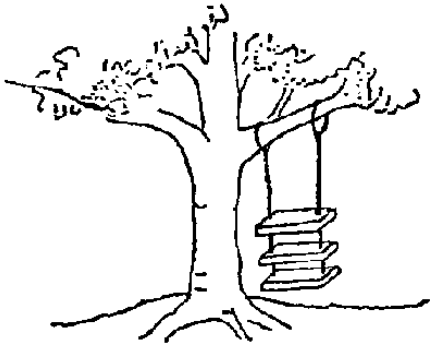
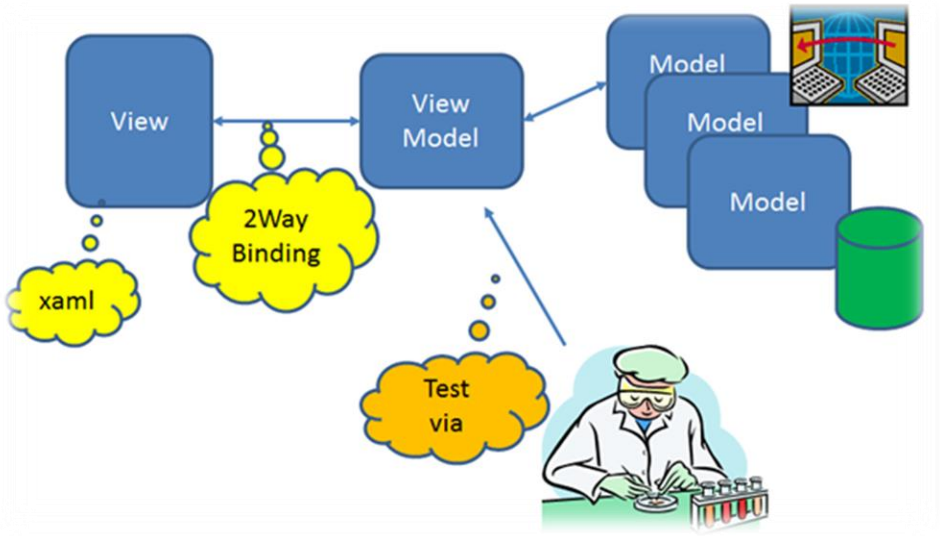
WPF Excercises

Software Entwicklung



Overview

- WPF Basics
  - New Window
  - MessageBox
- WPF Controls
- WPF Panels
- WPF Data Binding
  - Model View ViewModel (MVVM)
- WPF ICommand
  - Relay Command
- WPF Exercises



WPF Data Binding

INotifyPropertyChanged

Binding Mode – UpdateSourceTrigger

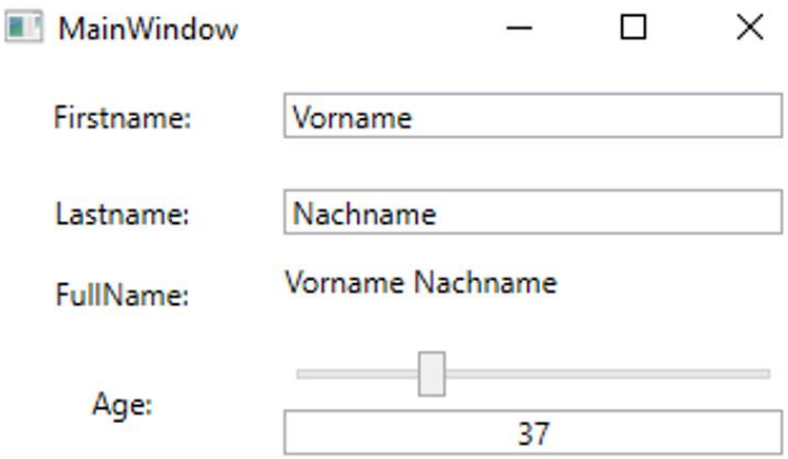
Observable Collection

Exercises

- Personal Details with Full Name Binding & Slider
- Hello Bound World: Mode & UpdateSourceTrigger
- Collection Binding with Employee
- List of Names: Add Name To List

