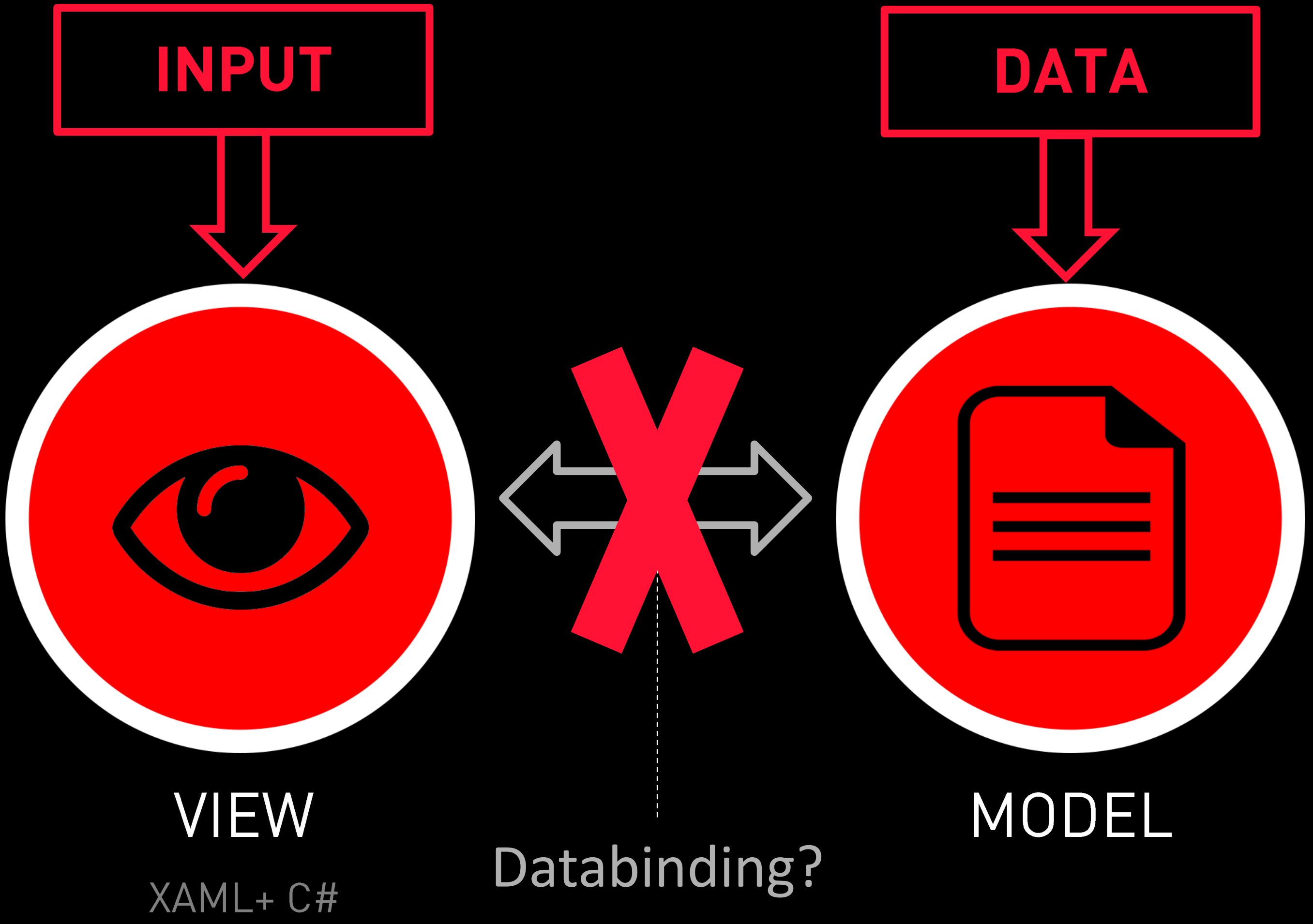
TOOL DEVELOPMENT

MVVM

A DIGITAL ARTS AND ENTERTAINMENT PRESENTATION

MVVM

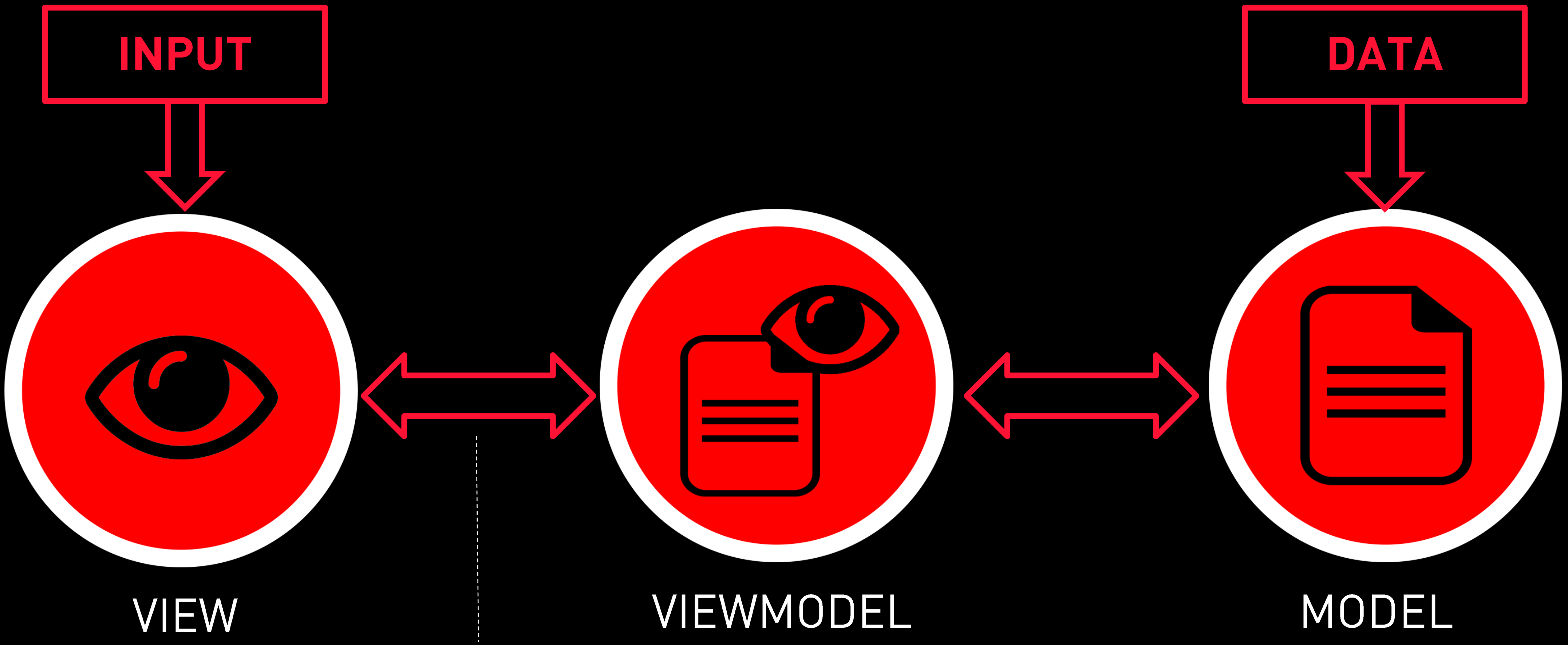
SITUATION



- DATA will most likely be defined by other applications (Game, Server application, website, ...?)
- We want to use Binding between our view and data:
  - Properties {get; set;}
  - Collection changes? (ObservableCollection,...)
  - WPF specific types unknown to model? (ImageSource, ...)
- It's NOT A GOOD IDEA to change the structure of the data to match our needs for WPF!!!

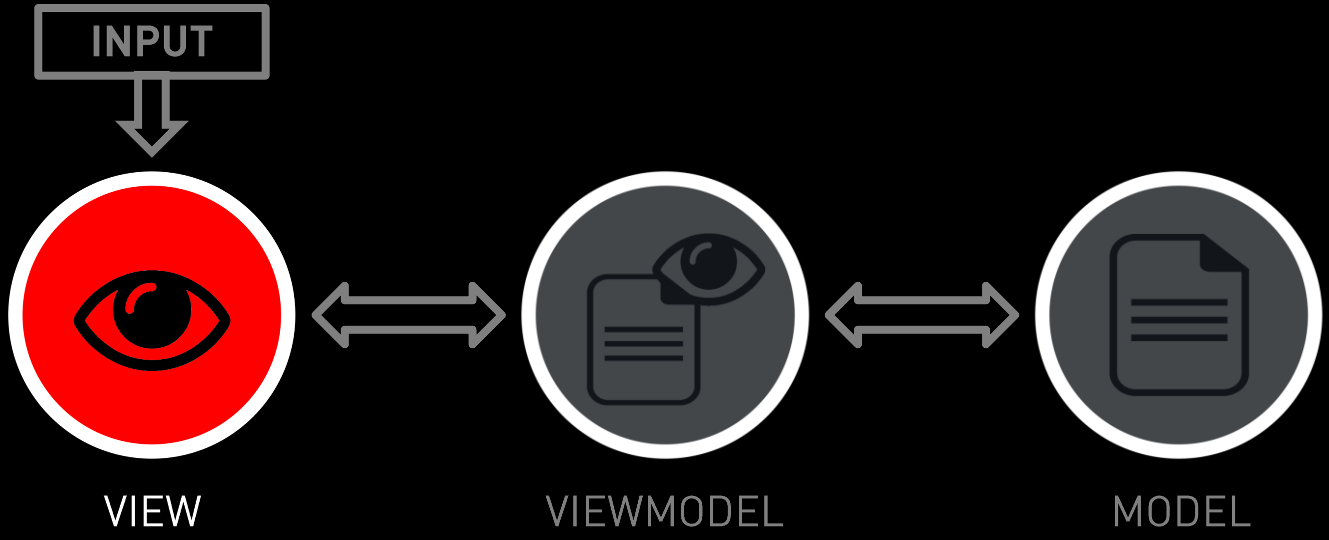
MVVM

MODEL VIEW VIEWMODEL – CONCEPT



Databinding
= Loose Coupling

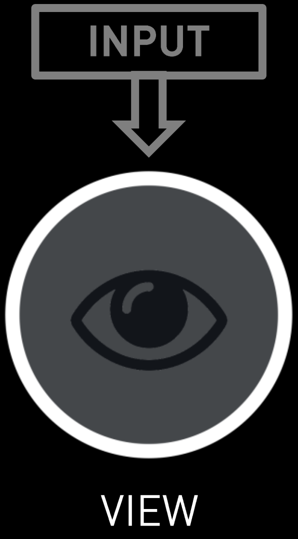
MVVM
VIEW



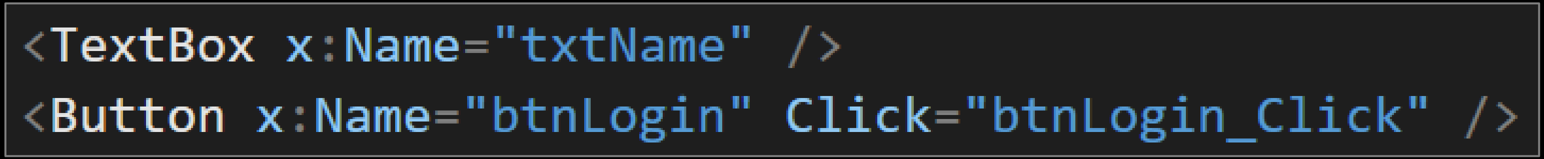
- UI / XAML
- **NO** (well.. minimal) code-behind
  - *MainWindow.cs will only call InitializeComponent, no other code*
- **NO** business logic!
- **NO** eventhandlers (we will use Commands)

```
<Button ~~Click="btnOk_Click"~~
```

VIEW – EXAMPLE WITHOUT MVVM

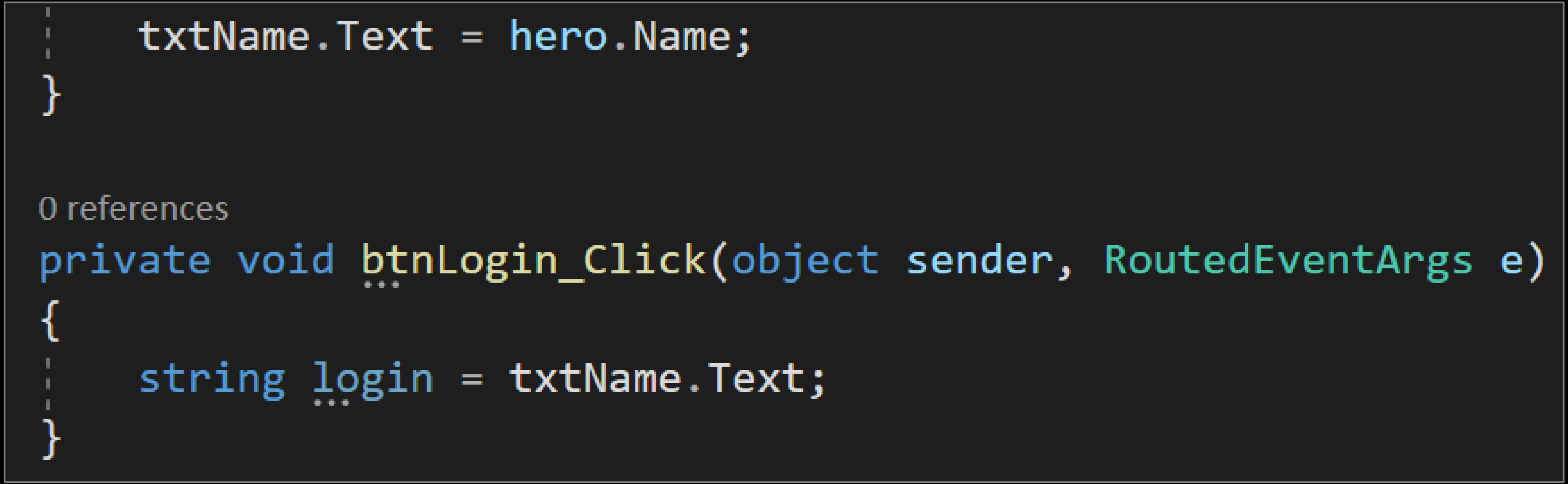


MainWindow.xaml



```
<TextBox x:Name="txtName" />
<Button x:Name="btnLogin" Click="btnLogin_Click" />
```

