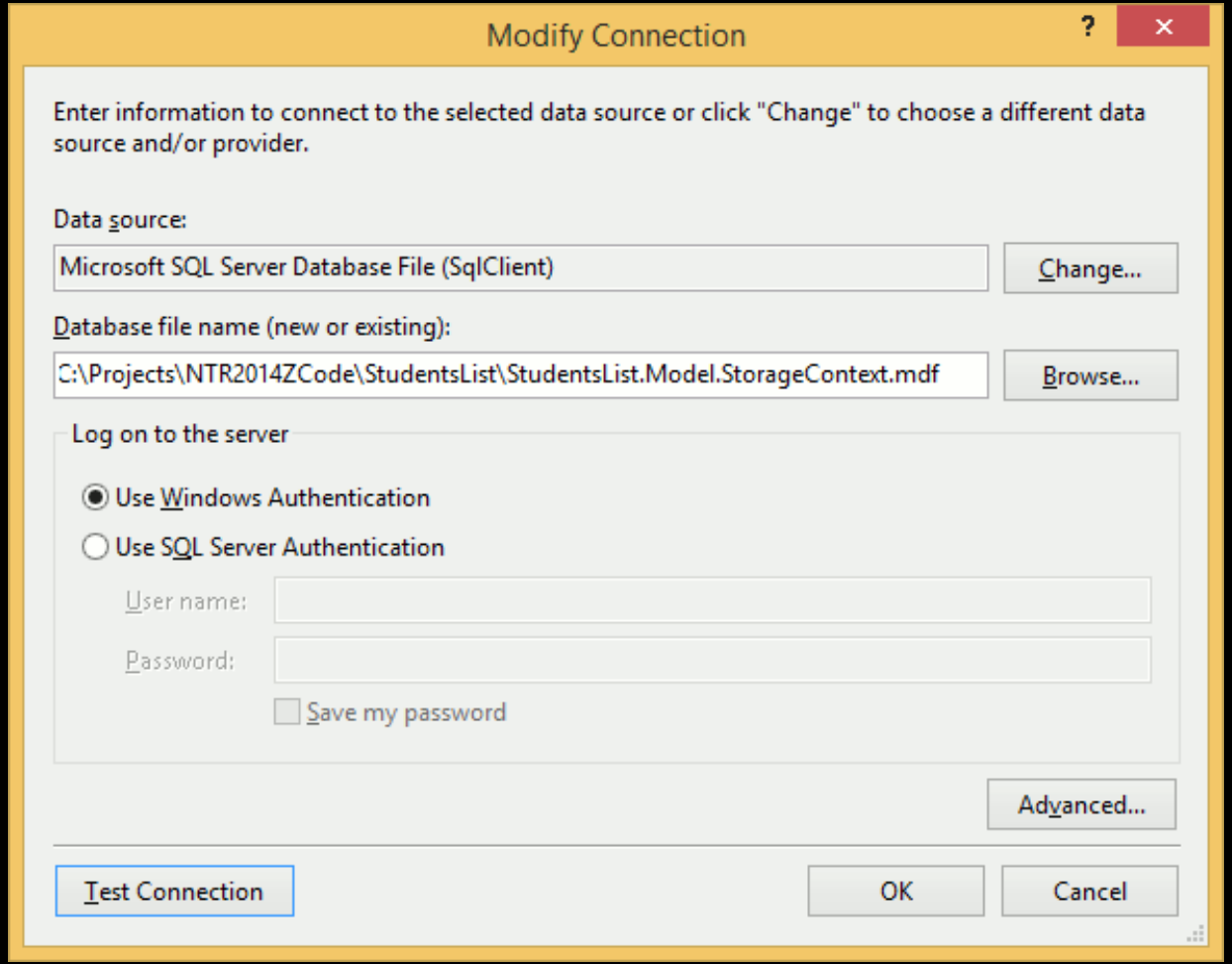
- Plik jest tekstowy - administrator może w dowolnym miejscu zmienić
- Do pewnego stopnia nie ma znaczenia z czym się łączymy

Możemy łączyć się z dowolną bazą ...
Ale najłatwiej z SQL-Serwerem i jego rodzinką:

- **SQL-Server** (6.0,7.0,2000,2005,2008,2008 R2, 2012, 2014)
- **SQL-Express**
- **LocalDb**

Server name - `(localdb)/v11`
Server name - `(localdb)/MSSQLLocalDB`
Attach database file - `ścieżka do pliku mdf`
automatyczne lądują w:
`C:/Users/jmy/AppData/Local/Microsoft/Micr`
`SQL Server Local DB/Instances`




Modify Connection
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
Microsoft SQL Server Database File (SqlClient)
Change...
Database file name (new or existing):
C:\Projects\NTR2014ZCode\StudentsList\StudentsList.Model.StorageContext.mdf
Browse...
Log on to the server
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Save my password
Advanced...
Test Connection
OK
Cancel

Server name `./SqlExpress`

automatyczne lądują w:
`C:/Users/jmy/AppData/Local/Microsoft/Micr`
`SQL Server Local DB/Instances`