- Create Student & save & show StudentList
- Create Subjects & save & show SubjectList
- SkillList with Progressbar
- Background Color Slider
- PatientList: Add, Delete & Find

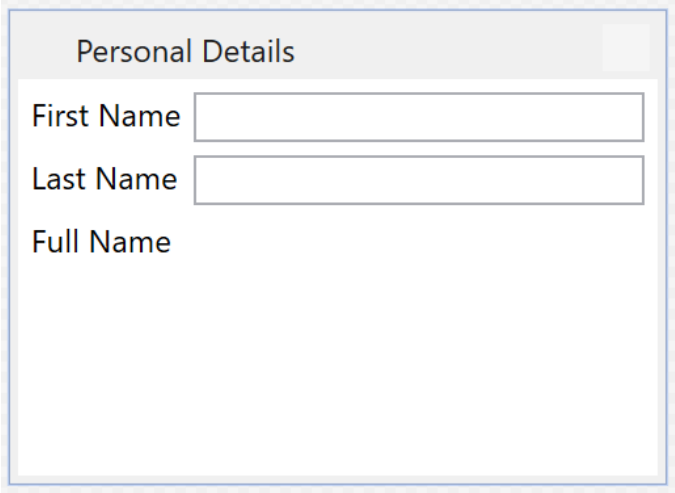
http://www.blackwasp.co.uk/WPFDataBindingFallback.aspx



Personal Details

Personal Detail

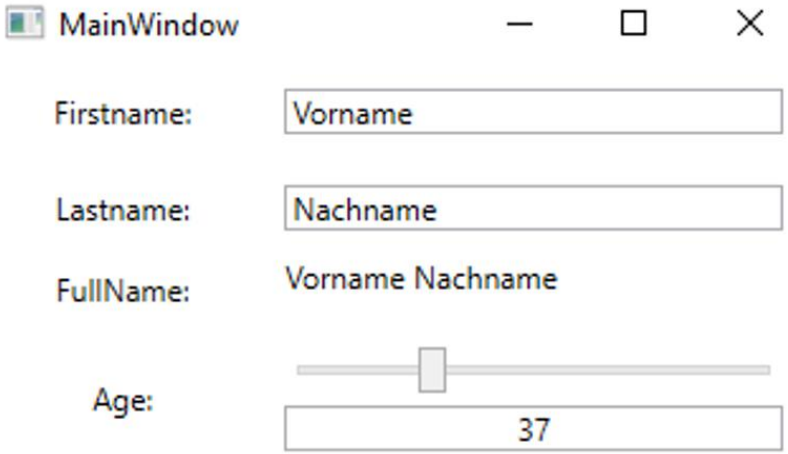
- Create TextBlock and TextBox
  - to Enter FirstName and Last Name
  - show FullName in a TextBlock below




```
<TextBlock>First Name</TextBlock>
<TextBox Grid.Column="1" Margin="5 0 0 5" Text="{Binding FirstName}"/>

<TextBlock Grid.Row="1">Last Name</TextBlock>
<TextBox Grid.Column="1" Grid.Row="1" Margin="5 0 0 5" Text="{Binding LastName}"/>

<TextBlock Grid.Row="2">Full Name</TextBlock>
<TextBlock Grid.Column="1" Grid.Row="2" Margin="5 0 0 5" Text="{Binding FullName}"/>
```

Data Binding

- Add the Age Slider & TextBlock



Firstly, replace the XAML for the first TextBox with the code below:

```
<TextBox Grid.Column="1" Margin="5 0 0 5" Text="{Binding FirstName}"/>
```

For the second TextBox use:

```
<TextBox Grid.Column="1" Grid.Row="1" Margin="5 0 0 5" Text="{Binding LastName}"/>
```

Finally, remove the explicit Paths from the Slider and TextBlock that show the age.

```
<Slider Minimum="16" Maximum="120" Value="{Binding Age}" />
<TextBlock HorizontalAlignment="Center" Text="{Binding Age}"/>
```