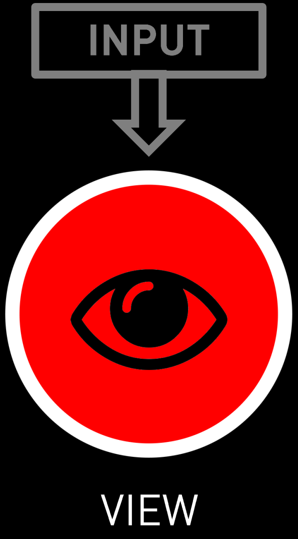
MainWindow.xaml.cs



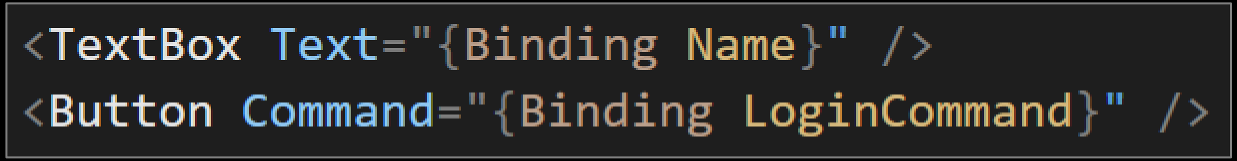
```
    txtName.Text = hero.Name;
}

0 references
private void btnLogin_Click(object sender, RoutedEventArgs e)
{
    string login = txtName.Text;
}
```

VIEW – EXAMPLE MVVM

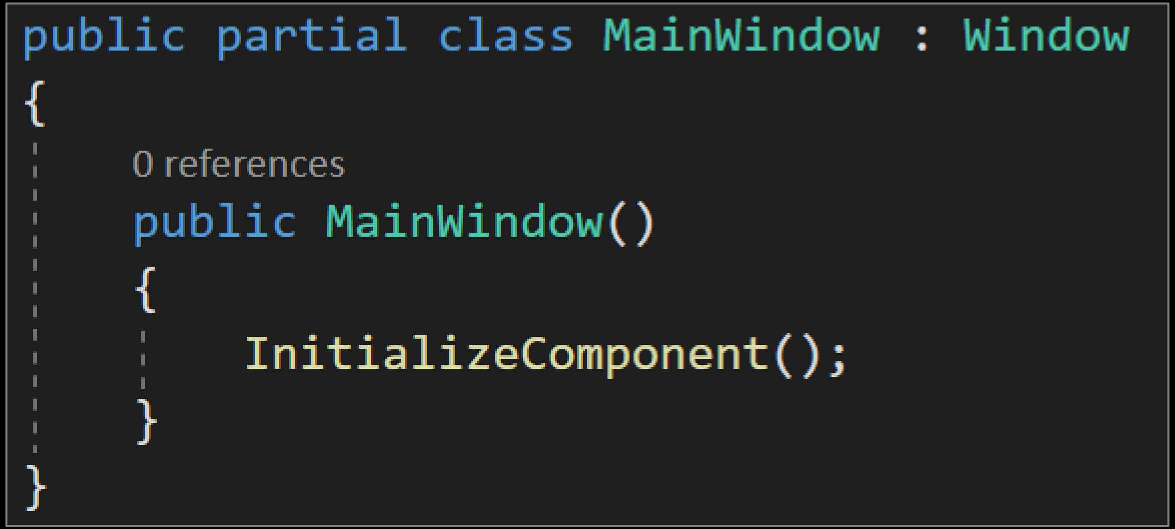


MainWindow.xaml



```
<TextBox Text="{Binding Name}" />
<Button Command="{Binding LoginCommand}" />
```

MainWindow.xaml.cs

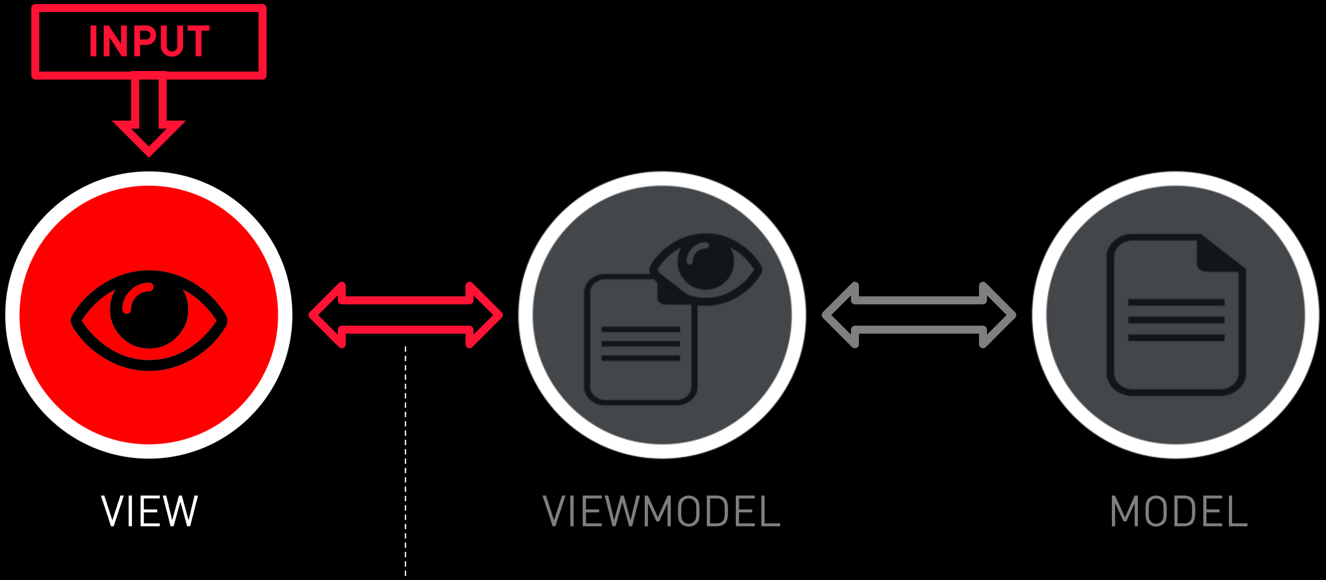


```
public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

→ no code, only InitializeComponent

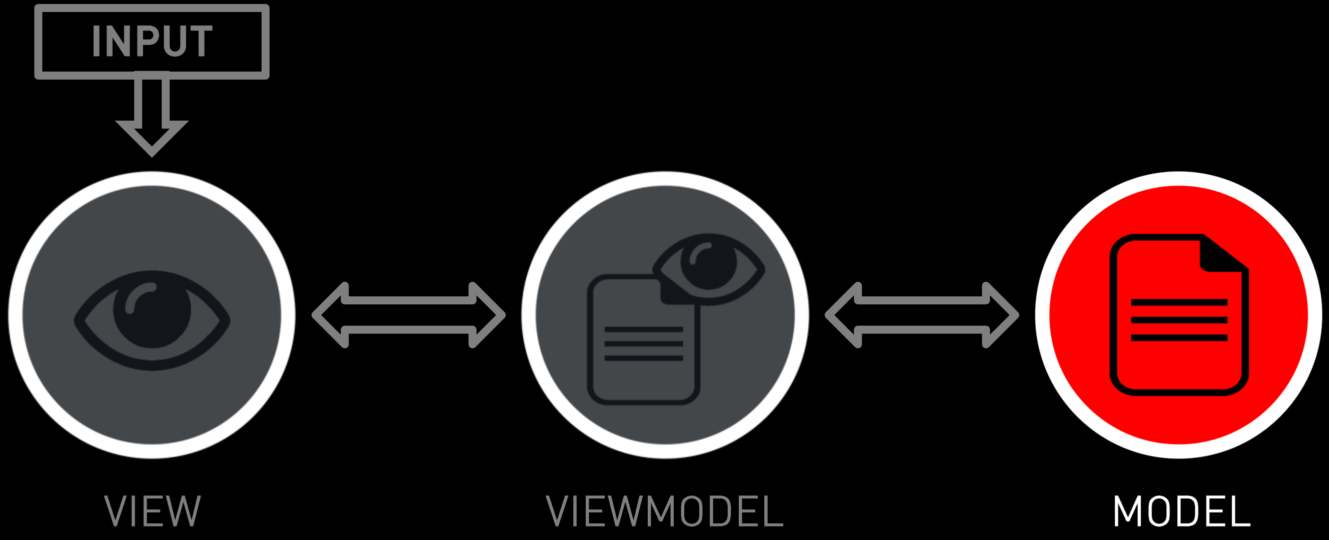
MVVM

VIEW – VIEWMODEL



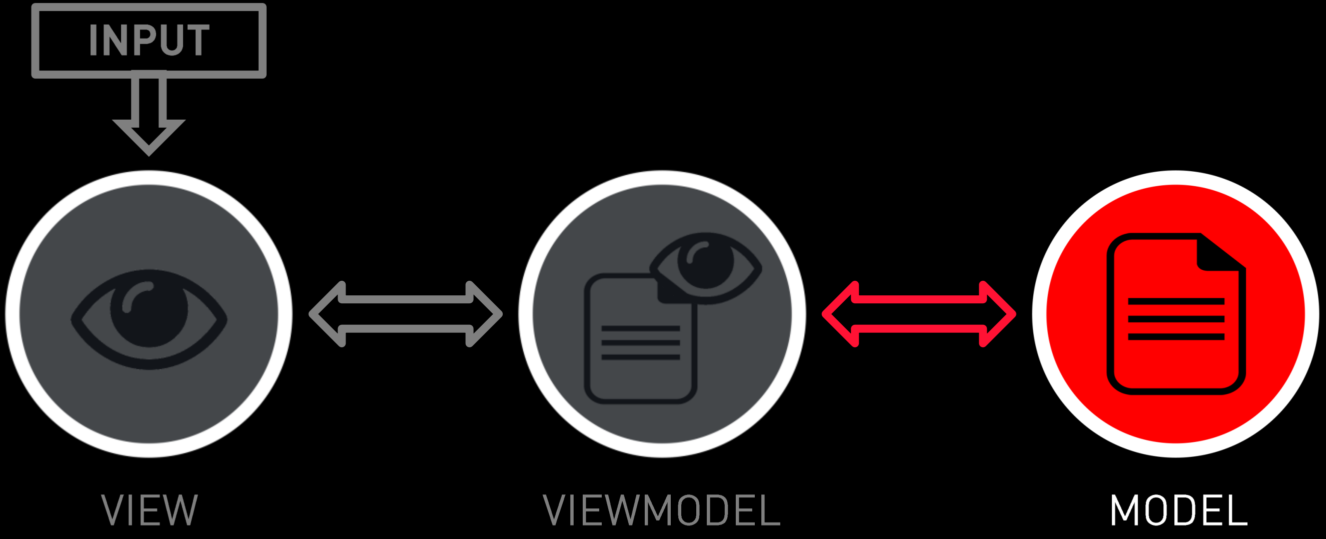
- VIEW binds to properties & commands (actions) of VIEWMODEL
- (view has no connection to the MODEL!)

MVVM MODEL



- POD (Plain Old Data)
- 'Plain' classes, interfaces
  - Remember the Model folder? ☺
- Libraries

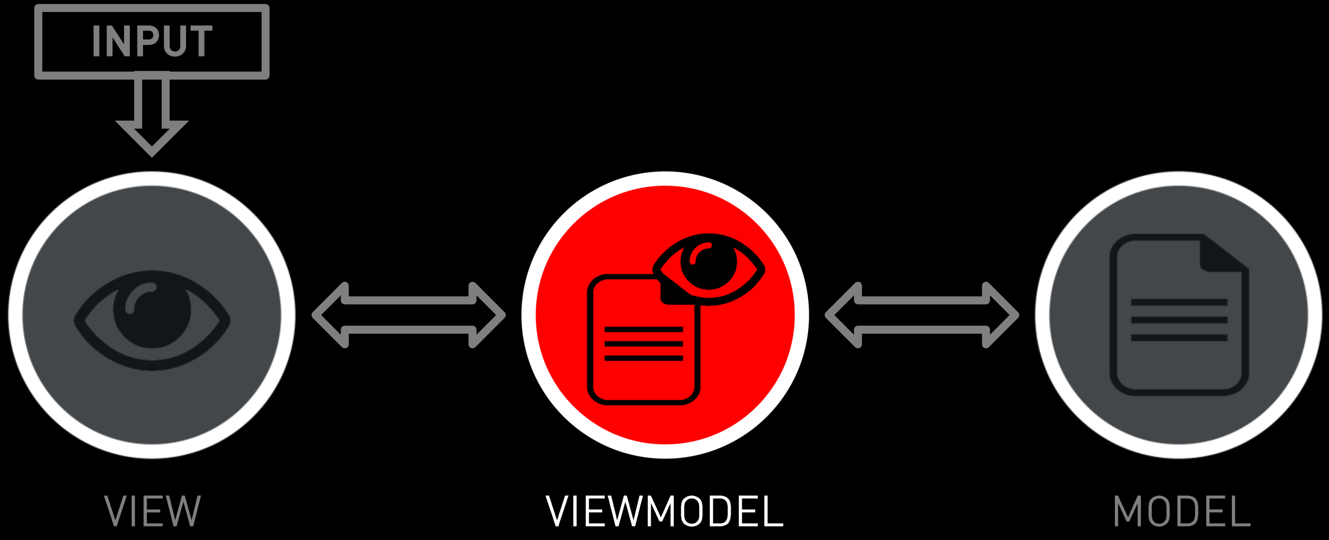
MVVM
MODEL



- Is used by the VIEWMODEL to create/use objects
- MODEL might notify the VIEWMODEL if a property has changed

MVVM

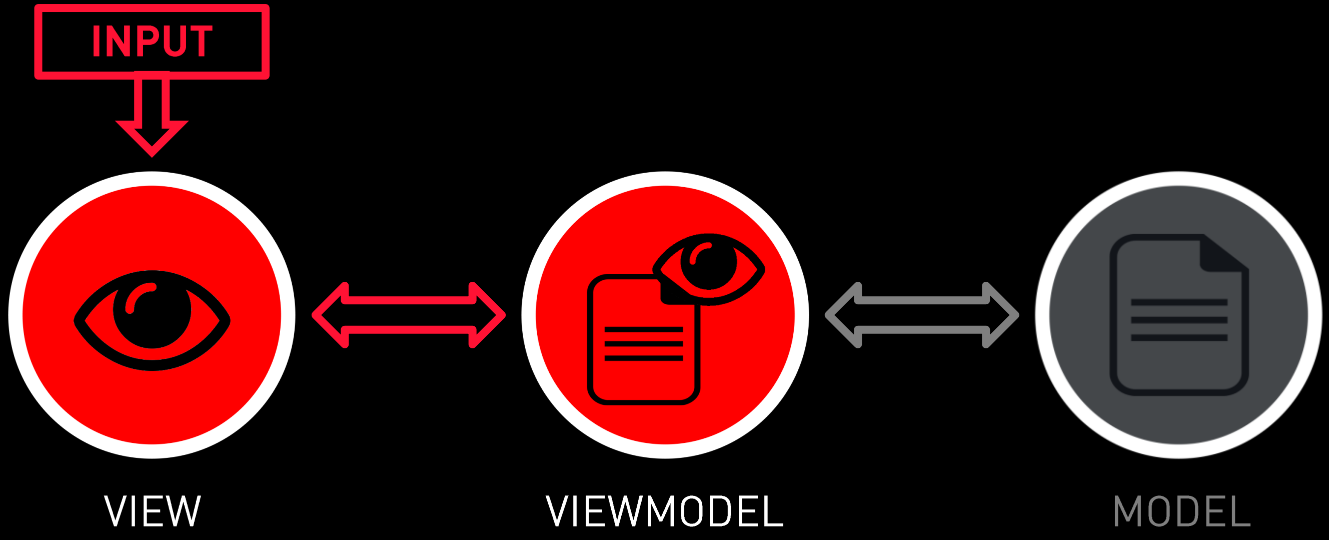
VIEWMODEL



- Business logic
    - Example:
        - Get a list of objects (using MODEL for structure)
        - Save a new object that was INPUT in VIEW
        - ...
- Contains data that might be displayed in VIEW
- (Testable using Unit Tests)

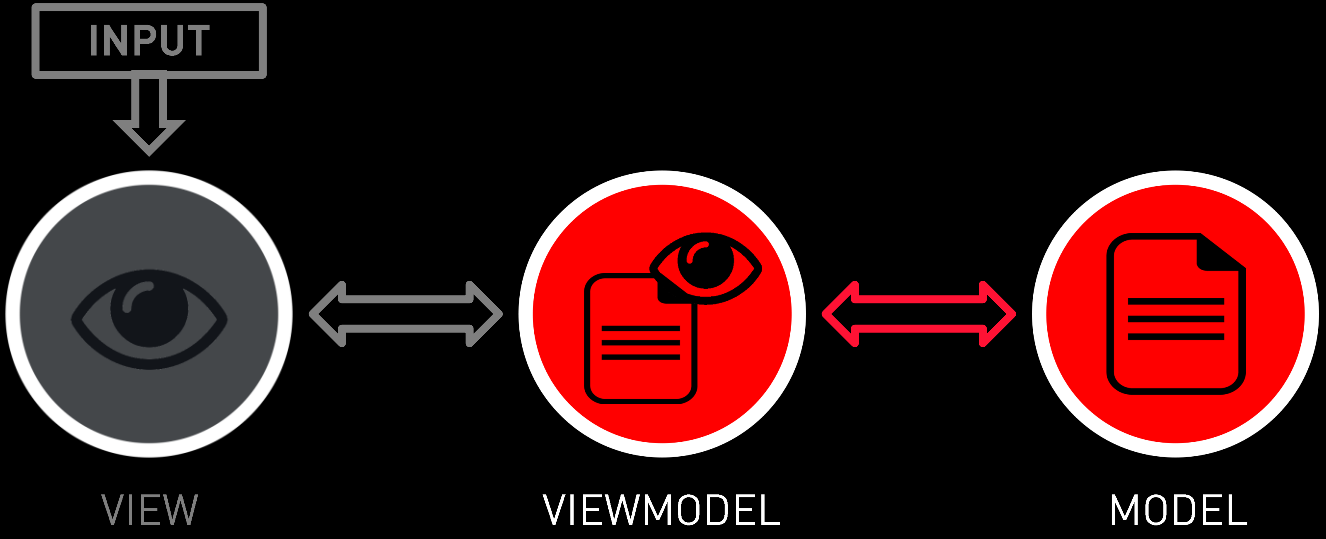
MVVM

VIEWMODEL



- Notify the VIEW in case something has changed
  - ➢ "You might want to refresh part of your view, VIEW"
- Listen to commands of VIEW and respond to them

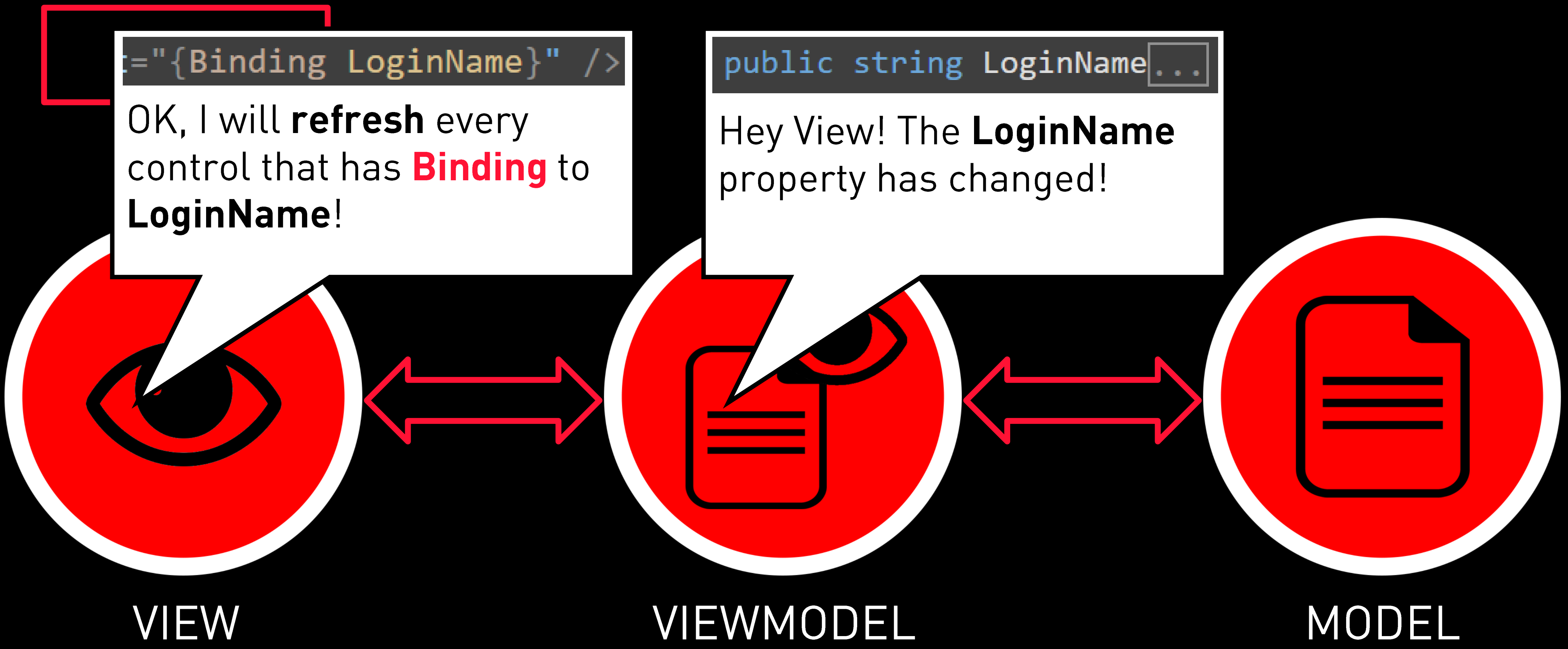
MVVM
VIEWMODEL



- Use the MODEL for data structure
- Listen to the MODEL for notifications of changes

MVVM

MODEL VIEW VIEWMODEL



