MVVM

AARHUS UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

MICHAEL LOFT
ML@ASE.AU.DK

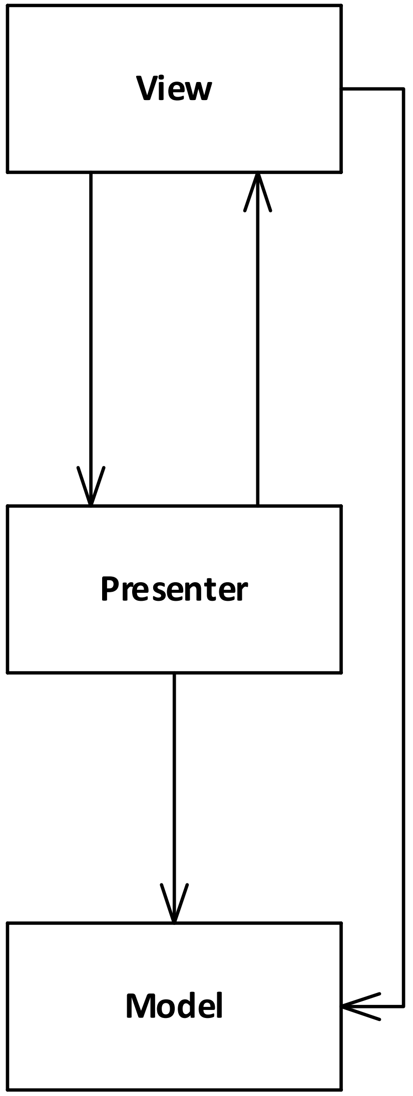
Agenda

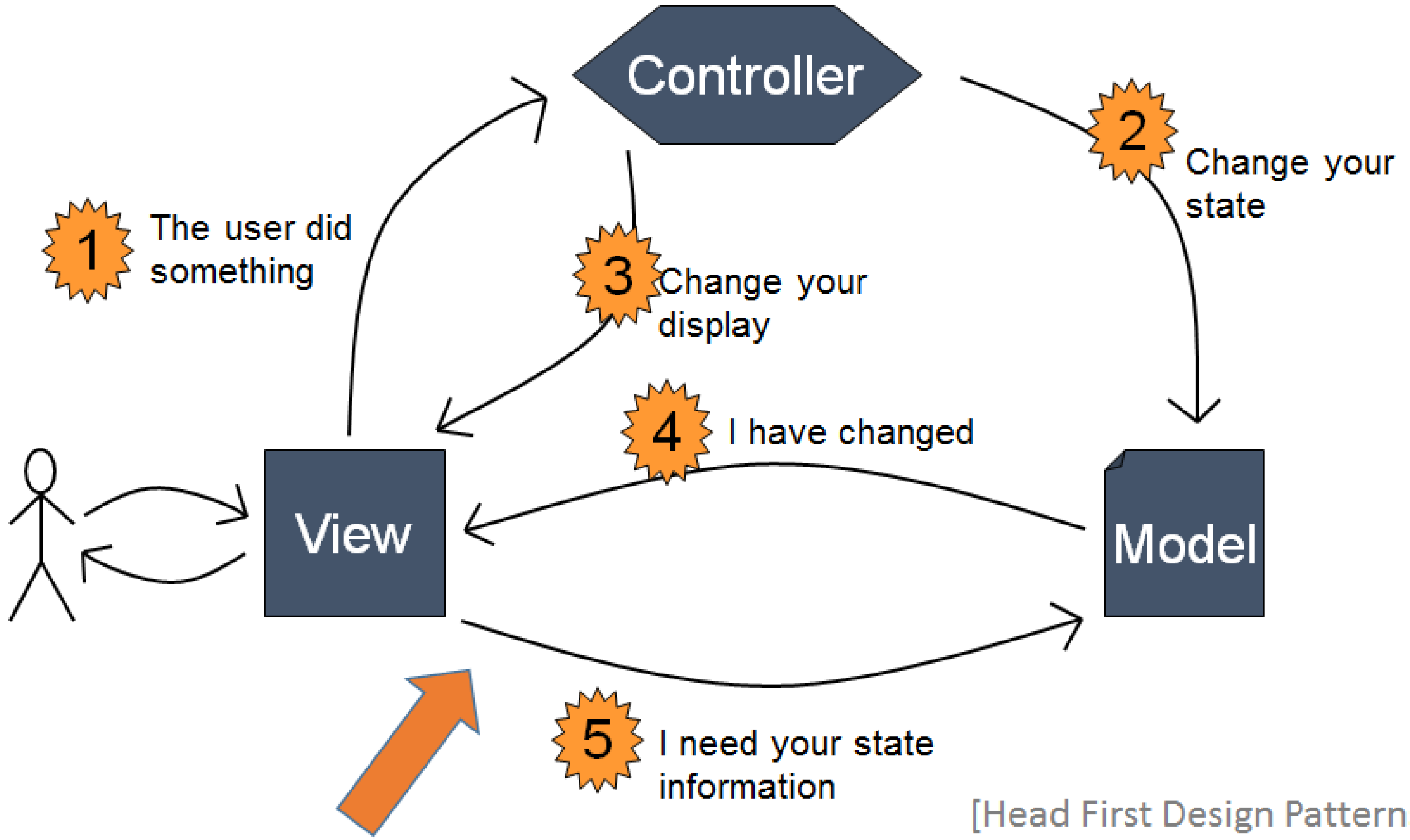
MVP Recap

Presentation Model

MVVM

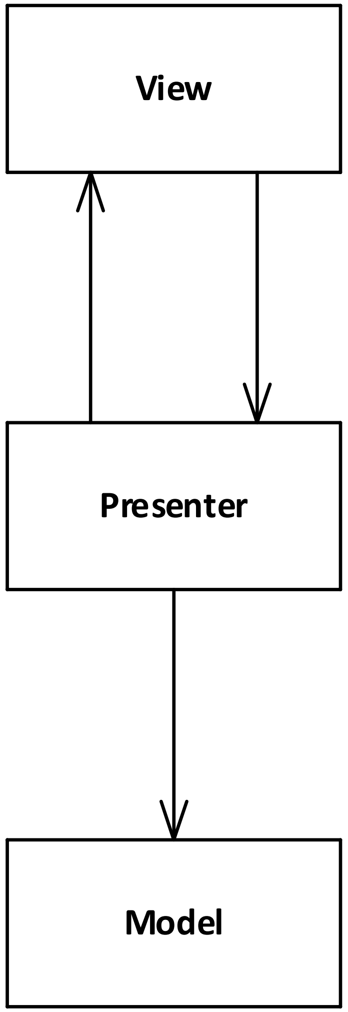