Personal Detail

Firstname: Vorname
Lastname: Nachname
FullName: Vorname Nachname
Age: 37

```
public class PersonalDetail : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private string firstname;
    private string lastname;
    private int age;

    public string FirstName...

    public string LastName...

    public int Age...

    public string FullName...

    protected virtual void OnPropertyChanged(string property)
    {
        if(PropertyChanged!= null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(property));
        }
    }
}
```

```
public string FirstName
{
    get { return firstname; }
    set
    {
        firstname = value;
        OnPropertyChanged("FullName");
    }
}
```

```
public int Age
{
    get { return age; }
    set
    {
        age = value;
        OnPropertyChanged("Age");
    }
}
```

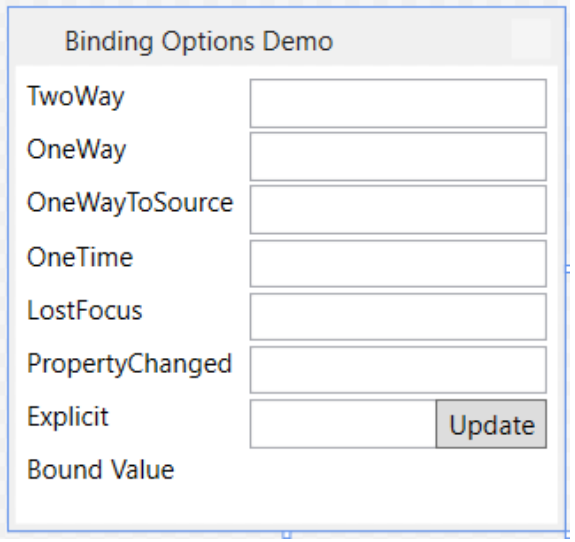
```
using System.ComponentModel;
```

Personal Detail

```
<Window.DataContext>
    <local:PersonalDetail/>
</Window.DataContext>
```

- Set DataContext
- XAML: Add Bindings to each Control

```
<TextBlock TextAlignment="Center" VerticalAlignment="Center">Firstname:</TextBlock>
<TextBox Grid.Column="1" Margin="10" Text="{Binding FirstName}"/>

<TextBlock Grid.Row="1" TextAlignment="Center" VerticalAlignment="Center">Lastname:</TextBlock>
<TextBox Grid.Column="1"  Grid.Row="1"  Margin="10" Text="{Binding LastName}"/>

<TextBlock Grid.Row="2" TextAlignment="Center" VerticalAlignment="Center">FullName:</TextBlock>
<TextBlock Grid.Column="1" Grid.Row="2" Margin="10 0 0 10" Text="{Binding FullName}"/>

<TextBlock Grid.Row="3" Grid.Column="0" TextAlignment="Center" VerticalAlignment="Center">Age:</TextBlock>
<StackPanel Grid.Row="3" Grid.Column="1" >
    <Slider Margin="10 10 10 0" Maximum="130" Value="{Binding Age}"></Slider>
    <TextBox Margin="10 5 10 10" Text="{Binding Age}" TextAlignment="Center"/>
</StackPanel>
```



Hello Bound World

Try the diffrent ways to UpdateSourceTrigger:

http://www.blackwasp.co.uk/WPFBindingOptions.aspx

Create UI

```
<Window x:Class="WPFBindingOptionsDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Binding Options Demo"
        Height="235"
        Width="250"
        ResizeMode="NoResize">
    <Grid Margin="5">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <TextBlock>TwoWay</TextBlock>
        <TextBox Grid.Column="1" Height="22"/>

        <TextBlock Grid.Row="1">OneWay</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="1" Height="22"/>

        <TextBlock Grid.Row="2">OneWayToSource</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="2" Height="22"/>

        <TextBlock Grid.Row="3">OneTime</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="3" Height="22"/>

        <TextBlock Grid.Row="4">LostFocus</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="4" Height="22"/>

        <TextBlock Grid.Row="5">PropertyChanged</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="5" Height="22"/>

        <TextBlock Grid.Row="6">Explicit</TextBlock>
        <TextBox Name="Explicit" Grid.Column="1" Grid.Row="6"
                 Height="22" Margin="0 0 49 0"/>
        <Button Grid.Column="1" Grid.Row="6"
                Height="22" Width="50" HorizontalAlignment="Right"
                Content="Update"/>

        <TextBlock Grid.Row="7">Bound Value</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="7" Text="{Binding Text}"/>
    </Grid>
</Window>
```