```
t="{Binding LoginName}" />
```

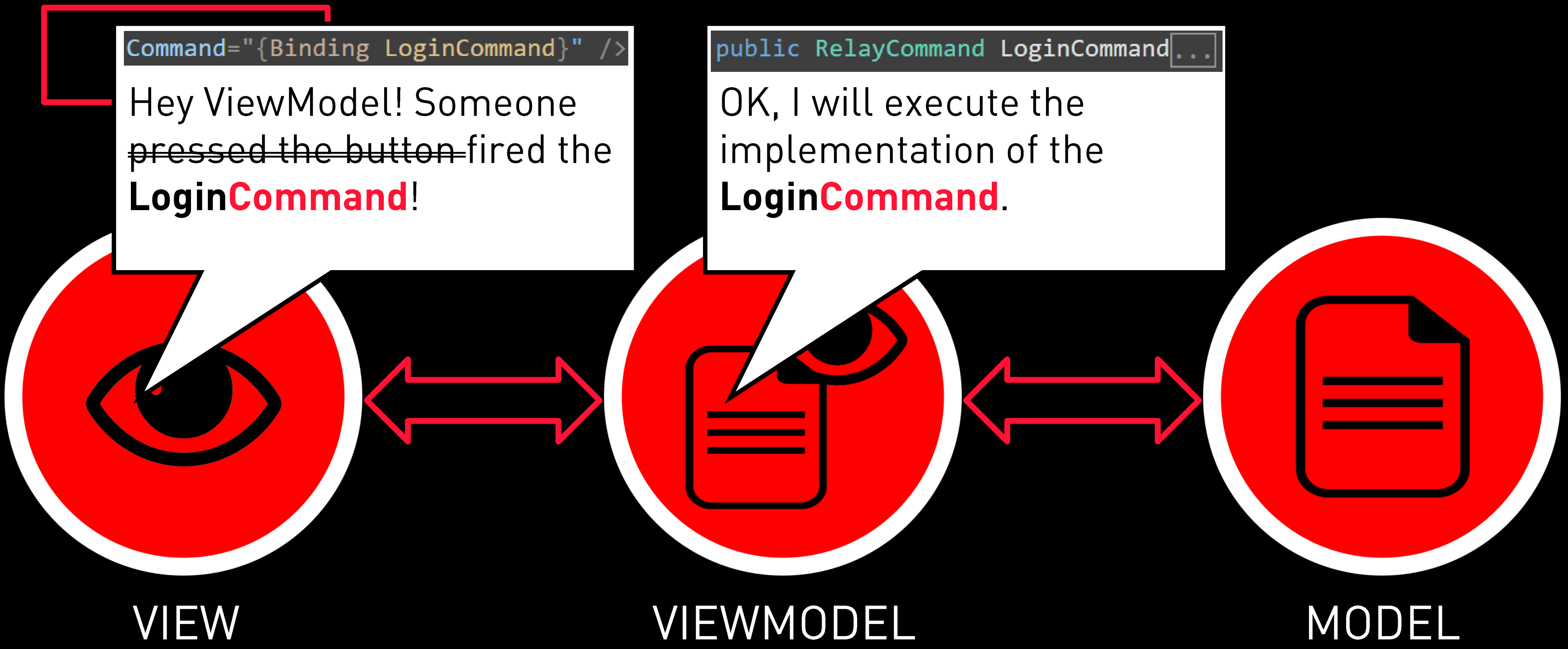
OK, I will **refresh** every control that has **Binding** to **LoginName**!

```
public string LoginName...
```

Hey View! The **LoginName** property has changed!

MVVM

MODEL VIEW VIEWMODEL



```
Command="{Binding LoginCommand}" />
```

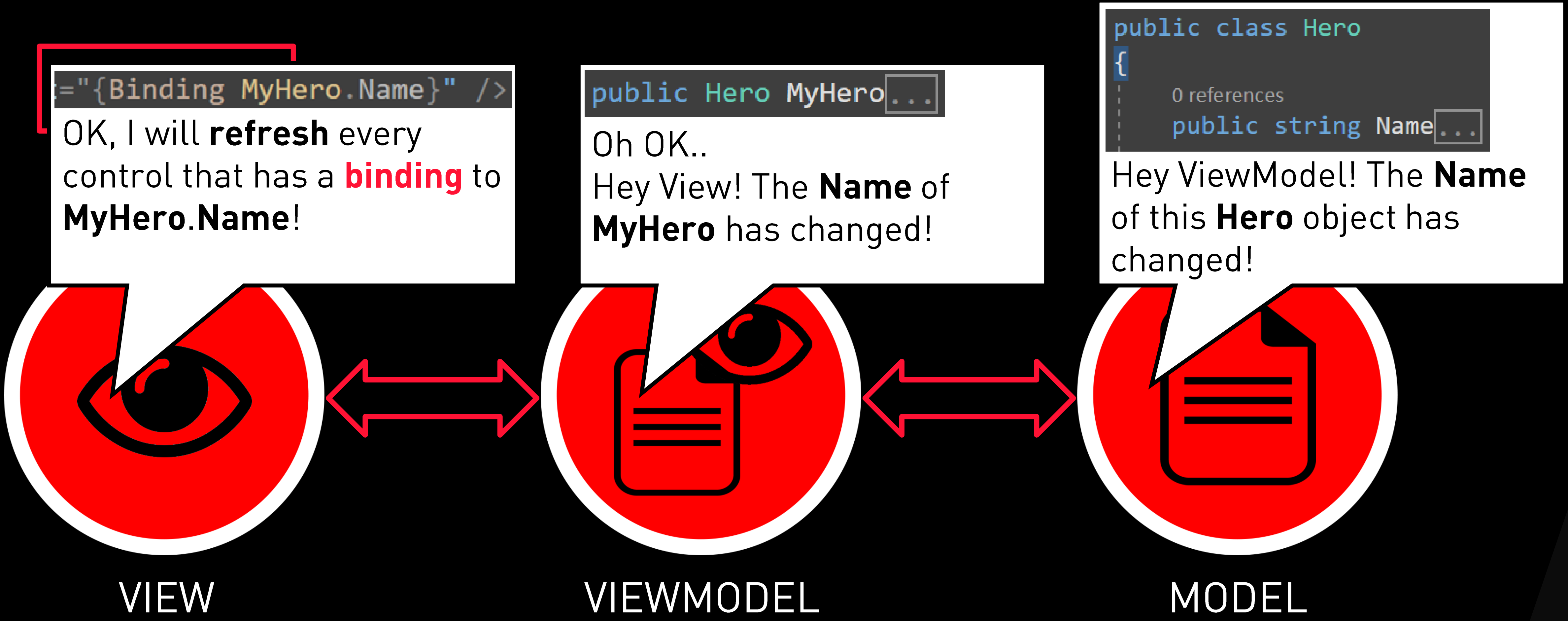
Hey ViewModel! Someone ~~pressed the button~~ fired the **LoginCommand**!