MVP Recap – Supervising Controller

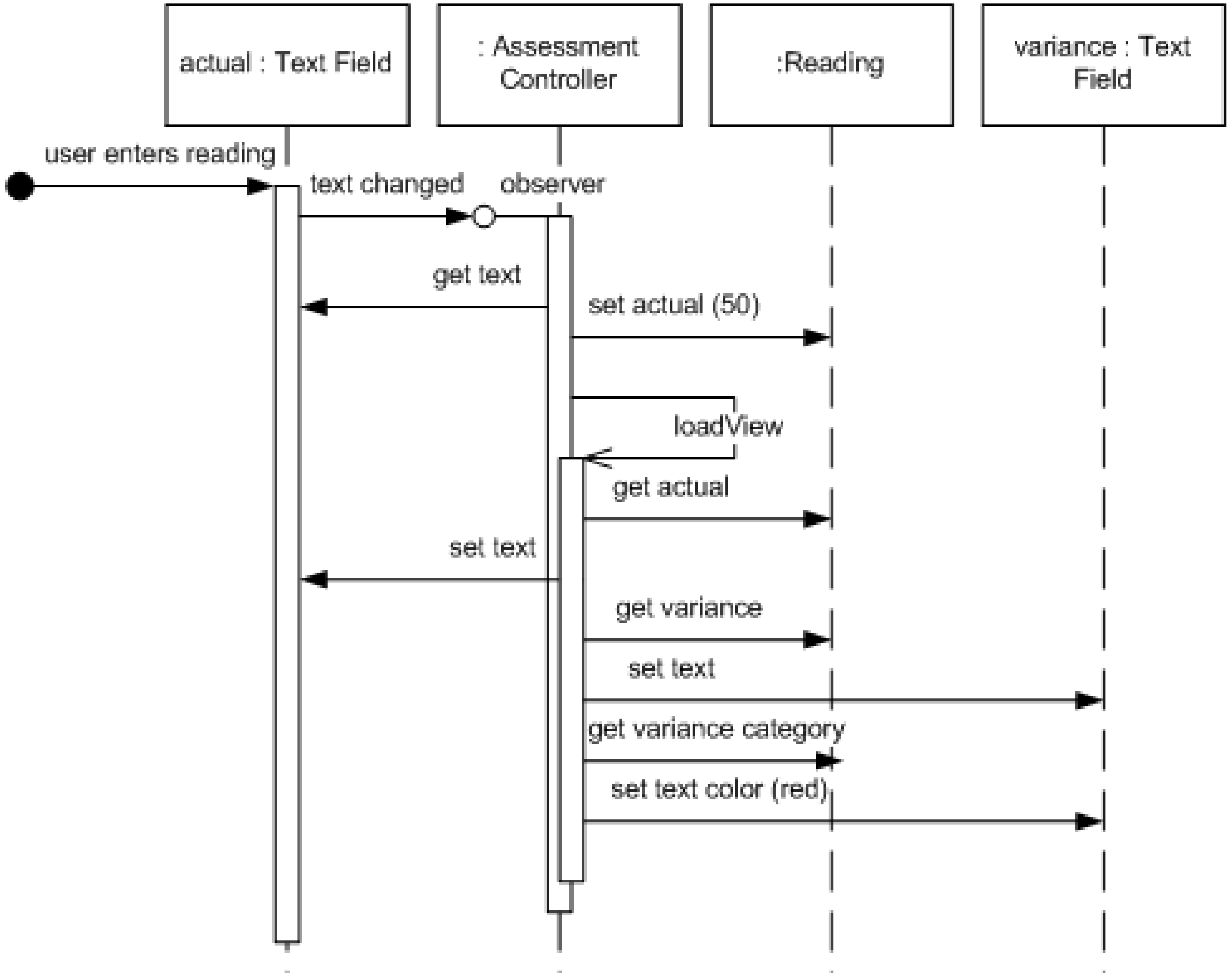




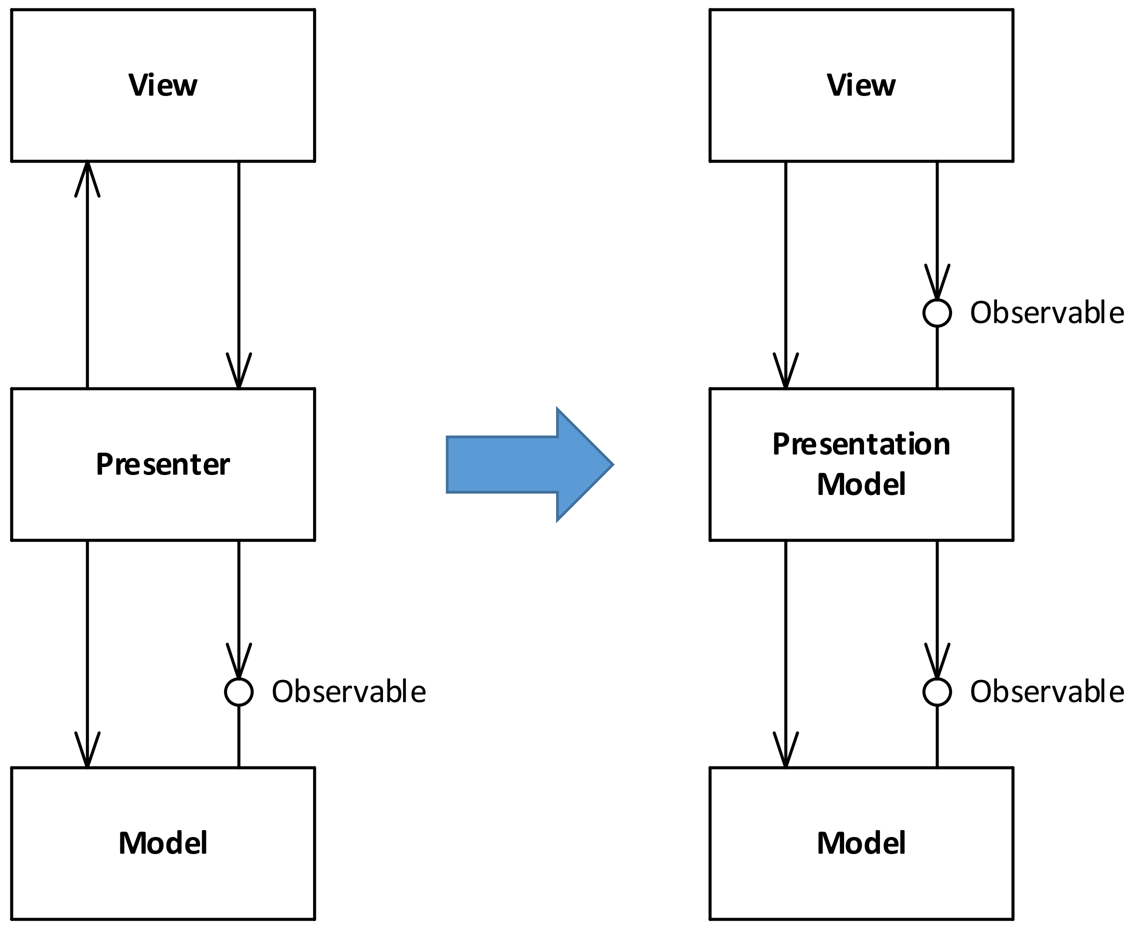
[Head First Design Patterns]

MVP Recap – Passive View





Presentation Model



The View becomes responsible for updating itself.