TestObject

```
public class TestObject : INotifyPropertyChanged {
    string _text = "Hello, world";

    public string Text {
        get { return _text; }
        set {
            _text = value;
            OnPropertyChanged("Text");
        }
    }
    private void OnPropertyChanged(string propertyName) {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Add Bindings

```
<TextBlock>TwoWay</TextBlock>
<TextBox Grid.Column="1" Height="22" Text="{Binding Text, Mode=TwoWay}"/>

<TextBlock Grid.Row="1">OneWay</TextBlock>
<TextBox Grid.Column="1" Grid.Row="1" Height="22" Text="{Binding Text, Mode=OneWay}"/>

<TextBlock Grid.Row="2">OneWayToSource</TextBlock>
<TextBox Grid.Column="1" Grid.Row="2" Height="22" Text="{Binding Text, Mode=OneWayToSource}"/>

<TextBlock Grid.Row="3">OneTime</TextBlock>
<TextBox Grid.Column="1" Grid.Row="3" Height="22" Text="{Binding Text, Mode=OneTime}"/>

<TextBlock Grid.Row="4">LostFocus</TextBlock>
<TextBox Grid.Column="1" Grid.Row="4" Height="22"
        Text="{Binding Text, Mode=TwoWay,UpdateSourceTrigger=LostFocus}"/>

<TextBlock Grid.Row="5">PropertyChanged</TextBlock>
<TextBox Grid.Column="1" Grid.Row="5" Height="22"
        Text="{Binding Text,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
```

HelloBoundWorldWindow

- Code Behind with Button_Click