```
public RelayCommand LoginCommand...
```

OK, I will execute the implementation of the **LoginCommand**.

MVVM

MODEL VIEW VIEWMODEL



```
="{Binding MyHero.Name}" />
```

OK, I will **refresh** every control that has a **binding** to **MyHero.Name**!

```
public Hero MyHero...
```

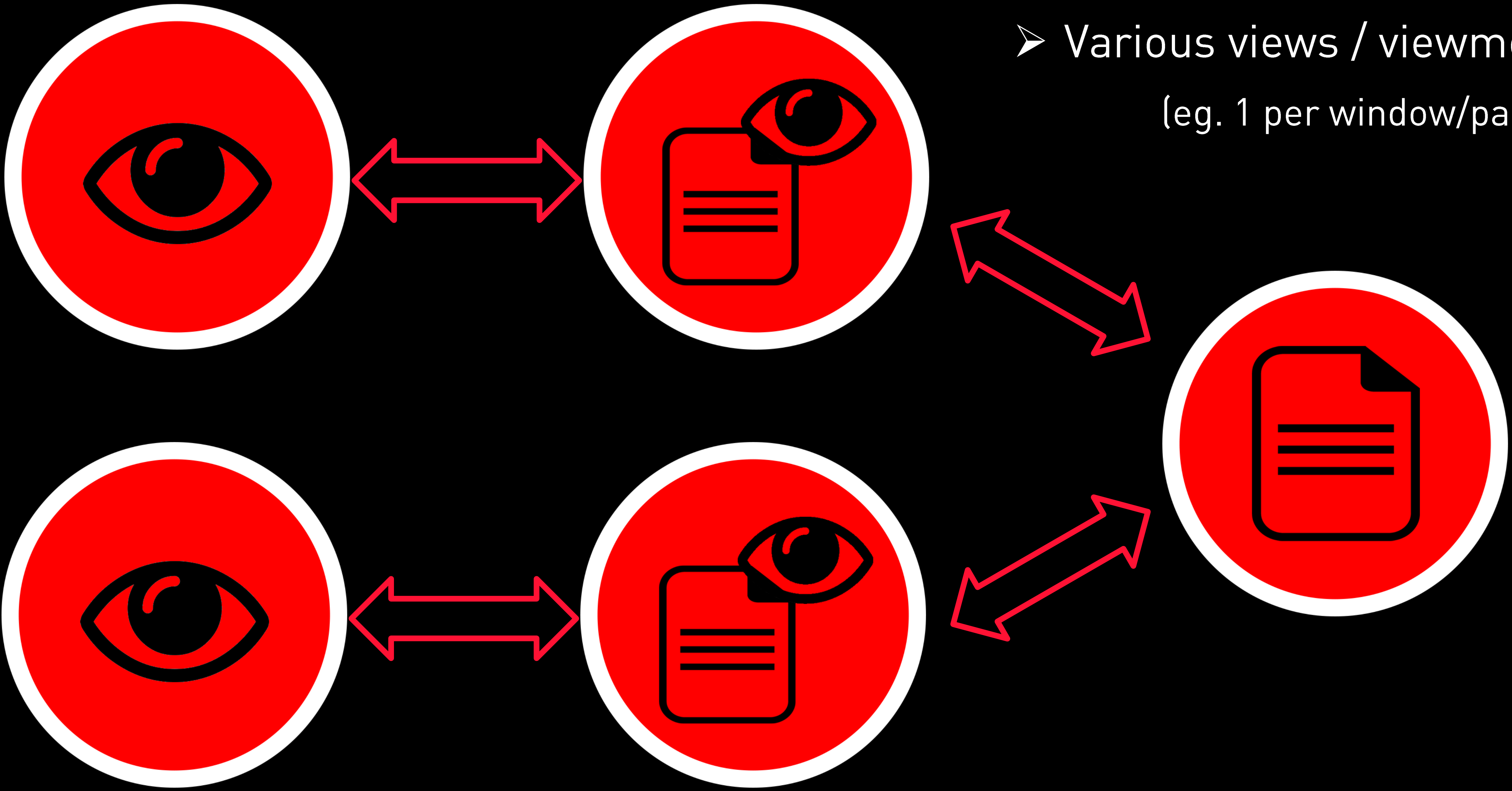
Oh OK..
Hey View! The **Name** of **MyHero** has changed!

```
public class Hero
{
    0 references
    public string Name...
```

Hey ViewModel! The **Name** of this **Hero** object has changed!

MVVM

MODEL VIEW VIEWMODEL

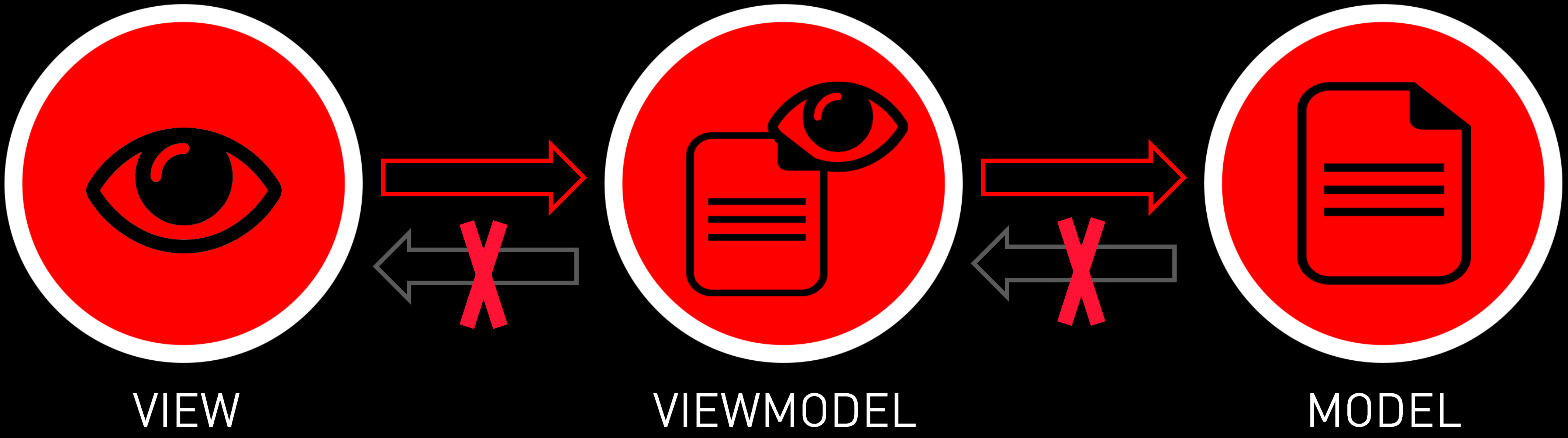


- Various views / viewmodels

  (eg. 1 per window/page)

MVVM Sidenote

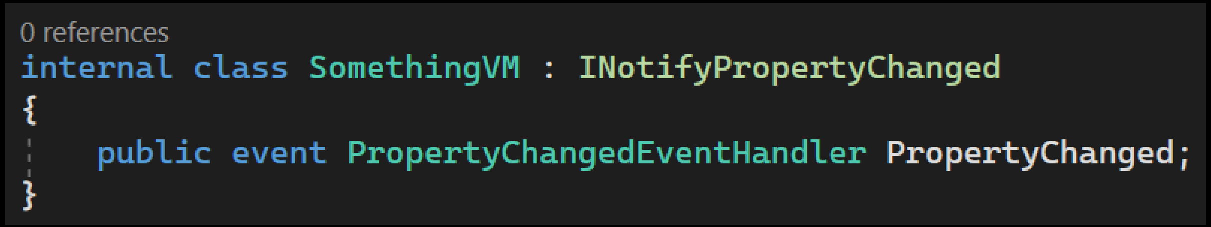
AVOID CIRCULAR DEPENDENCIES



- Dependencies in 1 WAY ONLY!
- View references ViewModel
- ViewModel references Model.
- NEVER THE OTHER WAY AROUND!!!
    - →ViewModel will never call a method in View directly!
      →View registers to Events in the ViewModel

OKAY, BUT HOW?

- using the INotifyPropertyChanged interface

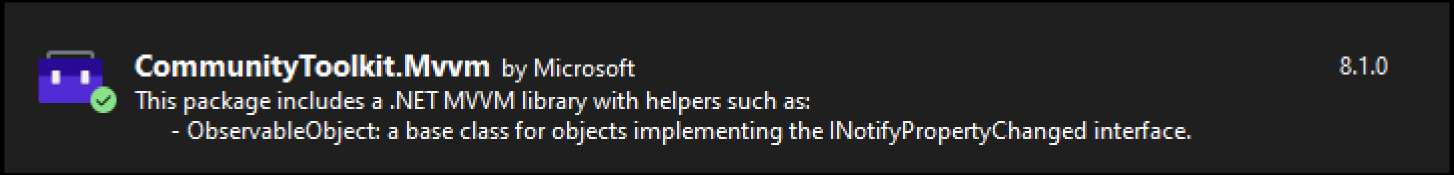