The Presentation Model becomes a model of the data needed for presentation.

It still recieves user input from the View and Modifies the Model.

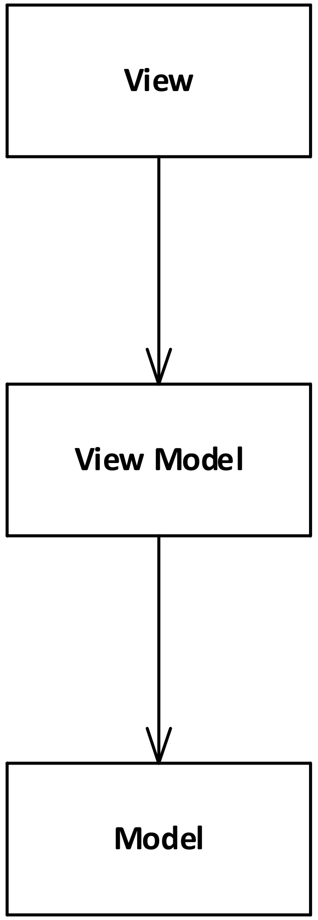
Model – View- ViewModel

Is a Microsoft specialization of the Presentation Model Pattern (MVP) used in WPF.

MVVM was designed to make use of specific functions in WPF (mostly data binding) to better facilitate the separation of View layer development by removing virtually all "code behind" from the View layer

Instead of requiring designers to write View code, they can use XAML and create bindings to the ViewModel, which is written and maintained by application developers

MVVM



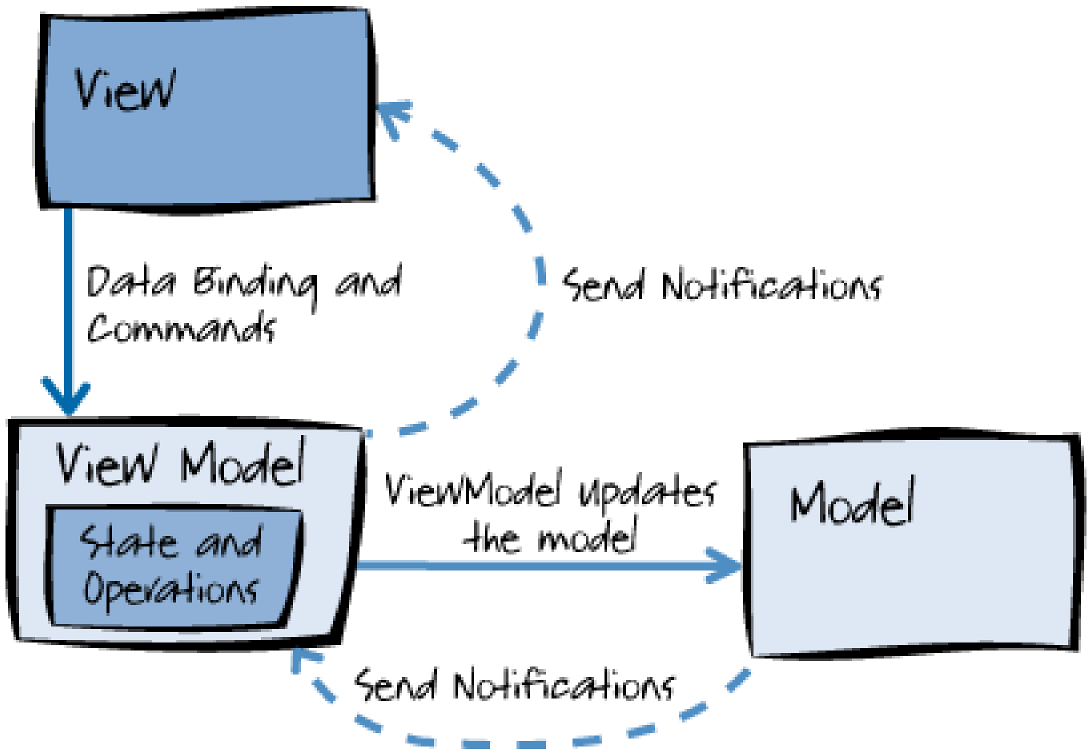
The View is connected to the ViewModel through data binding and sends commands to the View-Model.

The ViewModel is unaware of the View.

The ViewModel may interact with Model through properties, method calls and may receive events from the model.

The Model is unaware of the View-Model.

MVVM



The View is connected to the ViewModel through data binding and sends commands to the View-Model.

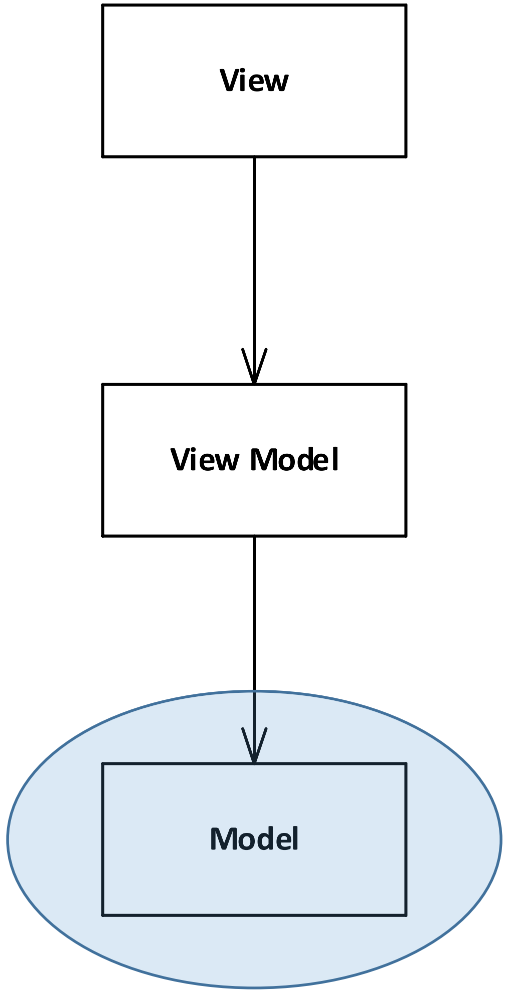
The ViewModel is unaware of the View.

The ViewModel may interact with Model through properties, method calls and may receive events from the model.