```
/// <summary>
/// Interaktionslogik für HelloBoundWorldWindow.xaml
/// </summary>
public partial class HelloBoundWorldWindow : Window
{
    public HelloBoundWorldWindow()
    {
        InitializeComponent();
        this.Show();
        DataContext = new TestObject();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        BindingExpression expr = Explicit.GetBindingExpression(TextBox.TextProperty);
        expr.UpdateSource();
    }
}
```

Bennene die Textbox in der View:

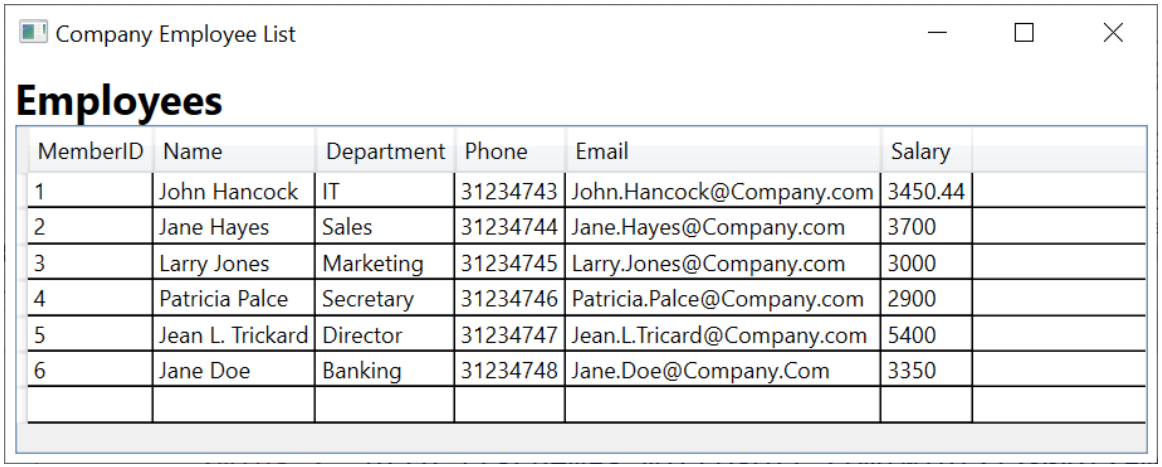
```
<TextBox Name="Explicit" />
```

Setting the Binding Mode

- **TwoWay**. Configures the binding to be bi-directional. Changes made by the user are passed back to the data source and changes in the source update the control. This option is generally used for user input controls.
- **OneWay**. Sets the binding so that changes made in the data source are copied into the bound property of the target control. Updates made by the user are not passed to the data source. This binding mode is generally used for read-only controls, such as TextBlocks.
- **OneWayToSource**. Configures the binding so that changes made by manipulating the control are passed back to the data source. Changes in the data source are not copied into the control.
- **OneTime**. A one-time data binding means that the control's property is set when control is created or when the data context is changed. Further changes to either the property or the data source are not transmitted. This type of binding is generally used for static data or when you wish to display a snapshot of the data at a point in time.
- **Default**. Uses the default binding mode for the property. This value varies according to the control. For example, a TextBlock's Text property defaults to being one-way but a TextBox's Text property uses a two-way binding as standard.

Update Triggers

- The update trigger for a data binding determines when changes made in the control's property are passed back to the data source. They are valid only for one-way and two-way bindings. Four options are available, each defined in the *UpdateSourceTrigger* enumeration.
- **LostFocus**. This mode causes changes in the property of a control to be copied to the data source when the control loses focus. It is useful for controls that receive many updates and the data source only needs to know the end result. For example, it is often unnecessary for a TextBox's Text property to be copied to the source after every key press.
- **PropertyChanged**. If you use this mode, changes to the information in the control are copied to the source immediately. This is useful for controls such as CheckBoxes, where the update should happen when the control is clicked, not when it loses focus.
- **Explicit**. When a data binding's update trigger is set to *Explicit*, changes to the property are not copied automatically. You must call the data binding's *UpdateSource* method.
- **Default**. As with the binding mode, controls use different options for the update trigger. Setting the update trigger to *Default* uses the standard option for the property.



MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400
6	Jane Doe	Banking	31234748	Jane.Doe@Company.Com	3350

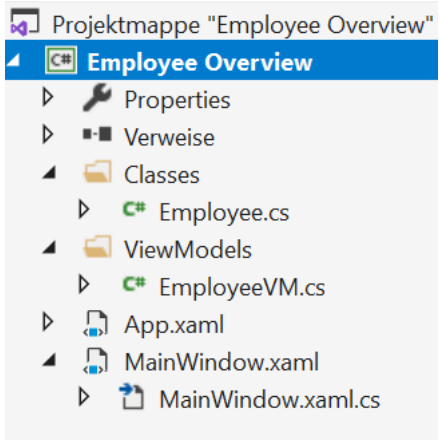
Example Employees

Collections in ListView, ListBox or DataGrid use ObservableCollection

Using MVVM

Class Employee & EmployeeVM

```
public class EmployeeVM : INotifyPropertyChanged
{
    public EmployeeVM()
    {
        Employees = GetEmployeeList();
    }

    ObservableCollection<Employee> GetEmployeeList()...

    private ObservableCollection<Employee> _employees;
    public ObservableCollection<Employee> Employees
    {
        get { return _employees; }
        set
        {
            _employees = value;
            RaiseChange("Employees");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    public void RaiseChange(string PropertyName)...
}
```




```
public class Employee
{
    public int MemberID { get; set; }
    public string Name { get; set; }
    public string Department { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string Salary { get; set; }
}
```

Data Binding im XAML

```
<Window x:Class="Employee_Overview.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VM="clr-namespace:Employee_Overview.ViewModels"
        Title="Company Employee List" Height="250" Width="625"
        Background="CornflowerBlue">

    <Window.DataContext>
        <VM:EmployeeVM/>
    </Window.DataContext>

    <Grid Margin="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <TextBlock Text="Employees" FontSize="22" FontWeight="Bold" Foreground="DarkBlue"/>
        <DataGrid ItemsSource="{Binding Employees}" Grid.Row="1"/>
    </Grid>
</Window>
```

Copy Code