```
0 references
internal class SomethingVM : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
}
```

- using the NuGet package MVVM Toolkit

  Provides base classes ObservableRecipient, ObservarbleObject

  (which implement the INotifyPropertyChanged interface)

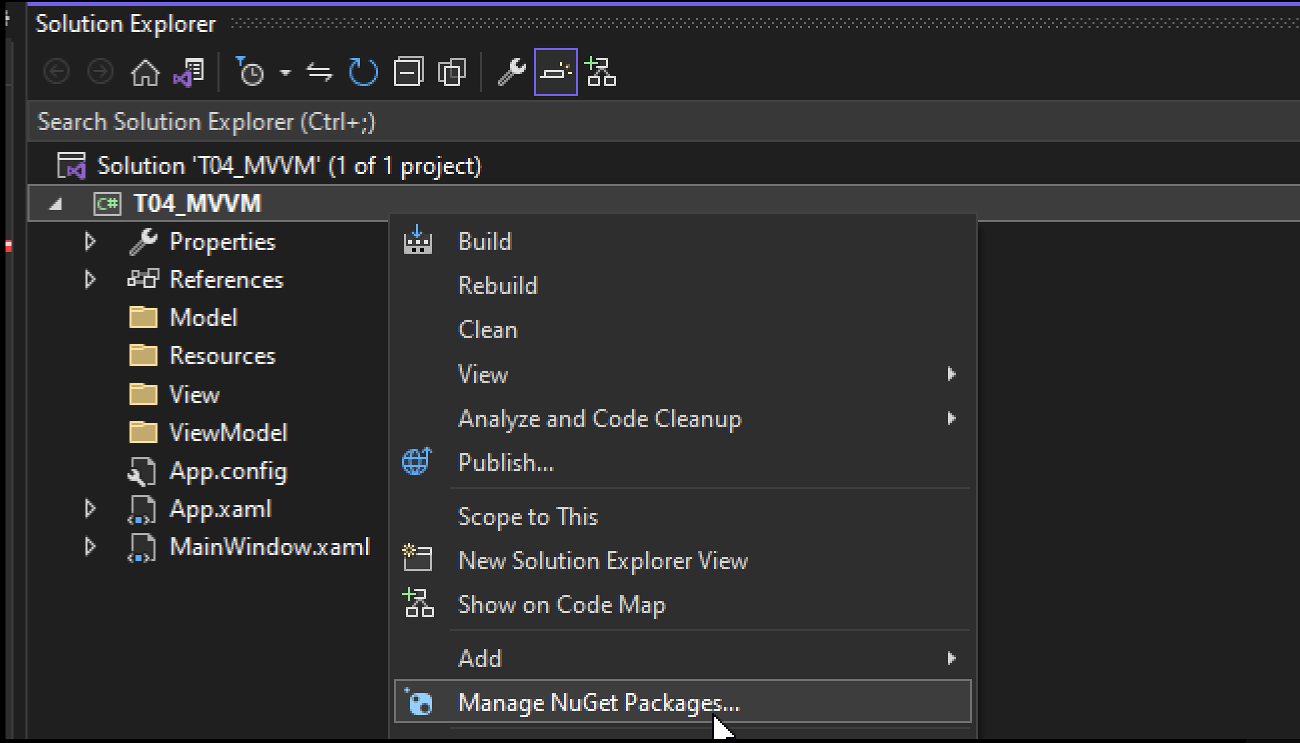


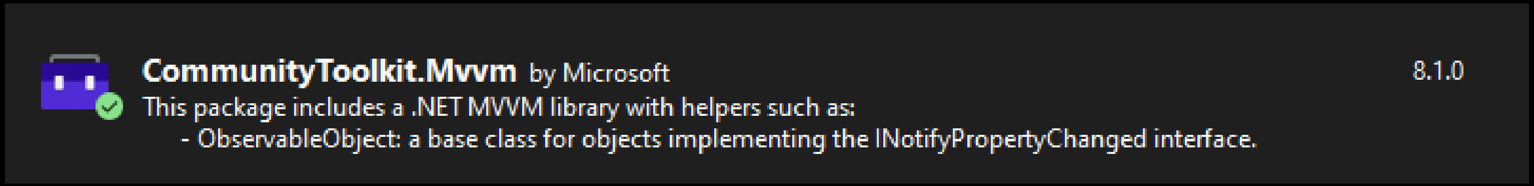
CommunityToolkit.Mvvm by Microsoft 8.1.0
This package includes a .NET MVVM library with helpers such as:
- ObservableObject: a base class for objects implementing the INotifyPropertyChanged interface.

MVVM project

HANDS-ON: MVVM PROJECT SETUP (1)

- Create a WPF project called T04_MVVM
- Create the folder structure:
  - Model
  - Resources
  - View
  - ViewModel
- Install MVVM Toolkit Package:





CommunityToolkit.Mvvm by Microsoft 8.1.0

This package includes a .NET MVVM library with helpers such as:
- ObservableObject: a base class for objects implementing the INotifyPropertyChanged interface.

MVVM project

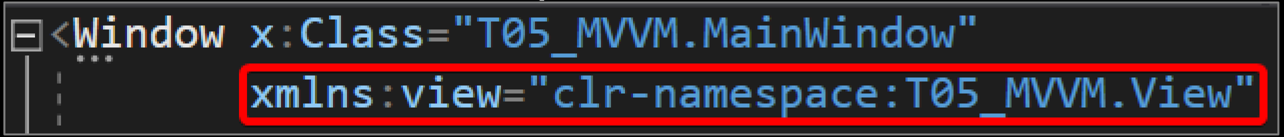
HANDS-ON: MVVM PROJECT SETUP (2)

- In Model, add the given model (class) Hero.cs
  - Check if the namespace matches your project's namespace!
- In Resources, add the given image superhero.jpg
- In View, create a new **page** (WPF) called 'HeroPage'
  - Replace the Grid by the given xaml code "HeroPageContent.xaml"
- In ViewModel, create a class called 'HeroPageVM'
  - Let it inherit the ObservableObject class (CommunityToolkit.Mvvm.ComponentModel namespace)
  - Create a property called 'UserName' (string), default value = your name
  - Create a second property called 'CurrentHero' (Hero)
    - (you will need a reference to your Model namespace)
    - give default values for Name and RealName (not Description):

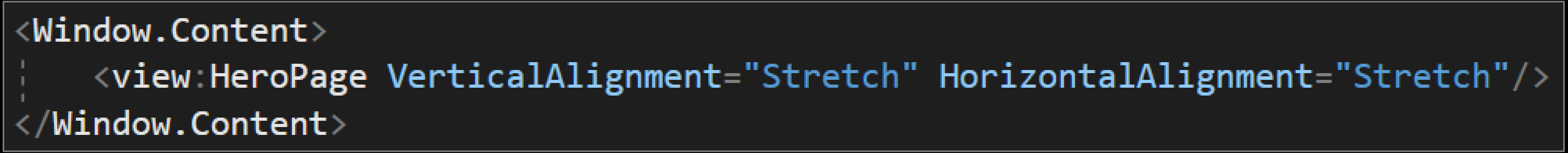
```
public Hero CurrentHero { get; set; }
    = new Hero() { Name = "SuperMe", RealName = "The real me" };
```

HANDS-ON: MVVM PROJECT SETUP (3)

- MainWindow will only contain a Page object
  - 'Navigating' = changing the page object in MainWindow
  - Set the background of the MainWindow to dark gray (#333333)

- Set HeroPage as the page: in **MainWindow.xaml**:
  - Replace the Grid by a `<Window.Content></Window.Content>` tag
  - Add the View namespace in xaml:
    - 

```
<Window x:Class="T05_MVVM.MainWindow"
    ...
        xmlns:view="clr-namespace:T05_MVVM.View"
```

  - Add the HeroPage as window content using the view: prefix:
    - 

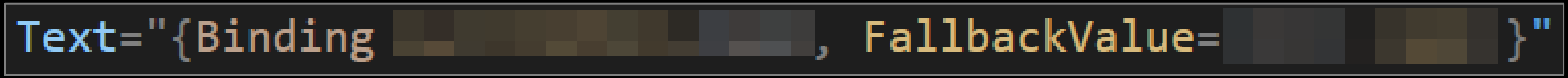
```
<Window.Content>
    <view:HeroPage VerticalAlignment="Stretch" HorizontalAlignment="Stretch"/>
</Window.Content>
```

  - Run: a static HeroPage should appear

MVVM project

HANDS-ON: CONNECT VIEW TO VIEWMODEL

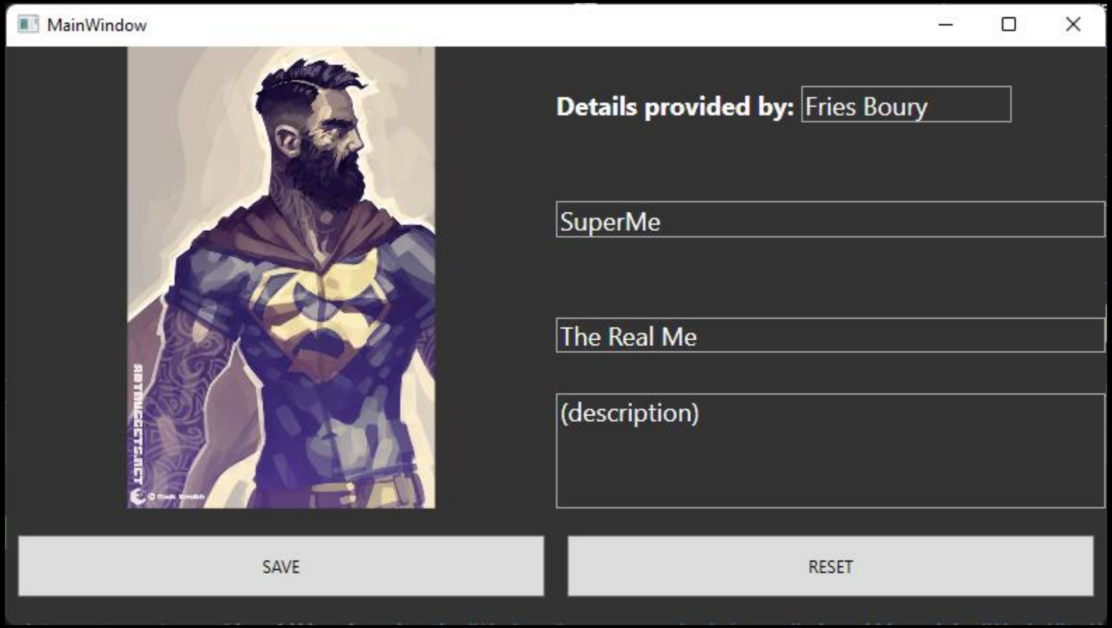
- In HeroPage.xaml, add the ViewModel namespace
  - See previous step on how to do this
  - Choose a prefix other than view, eg. 'vm'
- Add a `<Page.DataContext>` tag as the first element in your Page
  - Inside this tag, add a MainPageViewModel element
  - ! This creates a new instance of MainPageVM !
  - ! Every element in this page can now 'listen' to the ViewModel using Binding !
- Set the Binding on each TextBox element (see comments) and Image
  - As a second parameter when Binding, you can add a FallBackValue,
  - this is what appears when the property cannot be found,
  - and can be very useful to get your layout right in design mode



```
Text="{Binding [redacted], FallbackValue=[redacted]}"
```

MVVM project

HANDS-ON: RUN YOUR PROJECT TO TEST


MainWindow

Details provided by: Fries Boury

SuperMe

The Real Me

(description)

- Set a breakpoint with both the getter and the setter of Description
- Run your project
    - See the getter being called? This is because of Binding! (ask value to display)
- Type something in the Description textbox and hit Tab
    - See the setter being called? This is two-way Binding! (input changes prop value)

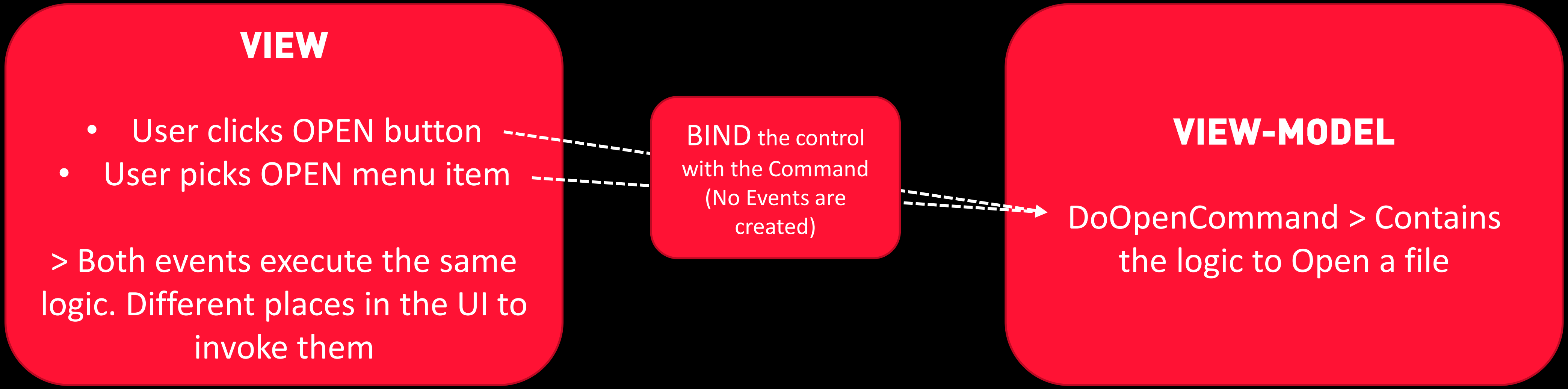
RELAYCOMMANDS

COMMANDS IN MVVM

MVVM project

MVVM COMMANDS

- Replace Events
  - Loose Coupling (No hard links between View & ViewModel)
  - <Button ~~Click="btnOk_Click"~~
- Contain the execution logic of the Event/Action



VIEW

- User clicks OPEN button
- User picks OPEN menu item

> Both events execute the same logic. Different places in the UI to invoke them

BIND the control with the Command (No Events are created)

VIEW-MODEL

DoOpenCommand > Contains the logic to Open a file

MVVM project

MVVM EXAMPLE: PROPERTY OF TYPE RELAYCOMMAND

- .xaml:

```
<Button Content="SAVE" Grid.Row="4" Margin="8" Command="{Binding SaveCommand}"
        Padding="13" VerticalAlignment="Center"/>
```

- Inside ViewModel: `using CommunityToolkit.Mvvm.Input;`

!

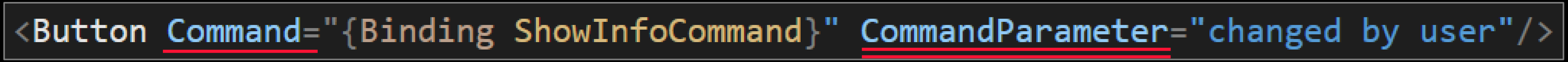
```
1 reference
public RelayCommand SaveCommand { get; private set; }
0 references
public HeroPageVM()
{
    SaveCommand = new RelayCommand(SaveHero);
}
```

➢ property type RelayCommand

```
private void SaveHero()
{
    Console.WriteLine("=> saving hero....");
    Console.WriteLine(CurrentHero);
}
```

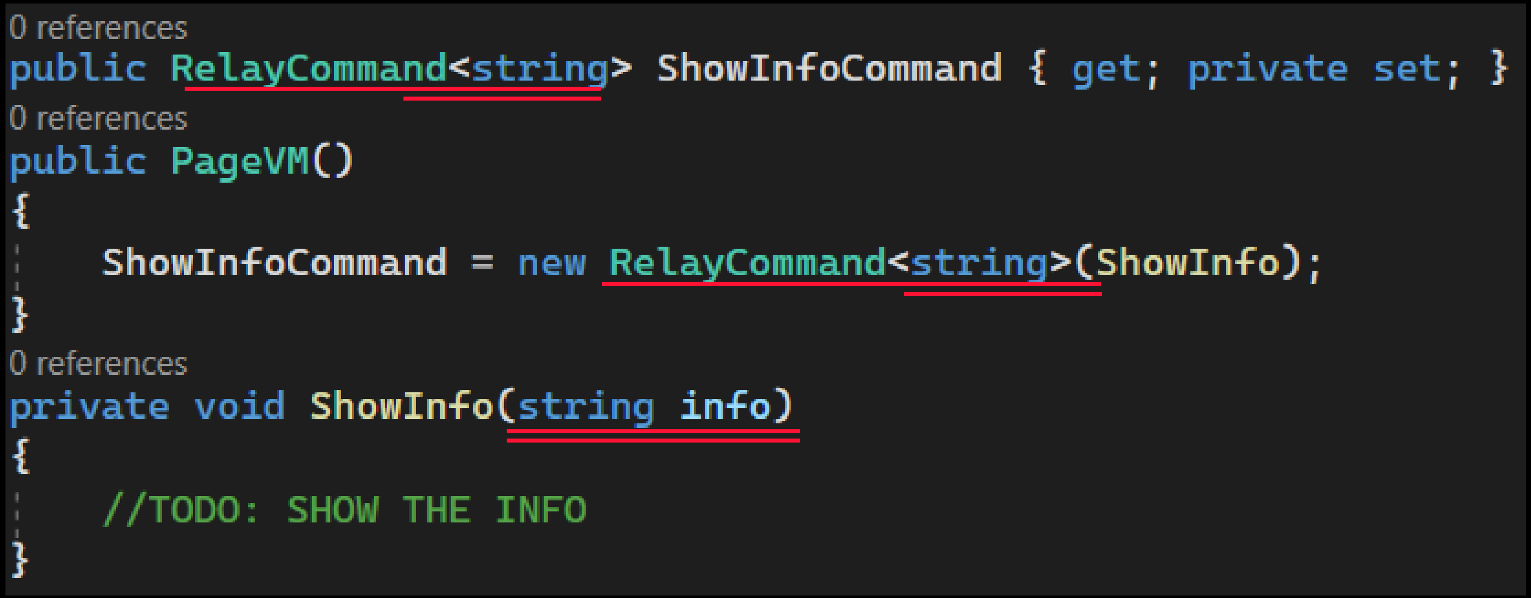
MVVM EXAMPLE: RELAYCOMMAND<T> (WITH PARAMETER)

- .xaml:



```
<Button Command="{Binding ShowInfoCommand}" CommandParameter="changed by user"/>
```

- Inside ViewModel (in this case, <T> is a string):

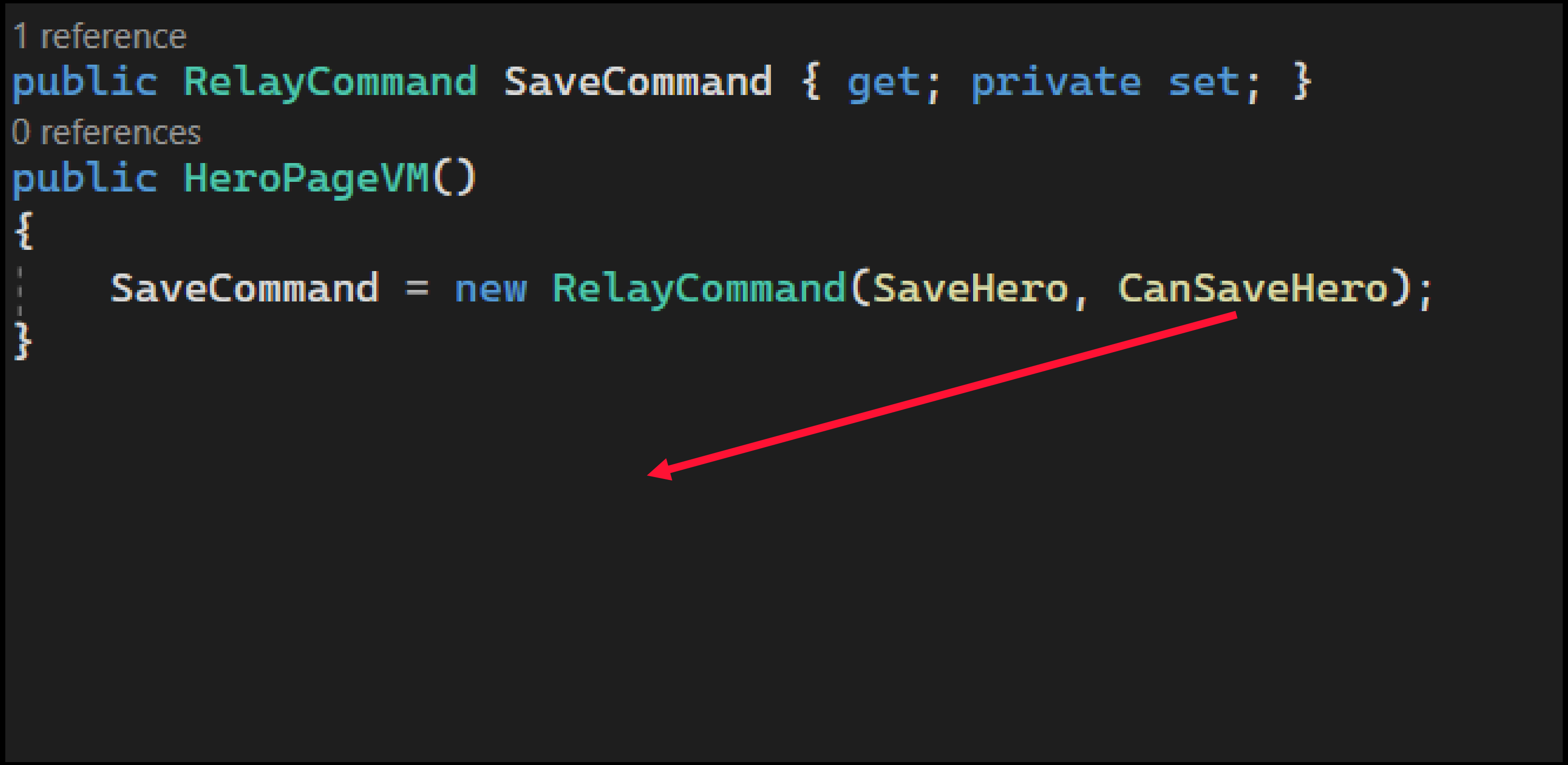


```
0 references
public RelayCommand<string> ShowInfoCommand { get; private set; }
0 references
public PageVM()
{
    ShowInfoCommand = new RelayCommand<string>(ShowInfo);
}
0 references
private void ShowInfo(string info)
{
    //TODO: SHOW THE INFO
}
```

MVVM project

MVVM EXAMPLE: ENABLE / DISABLE

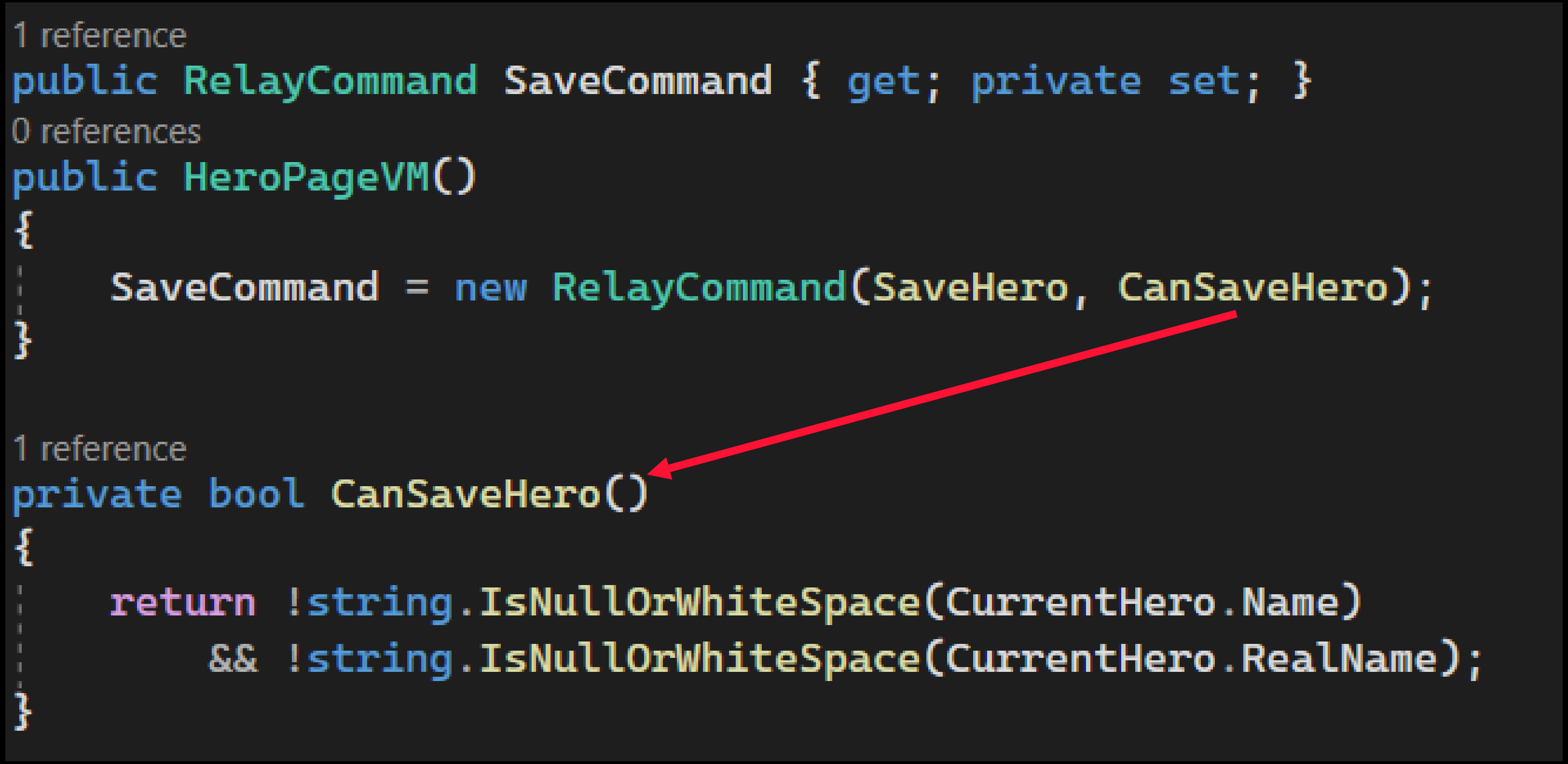
- Inside ViewModel (in this case, only enable save when a name and realname are given):



```
1 reference
public RelayCommand SaveCommand { get; private set; }
0 references
public HeroPageVM()
{
    SaveCommand = new RelayCommand(SaveHero, CanSaveHero);
}
```

MVVM EXAMPLE: ENABLE / DISABLE

- Inside ViewModel (in this case, only enable save when a name and realname are given):

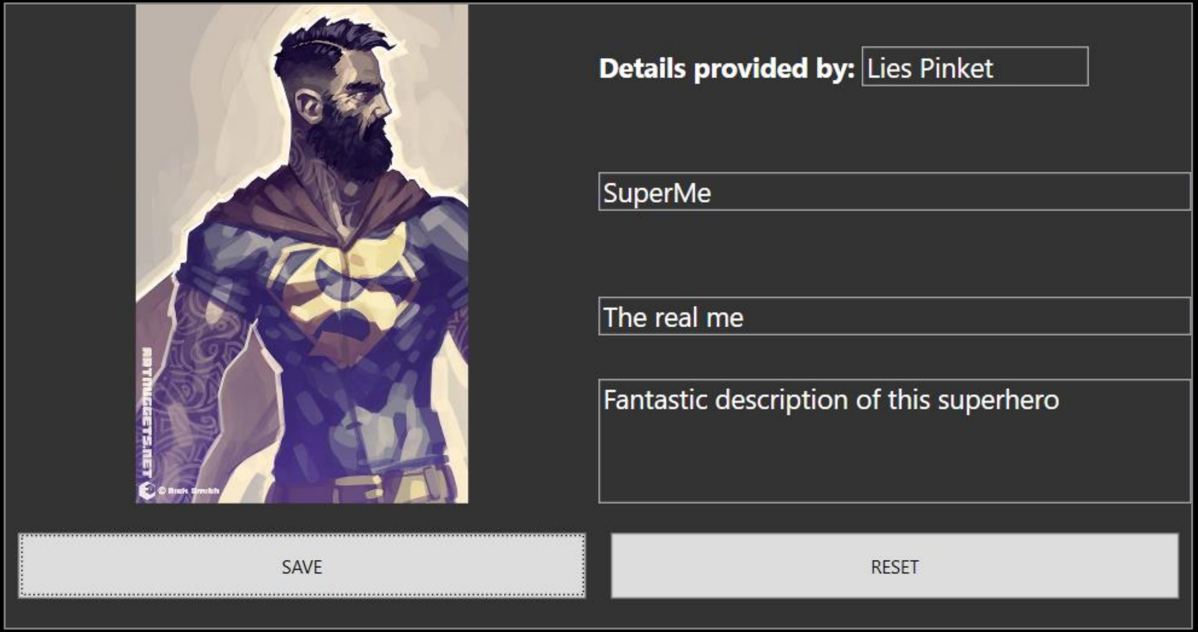


```
1 reference
public RelayCommand SaveCommand { get; private set; }
0 references
public HeroPageVM()
{
    SaveCommand = new RelayCommand(SaveHero, CanSaveHero);
}

1 reference
private bool CanSaveHero()
{
    return !string.IsNullOrWhiteSpace(CurrentHero.Name)
        && !string.IsNullOrWhiteSpace(CurrentHero.RealName);
}
```

MVVM project

HANDS-ON: ADD RESET COMMAND

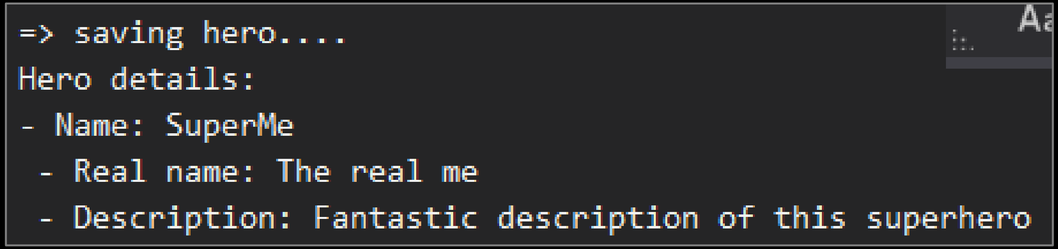


Details provided by: Lies Pinket

SuperMe

The real me

Fantastic description of this superhero