The Model is unaware of the View-Model.

image: https://msdn.microsoft.com/en-us/library/hh848246.aspx

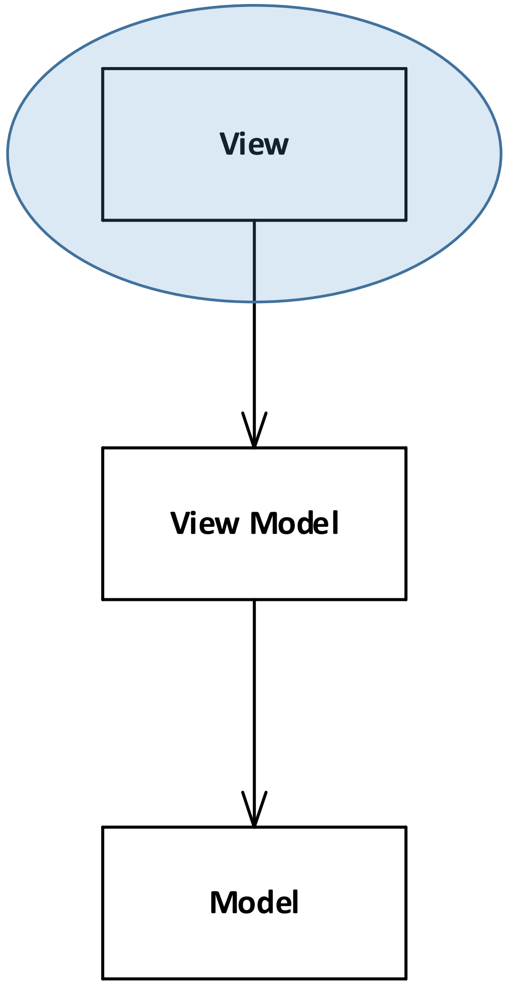
MVVM - Model



As in MVC/MVP, the model refers to either:

- an object model that represents the real state content (an object-oriented approach)
- the data access layer that represents that content (a data-centric approach)

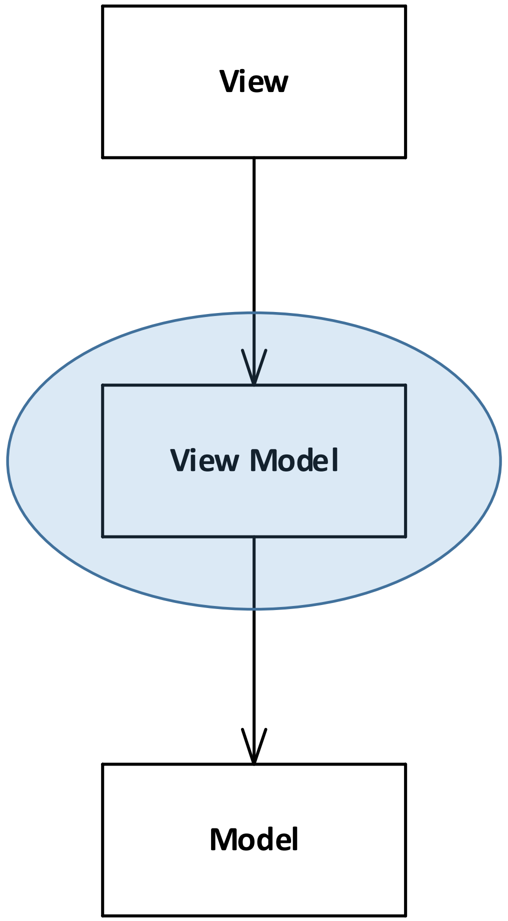
MVVM - View



All elements displayed by the GUI such as windows, buttons, graphics, and other controls

A View may represent the whole window - or it may only represent a part of a window.

MVVM - ViewModel

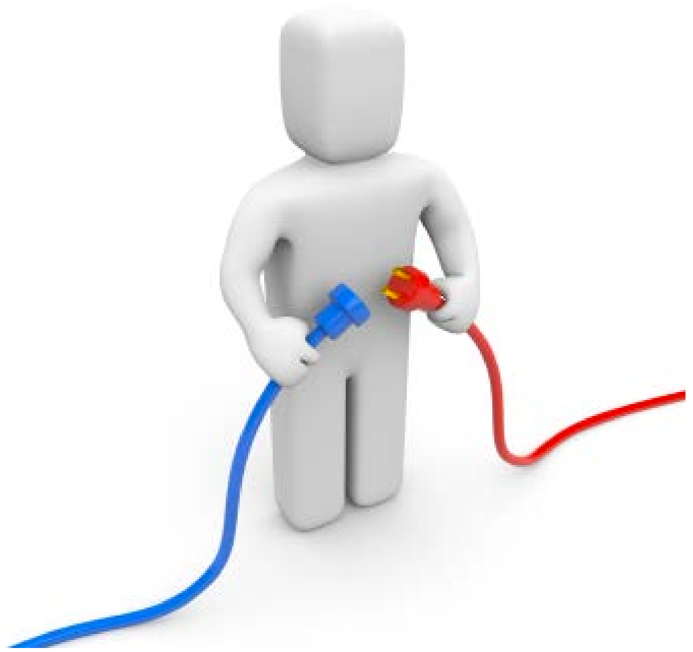


A "Model of the View"

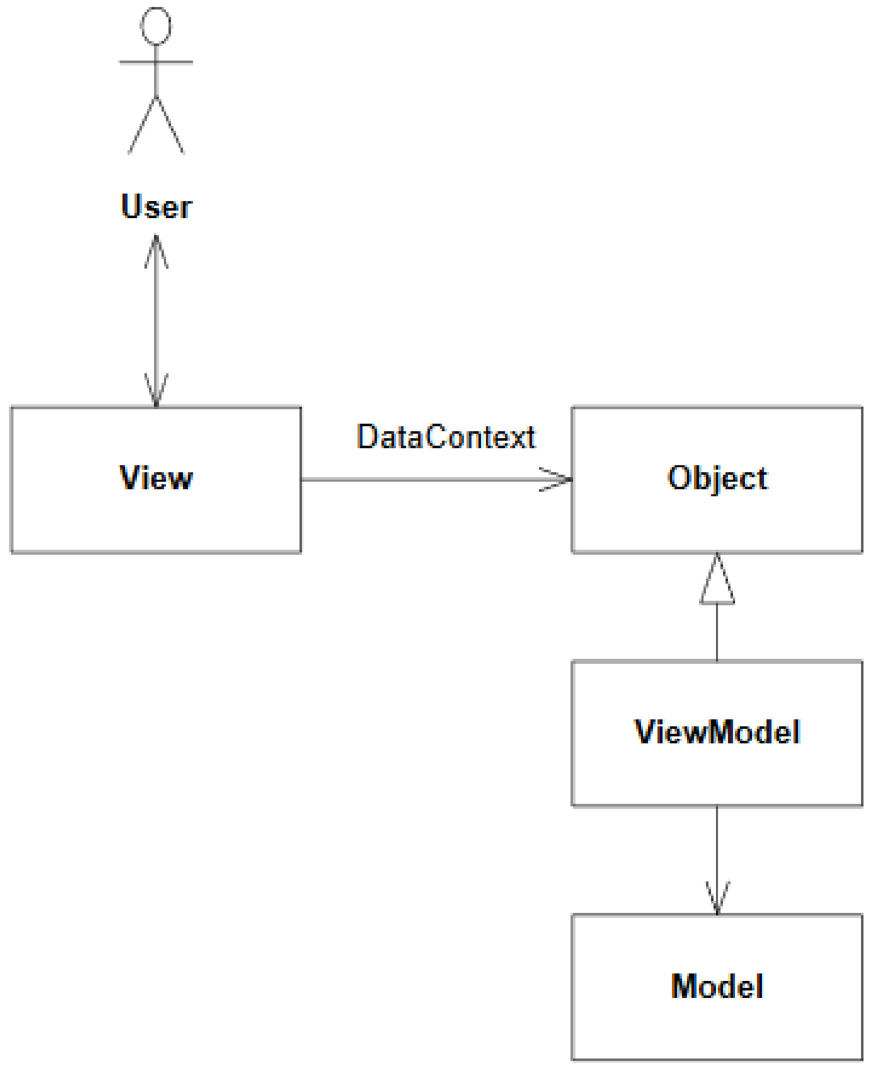
Meaning it is an abstraction of the View that also serves in data binding between the View and the Model .