```
//Do not change this class

public class Employee {

        public int MemberID { get; set; }

        public string Name { get; set; }

        public string Department { get; set; }

        public string Phone { get; set; }

        public string Email { get; set; }

        public string Salary { get; set; }

}

//Einzelnen Employee erzeugen und hinzufügen:

 Employee employee = new Employee() {

                MemberID = 6,

                Name = "Jane Doe",

                Department = "Banking",

                Phone = "31234748",

                Email = "Jane.Doe@Company.Com",

                Salary = "3350"

            };

 employees.Add(employee);
```

```
//ObservableCollection Daten erzeugen – copy it to MainWindowVM

ObservableCollection<Employee> employees = new ObservableCollection<Employee>();

            employees.Add(new Employee { MemberID = 1, Name = "John Hancock", Department = "IT", Phone = "31234743", Email = @"John.Hancock@Company.com", Salary = "3450.44" });

            employees.Add(new Employee { MemberID = 2, Name = "Jane Hayes", Department = "Sales", Phone = "31234744", Email = @"Jane.Hayes@Company.com", Salary = "3700" });

            employees.Add(new Employee { MemberID = 3, Name = "Larry Jones", Department = "Marketing", Phone = "31234745", Email = @"Larry.Jones@Company.com", Salary = "3000" });

            employees.Add(new Employee { MemberID = 4, Name = "Patricia Palce", Department = "Secretary", Phone = "31234746", Email = @"Patricia.Palce@Company.com", Salary = "2900" });

            employees.Add(new Employee { MemberID = 5, Name = "Jean L. Trickard", Department = "Director", Phone = "31234747", Email = @"Jean.L.Tricard@Company.com", Salary = "5400" });
```

Show the Selected Item underneath

MainWindow

Employees

MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400

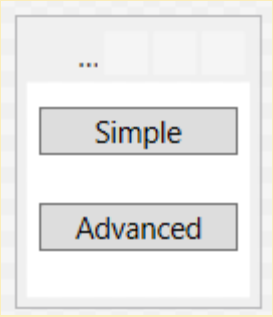
1
John Hancock
IT
John.Hancock@Company.com
31234743
3450.44

```
<TextBlock Text="Employees" FontSize="22" FontWeight="Bold"/>
<DataGrid ItemsSource="{Binding Employees}"
          SelectedItem="{Binding SelectedEmployee}" Grid.Row="1"></DataGrid>
<StackPanel Grid.Row="2">
    <TextBlock Text="{Binding SelectedEmployee.MemberID}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Name}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Department}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Email}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Phone}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Salary}"></TextBlock>
</StackPanel>
```

EmployeeVM with SelectedItem

```
public class EmployeeVM : INotifyPropertyChanged
{
    private ObservableCollection<Employee> _employees;
    private Employee _selectedEmployee;

    private ObservableCollection<Employee> InitializeEmployees()...
    public ObservableCollection<Employee> Employees...

    public Employee SelectedEmployee
    {
        get
        {
            if (_selectedEmployee == null)
                _selectedEmployee = _employees[0];
            return _selectedEmployee;
        }
        set
        {
            _selectedEmployee = value;
            OnPropertyChanged("SelectedEmployee");
        }
    }
```

```
public class Model : INotifyPropertyChanged
{
    #region CurrentName

    public string CurrentName
    {
        get { return mCurrentName; }
        set
        {
            if (value == mCurrentName)
                return;
            mCurrentName = value;
            OnPropertyChanged();
        }
    }
    string mCurrentName;

    #endregion

    public ObservableCollection<string>
    AddedNames { get; } = new ObservableCollection<string>();


    public event PropertyChangedEventHandler PropertyChanged;

    void OnPropertyChanged([CallerMemberName]string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```