```
=> saving hero....
Hero details:
- Name: SuperMe
 - Real name: The real me
 - Description: Fantastic description of this superhero
```

- Add a RelayCommand property 'ResetCommand' to HeroPageVM
  - This connects to a method ResetAll ( ) – no parameters:
    - ✓ Set the UserName property to "?" + print it to console (!)
    - ✓ Change hero object + print CurrentHero to console (!):

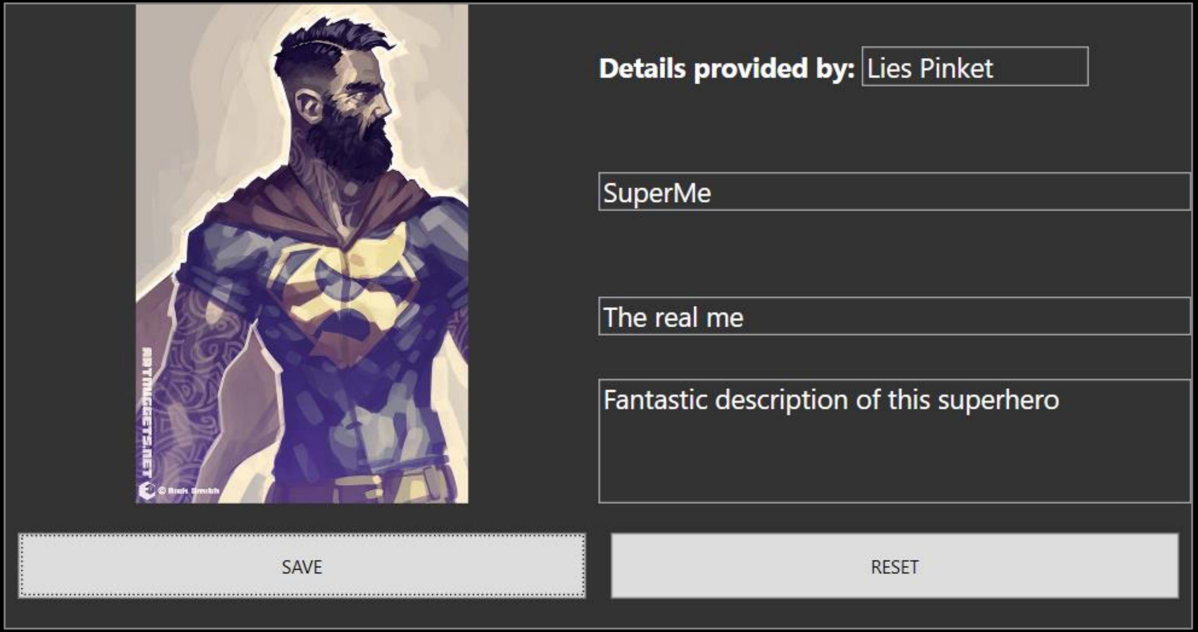
```
CurrentHero.Name = "(no name)";
CurrentHero.RealName = "(no real name)";
CurrentHero.Description = "(this ain't no hero)";
```

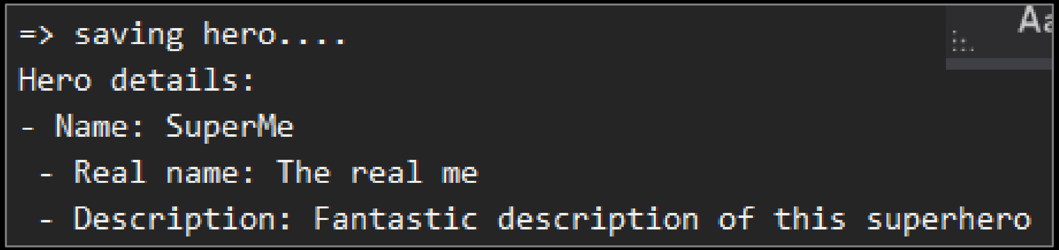
??

What seems to go wrong?

MVVM project

HANDS-ON: ADD SAVE COMMAND





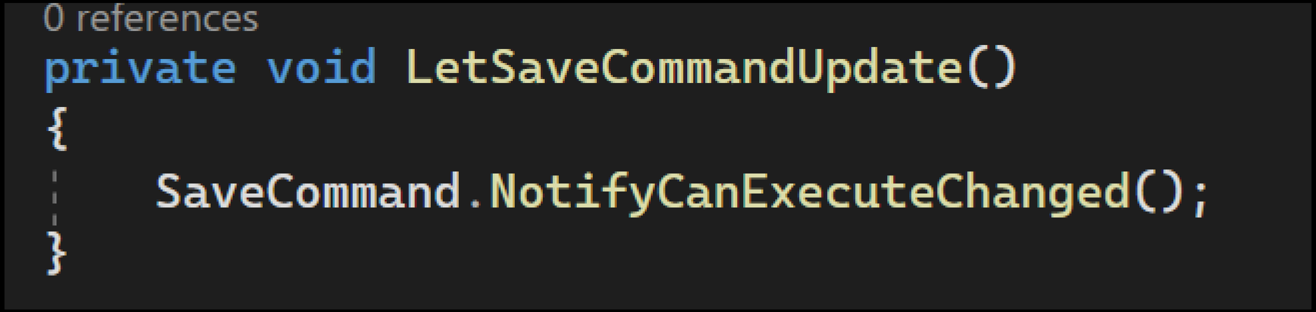
```
=> saving hero....
Hero details:
- Name: SuperMe
 - Real name: The real me
 - Description: Fantastic description of this superhero
```

- Add a RelayCommand property ‘SaveCommand’ to HeroPageVM
  - See previous slides
  - No parameters needed
  - Save should be disabled when Name or RealName are empty
    - Test by setting the default Hero (Real)Name to an empty string.
    - Test by clearing the Name or RealName inputfields and pressing Tab →Nothing happens?

MVVM project

MVVM EXAMPLE: ENABLE / DISABLE

- The SaveCommand doesn't automatically evaluate the CanExecute function when something changes.
- We need to tell it when to re-evaluate it manually using SaveCommand.NotifyCanExecuteChanged()

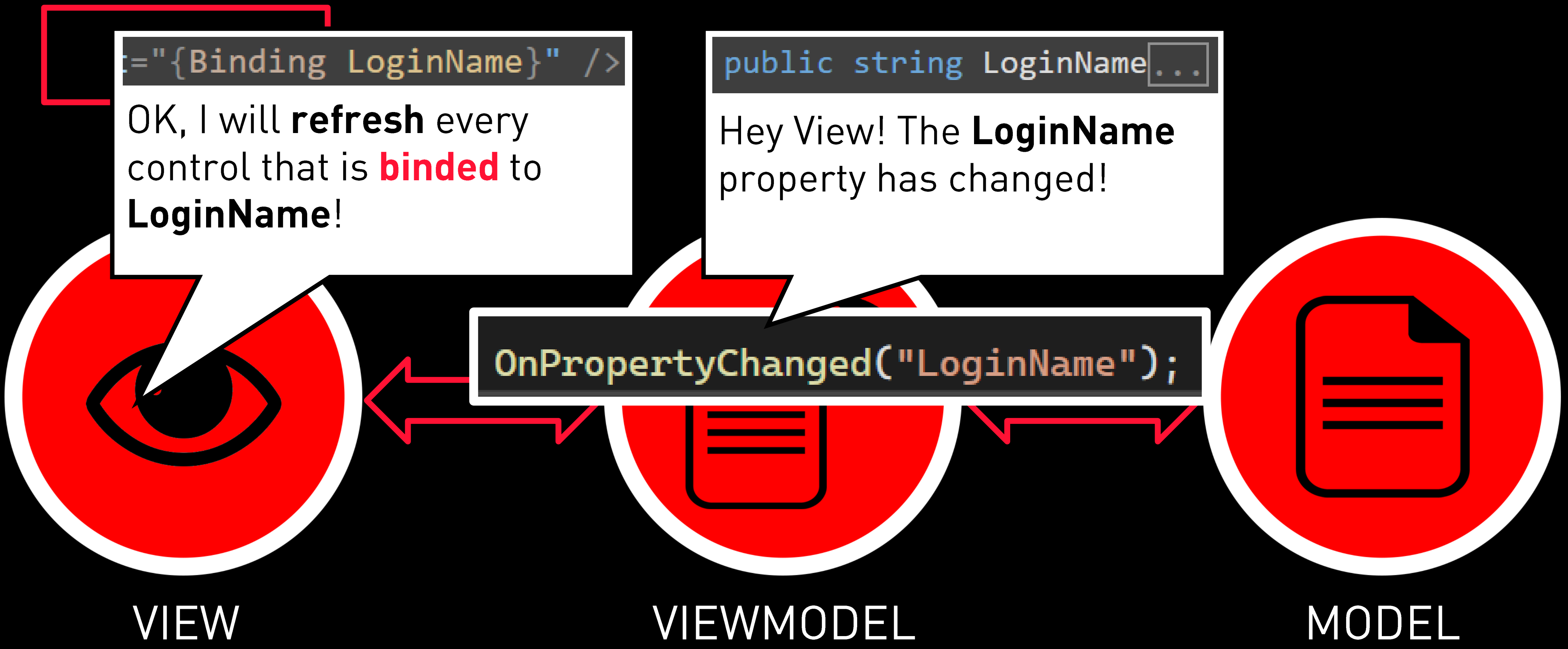


```
0 references
private void LetSaveCommandUpdate()
{
    SaveCommand.NotifyCanExecuteChanged();
}
```

INOTIFYPROPERTYCHANGED

NOTIFY OF CHANGES

MVVM
MODEL VIEW VIEWMODEL



```
t="{Binding LoginName}" />
```

OK, I will **refresh** every control that is **binded** to **LoginName**!

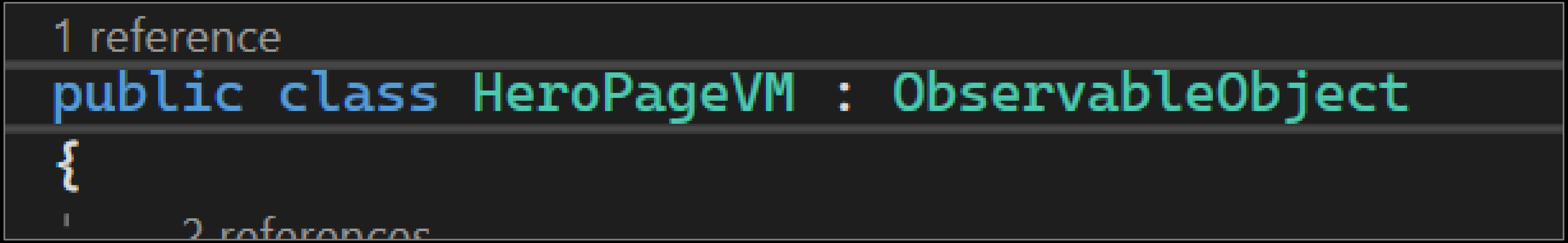
```
public string LoginName...
```

Hey View! The **LoginName** property has changed!

```
OnPropertyChanged("LoginName");
```

raisepropertychanged

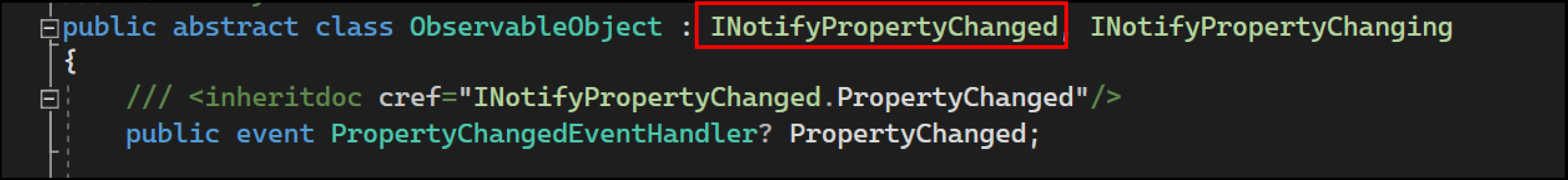
VIEWMODEL → OBSERVABLEOBJECT



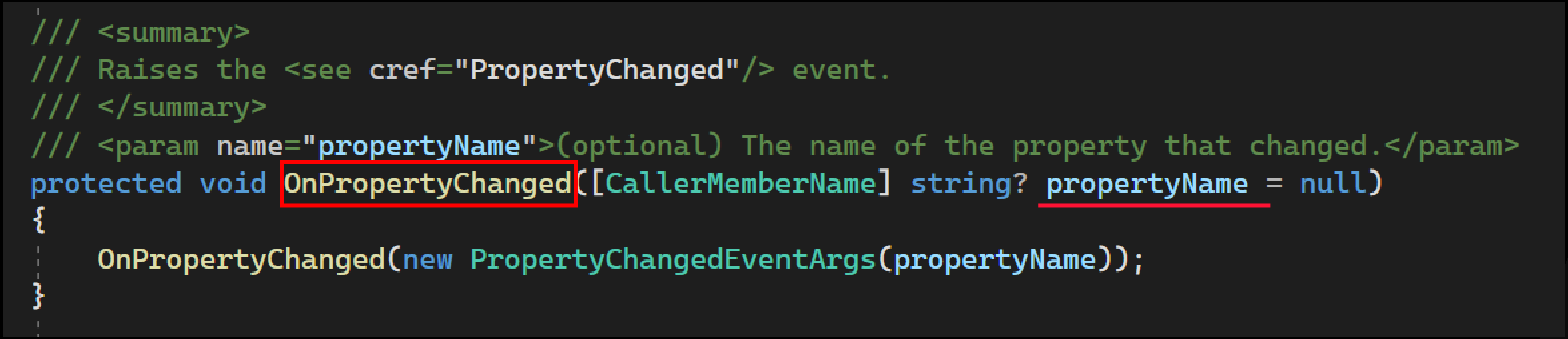
```
1 reference
public class HeroPageVM : ObservableObject
{
```

raisepropertychanged

OBSERVABLEOBJECT → INOTIFYPROPERTYCHANGED