It could be seen as a specialized aspect of a Presenter (in the MVP pattern) that acts as a data binder/converter that changes Model information into View information and passes commands from the View into the Model

Connecting Views and ViewModels



View – ViewModel relation



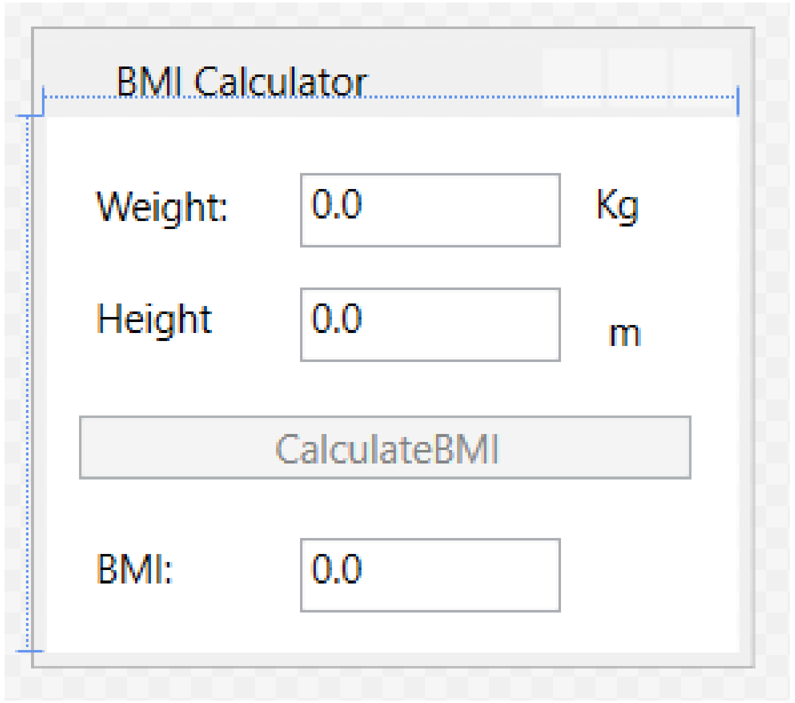
The communication between a View and its ViewModel is mainly done by use of Data Bindings where the View's DataContext property holds the reference to the ViewModel.

The relation from View to ViewModel is speficied in XAML and may be obtained from a Locator or some kind of dependency injection.

Binding the View to the ViewModel

```
<Window x:Class="BMICalculator.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentat
ion"

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:BMICalculator"
        xmlns:viewModel="clr-
namespace:BMICalculator.ViewModel"
        mc:Ignorable="d"
        Title="BMI Calculator" Height="195.098"
Width="220.098">
    <Window.DataContext>
        <viewModel:BMIViewModel/>
    </Window.DataContext>
```

The DataContext of the View is set to the ViewModel to which the View shall bind.



Binding the View to the ViewModel

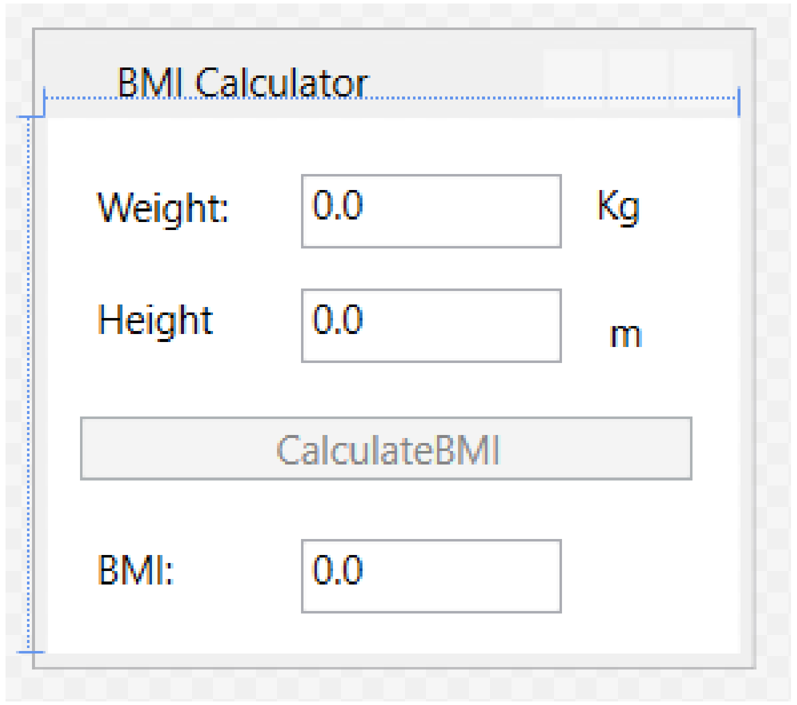
```
class BMIViewModel : INotifyPropertyChanged
{
    private double _bmi;

    public double Height
    {
        get => bmiModel.Height;
        set
        {
            if (value != bmiModel.Height)
            {
                bmiModel.Height = value;
                OnPropertyChanged();
            }
        }
    }

    ....
```

The ViewModel exposes properties, that the View can bind to.

It also implements the INotifyPropertyChanged interface.

Binding the View to the ViewModel

```
<TextBox
    HorizontalAlignment="Left"
    Height="23"
    Margin="78,17,0,0"
    TextWrapping="Wrap"
    Text="{Binding Path=Height, StringFormat=F1,
Mode=TwoWay}"
    VerticalAlignment="Top"
    Width="80"/>
```



Commands from View to ViewModel

- Binding the presentation layer directly to the properties of a view model works like a charm, but binding user input to the methods of the view model doesn't work at all
    - The built in WPF command routing can’t reach the ViewModel!
- To overcome this limitation of WPF, different developers has come up with similar solutions:
    - Josh Smith's **RelayCommand**
    - Prism's **DelegateCommand**
- Both take advantage of the fact that the command properties of some WPF controls allow the use of command types other than RoutedCommand
- The commands just need to implement the ICommand interface
- The suggested way to use the RelayCommands or DelegateCommands is to add command properties to the view model, map the commands to methods in the view model, and then bind the command properties of the view model to the command properties of the WPF controls

RelayCommand

```
public class RelayCommand<T> : ICommand
{
    public RelayCommand(Action<T> execute)
    { ...

    public RelayCommand(Action<T> execute,
                        Predicate<T> canExecute)
    { ...


public class RelayCommand : ICommand
{
    public RelayCommand(Action execute)
    { ...

    public RelayCommand(Action execute,
                        Func<bool> canExecute)
    { ...
```

RelayCommand wraps the method to be executed in a class, which implements the ICommand interface.

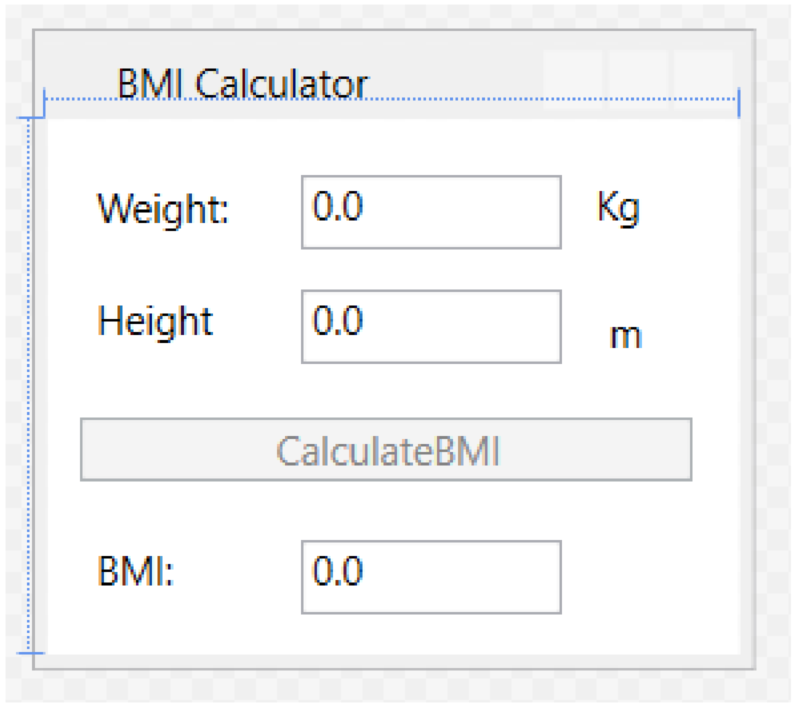
The RelayCommand can be bound to Command properties in the View.

Binding to Commands

```
private ICommand _calcBMICommand;
public ICommand CalcBMICommand
{
    get
    {
        return _calcBMICommand ?? (_calcBMICommand =
            new RelayCommand(CalcBMI, CalcBMICanExecute));
    }
}

private void CalcBMI()
{
    BMI = bmiModel.CalculateBMI();
    OnPropertyChanged("BMI");
}

private bool CalcBMICanExecute()
{
    bool paramsAreValid = (Weight != 0.0 && Height != 0.0);
    return paramsAreValid;
}
```

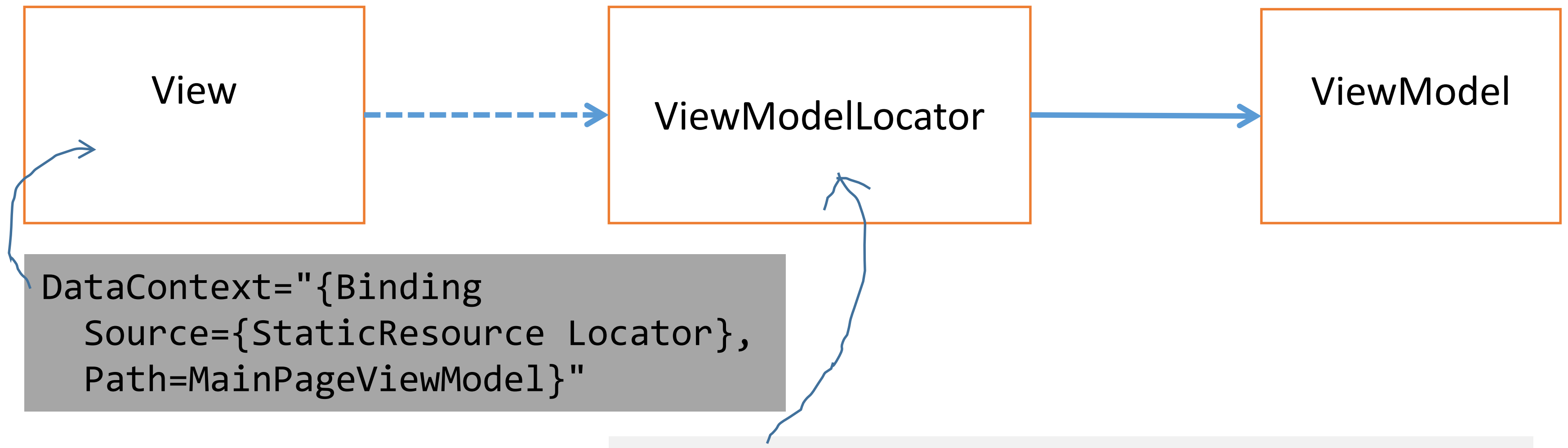
Binding to Commands

```
<Button Content="CalculateBMI"
  HorizontalAlignment="Left"
  Margin="10,91,0,0"
  VerticalAlignment="Top"
  Width="188"
  Command="{Binding CalcBMICommand, Mode=OneTime}"/>
```



ViewModel Locator

ViewModel Locator Structure



```
DataContext="{Binding
   Source={StaticResource Locator},
   Path=MainPageViewModel}"
```

```
public MainPageViewModel MainPageViewModel
{
   get { return new MainPageViewModel(); }
}
```

- There are several variations of the ViewModelLocator pattern E.g.:
  - You can instantiate it as a resource in App.xaml
  - You can make it static

View First: Using a ViewModel Locator

```
public class ViewModelLocator {
   public BMIViewModel BmiViewModel
   {  get{return new BMIViewModel(new BMIModel());}}
}
```

App.xaml:

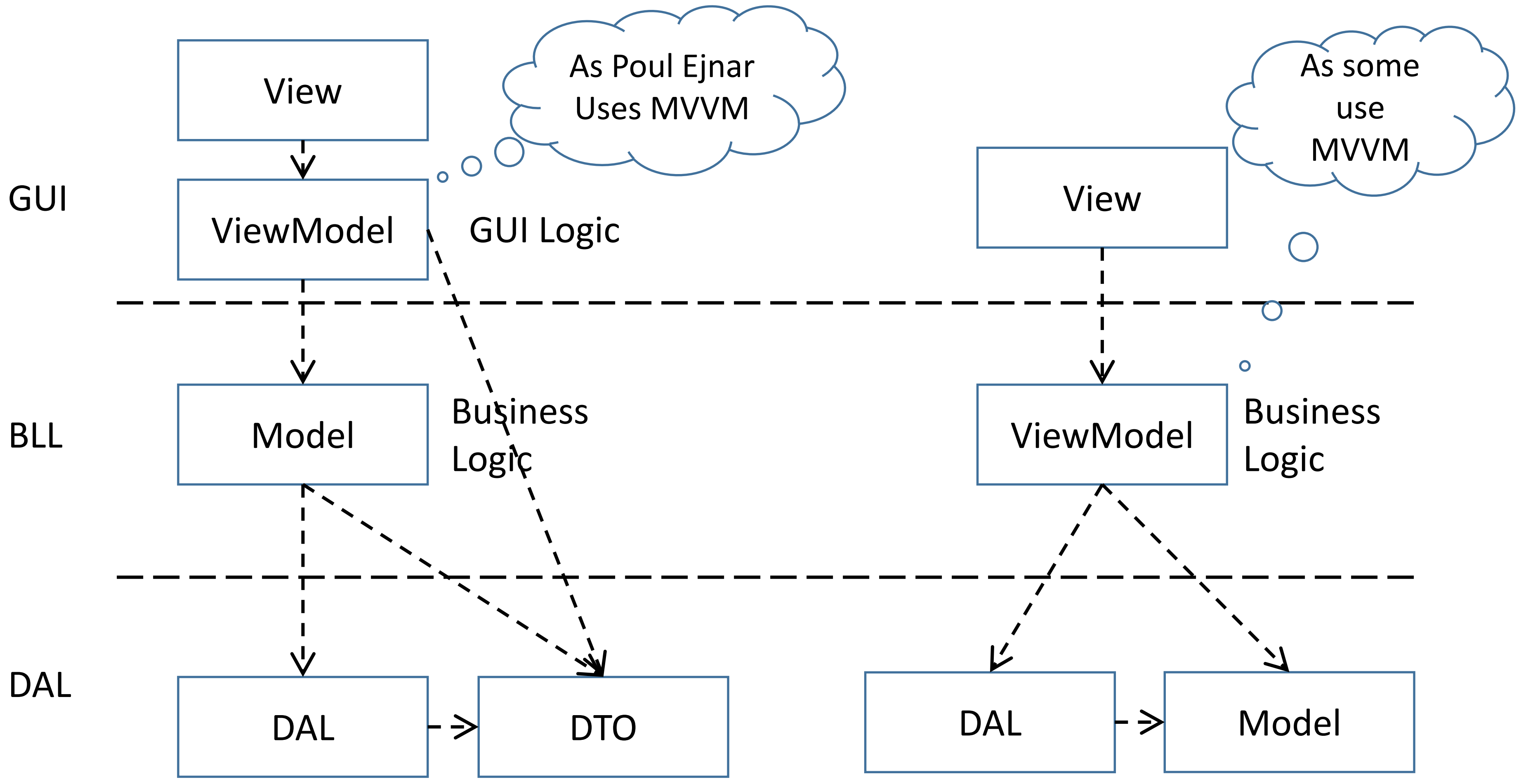
```
<Application x:Class="UsingAViewModelLocator.App"
    xmlns:local="clr-namespace:UsingAViewModelLocator"
    StartupUri="MainWindow.xaml">
  <Application.Resources>
    <local:ViewModelLocator x:Key="ViewModelLocator" />
  </Application.Resources>
</Application>
```

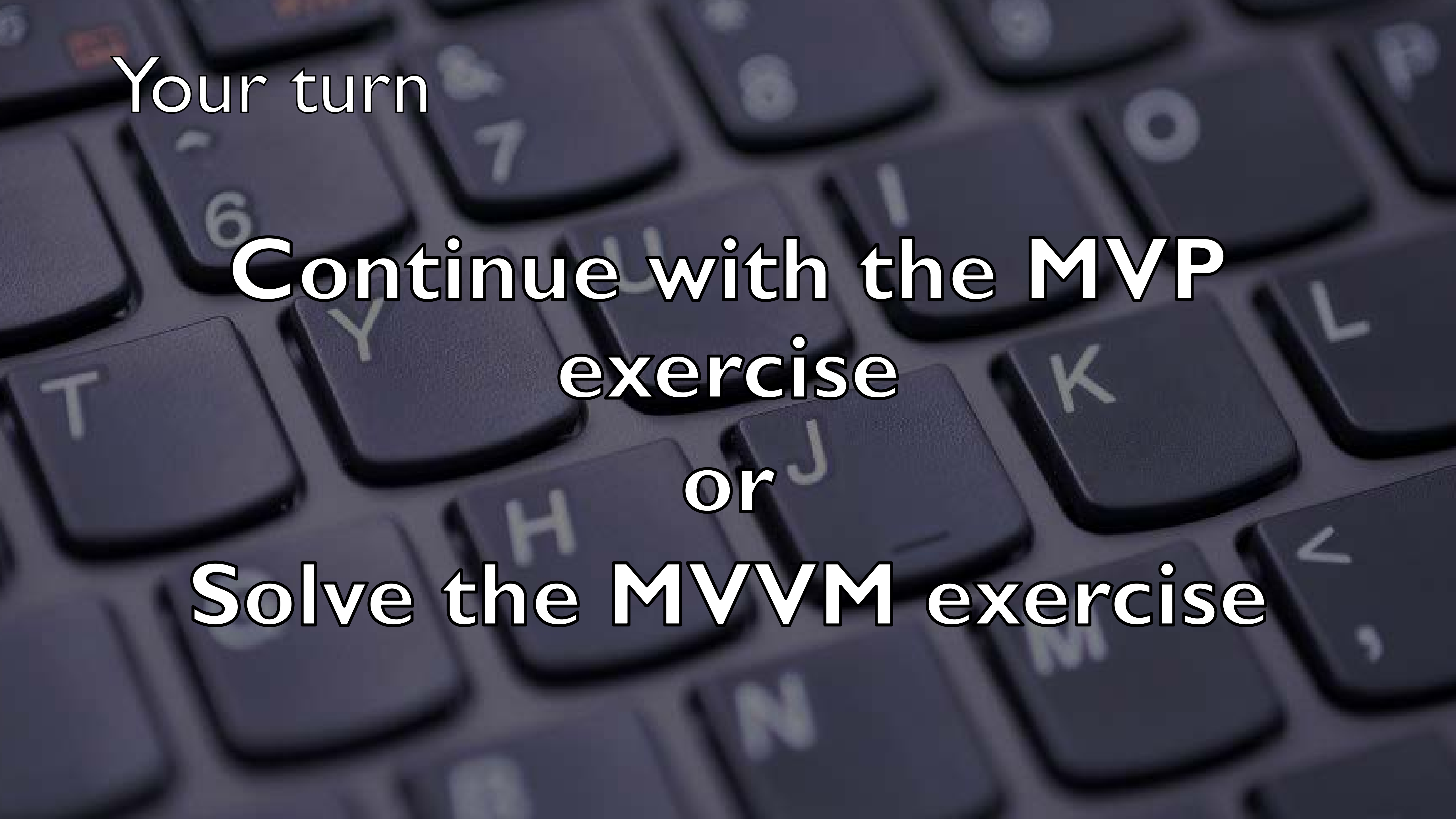
The View:

```
<Window x:Class="UsingAViewModelLocator.MainWindow"
        Title="BMI Calculator" Height="350" Width="525"
        DataContext="{Binding
                   Source={StaticResource ViewModelLocator},
                   Path=BmiViewModel}" >
```