```
public abstract class ObservableObject : INotifyPropertyChanged, INotifyPropertyChanging
{
    /// <inheritdoc cref="INotifyPropertyChanged.PropertyChanged"/>
    public event PropertyChangedEventHandler? PropertyChanged;
```



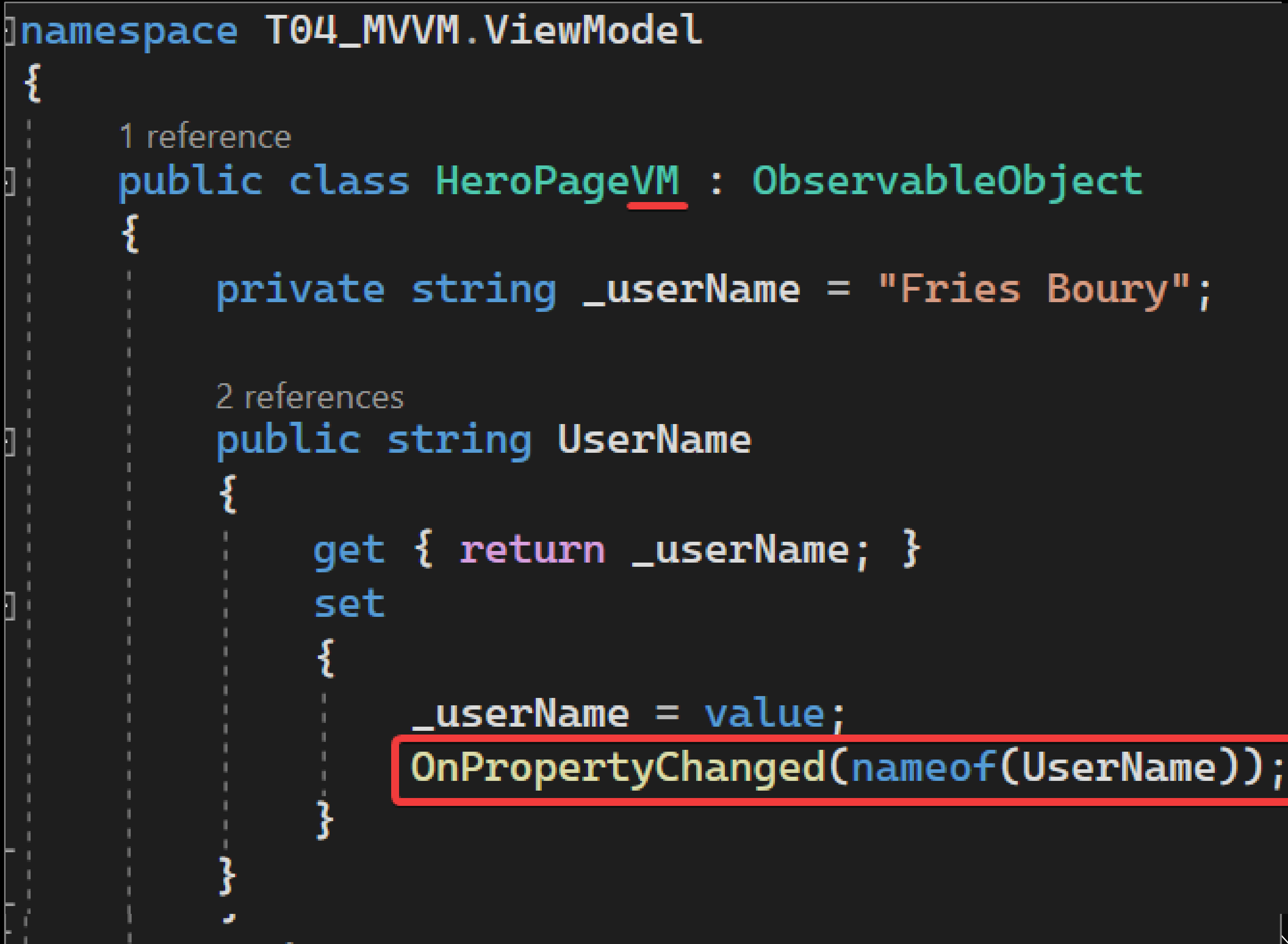
```
/// <summary>
/// Raises the <see cref="PropertyChanged"/> event.
/// </summary>
/// <param name="propertyName">(optional) The name of the property that changed.</param>
protected void OnPropertyChanged([CallerMemberName] string? propertyName = null)
{
    OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
}
```

MVVM project

HANDS-ON: ONPROPERTYCHANGED (1)

- Make sure your UserName property is a full property (_field + get/set)
- Call OnPropertyChanged with your property name:
- Test:
  - Click reset button
  - UserName textbox value should change to "?"

- Do the same for your CurrentHero property
  - Does it help?
  - Why (not)?



```
namespace T04_MVVM.ViewModel
{
        1 reference
        public class HeroPageVM : ObservableObject
        {
            private string _userName = "Fries Boury";

            2 references
            public string UserName
            {
                get { return _userName; }
                set
                {
                    _userName = value;
                    OnPropertyChanged(nameof(UserName));
                }
            }
```

MVVM project

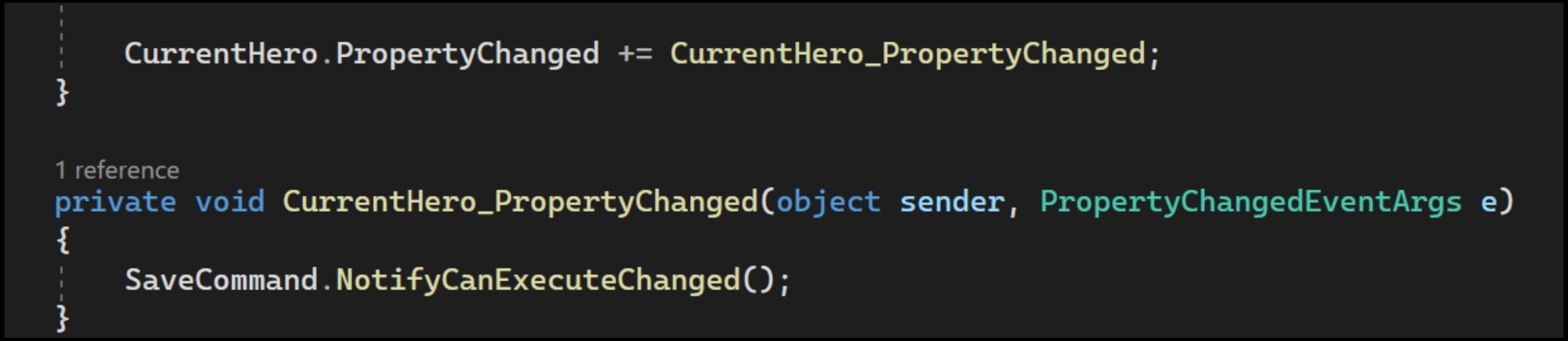
HANDS-ON: ONPROPERTYCHANGED (2)

- Problem: CurrentHero will only invoke OnPropertyChanged if the whole instance changes, eg.: `CurrentHero = new Hero();`
- Solution:
  - ✓ Make the Hero model (class) inherit ObservableObject
    - ➢ Necessary to allow raising a property changed
  - ✓ Call OnPropertyChanged (only) on the properties where needed
- Test: hitting the reset button should now change all textboxes' values

MVVM project

HANDS-ON: ONPROPERTYCHANGED (3)

- Problem: SaveCommand needs to automatically be enabled/disabled based on whether the Hero object has an empty (Real)Name or not.
- Solution:
  - ✓ Register to the PropertyChanged event of the Hero object
  - ✓ When this event fires, Notify the SaveCommand that the CanExecute has changed.



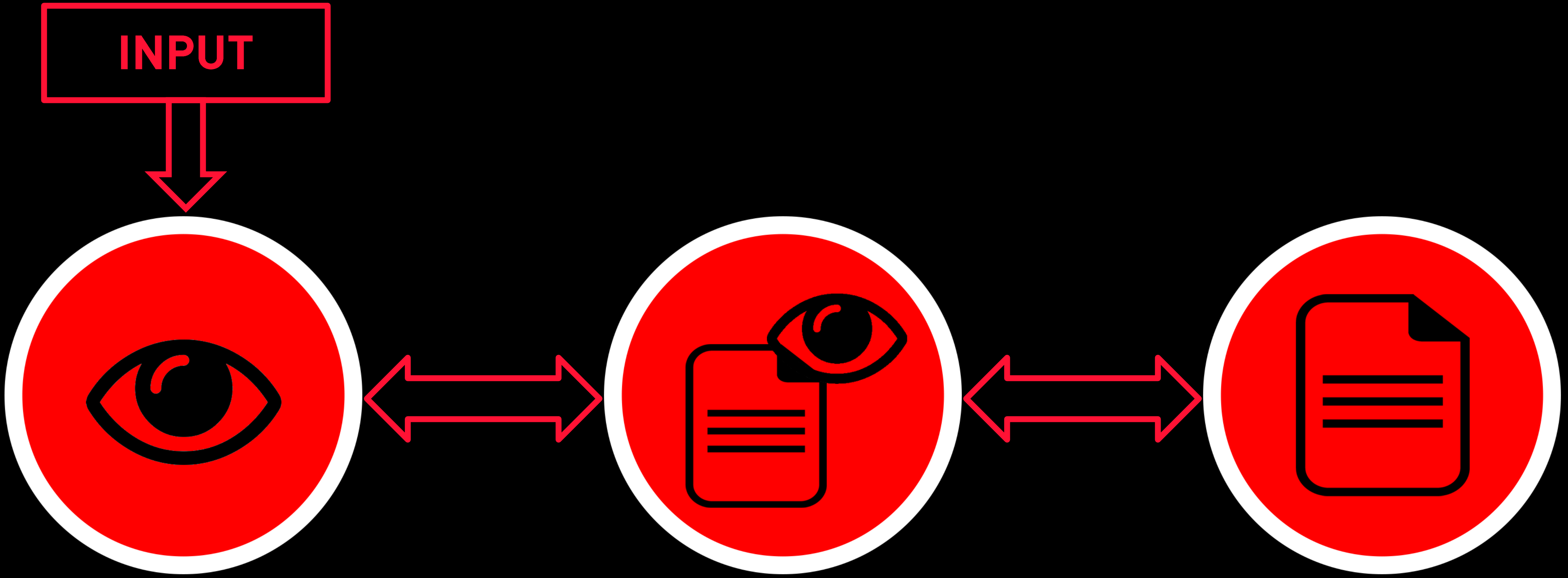
```
    CurrentHero.PropertyChanged += CurrentHero_PropertyChanged;
}

1 reference
private void CurrentHero_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    SaveCommand.NotifyCanExecuteChanged();
}
```

- Test: clearing the RealName or Name textfield should now disable the Save button.

MVVM

MODEL VIEW VIEWMODEL - SUMMARY



VIEW

- ✓ Use Binding to bind to properties
- ✓ Replace Events by Binding to Commands
- ✓ (Almost) no C# code

VIEWMODEL

- ✓ All business logic
- ✓ Use Model for structure
- ✓ OnPropertyChanged to refresh View

MODEL

- ✓ POD (Plain Old Data)
- ✓ Avoid modifying!

LIST<T> TYPE AND MVVM

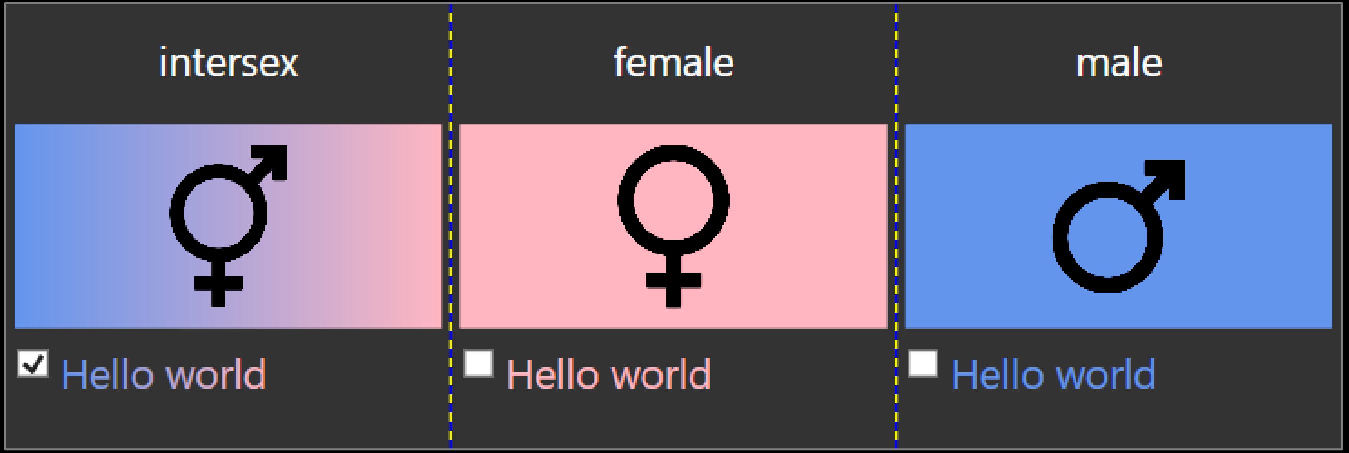
- Changes in a `List<T>` :
  - Adding / removing items in a `List<T>` will **not** notify a property changed!
  - Just as with the Hero object: only if the whole instance changes

- <u>Solution:</u> `ObservableCollection<T>`
  - Automatically calls propertychanged when the items in the collection change
    - Item added
    - Item removed
    - **Not** when the whole property changes (= `new ObservableCollection<T>`);
      in that case you still have to call OnPropertyChanged on the Property **if** necessary
  - Careful!: 'expensive' type,
  - don't just replace every `List<T>` by an ObservableCollection!
  - ONLY USE THIS TYPE IF NECESSARY!

MVVM

WHAT'S NEXT...

- ValueConverters:



- ➢ Everything has `{Binding Gender}` !
  - ➢ Except for "hello world" text
- ➢ Person objects (3) with Gender values:
  - ➢ "intersex", "female", "male"
- ➢ Converted into color, boolean, image

- Navigation:
  - The 'real' MVVM way: ViewModelLocator, dependency injection
  - The 'ToolDev' way: much simplified version because of time limit ☺