MVVM and N-layer Architectures

MVVM and N-layer Architecture





Your turn

Continue with the MVP exercise

or

Solve the MVVM exercise

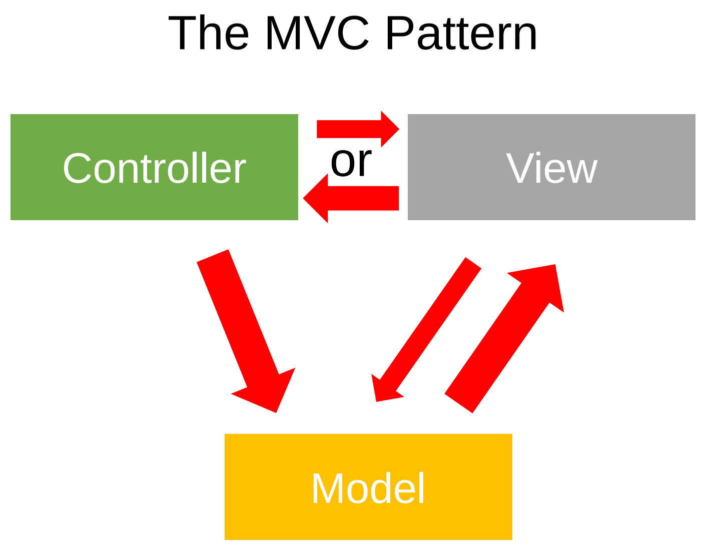
AARHUS UNIVERSITY

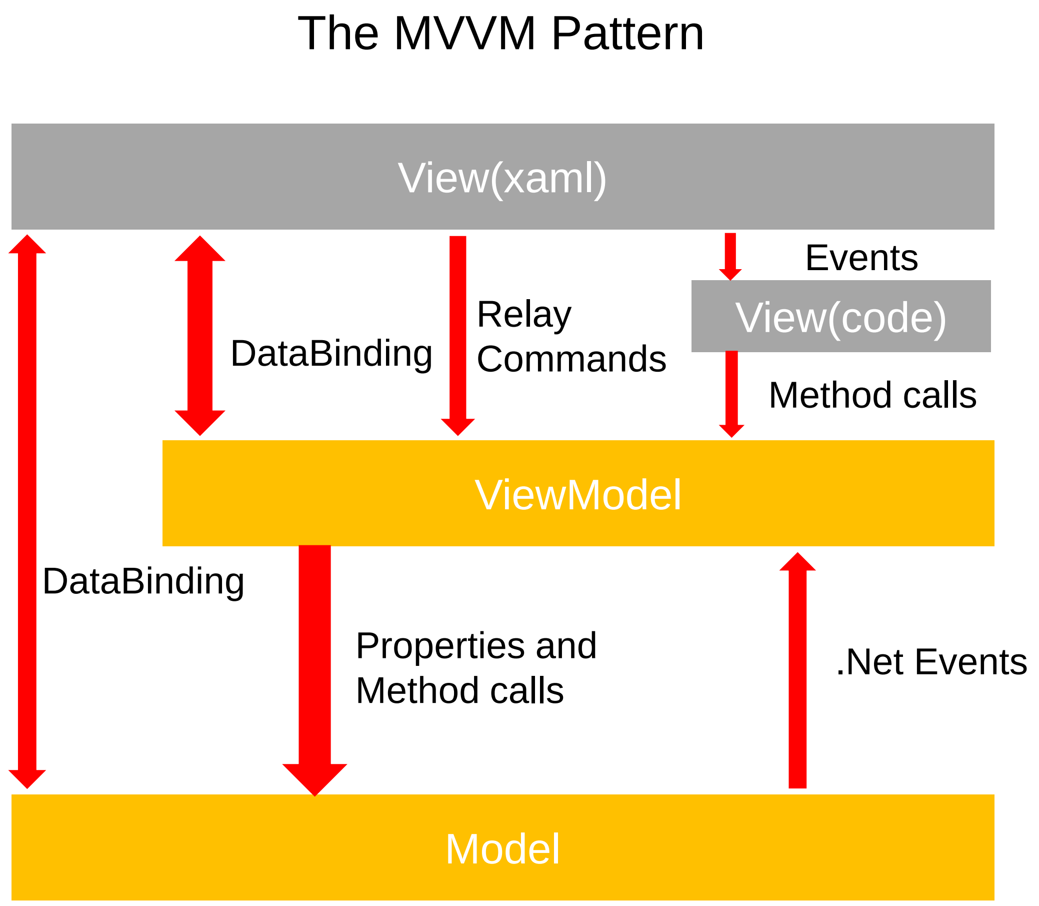
The spirit of MVVM

- Building UIs that utilize platform enhancements in WPF and Silverlight to provide good separation between UI and business logic in order to make those UIs easier to maintain by developers and designers

  - John Gossman (inventor of MVVM):
    - Model/View/ViewModel is tailored for modern UI development platforms where the View is the responsibility of a designer rather than a classic developer. The designer is generally a more graphical, artistic focused person, and does less classic coding than a traditional developer
  - Martin Fowler (comment on PresentationModel):
    - It's useful for allowing you to test without the UI, support for some form of multiple view and a separation of concerns which may make it easier to develop the user interface

From MVC to MVVM





Open Source MVVM Frameworks

- MS Patterns&Practices, PRISM – installeres via Nuget
- Laurent Bugnion, "MVVM Light Toolkit“ http:// http://www.galasoft.ch/mvvm
- Rob Eisenberg, "Caliburn” http://caliburnmicro.com/
- Tony Sneed, Simple MVVM Toolkit http://simplemvvmtoolkit.codeplex.com/
- Josh Smith, "MVVM Foundation“ http://mvvmfoundation.codeplex.com
- Sacha Barber, "Cinch v2” http://cinch.codeplex.com
- Daniel Vaughan, "Calcium SDK” http://www.calciumsdk.net.
- Karl Shifflett, "Ocean” http://karlshifflett.wordpress.com.
- Jbe, "WPF Application Framework (WAF)” http://waf.codeplex.com.

References and image sources

Images:

Binoculars: https://www.bird-watching.net/wp-content/uploads/2016/05/brds.jpg