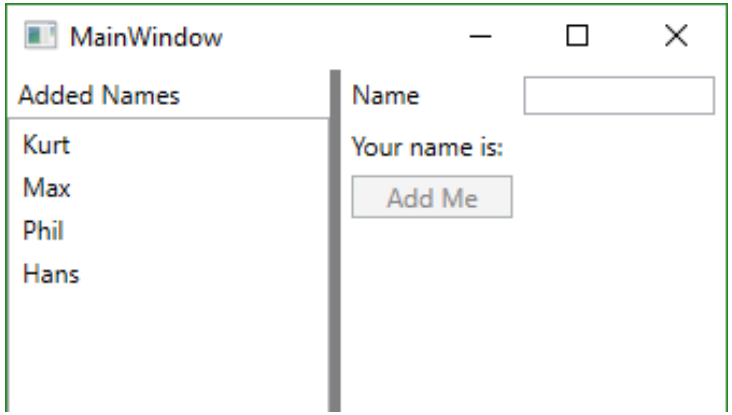
List of Names

Write your own AddCommand

https://www.codeproject.com/Articles/1112919/MVVM-for-beginners

Simple Window

- Create a WPF Application,
- add some names to a list and show it.

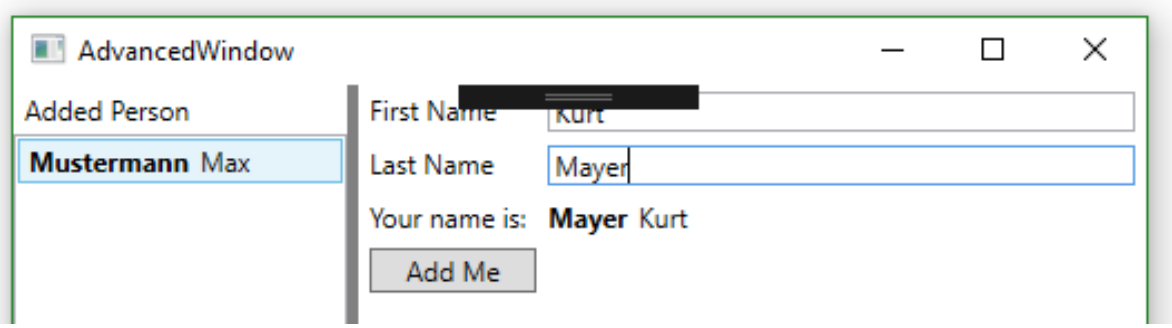



```
class AddNameCommand : ICommand
{
    Model parent;

    public AddNameCommand(Model parent)
    {
        this.parent = parent;
        parent.PropertyChanged += delegate { CanExecuteChanged?.Invoke(this, EventArgs.Empty); };
    }

    public event EventHandler CanExecuteChanged;

    public bool CanExecute(object parameter) { return !string.IsNullOrEmpty(parent.CurrentName); }

    public void Execute(object parameter)
    {
        parent.AddedNames.Add(parent.CurrentName); ;
        parent.CurrentName = null;
    }
}
```

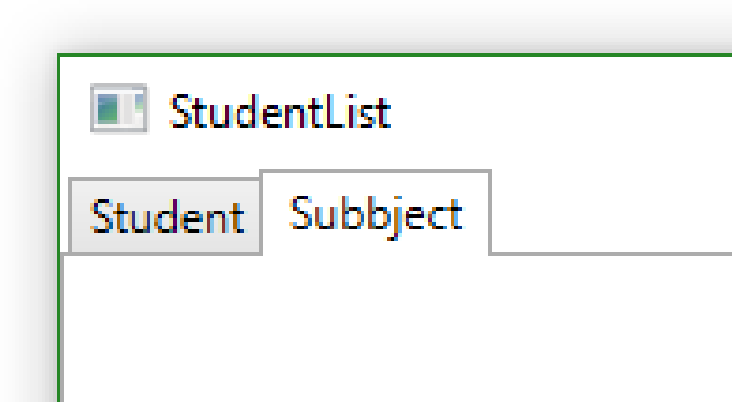
```
<Button Grid.Row="2" Grid.ColumnSpan="2"
HorizontalAlignment="Left" Content="Add Me"
        Command="{Binding AddCommand}"/>
```

Advanced Window

- Add a person with firstname and lastname bold to a listbox.
- Show the name above the button.




```
<DataTemplate x:Key="PersonTemplate">
    <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding LastName}" FontWeight="Bold" Margin="0,0,5,0"/>
        <TextBlock Text="{Binding FirstName}"/>
    </StackPanel>
</DataTemplate>



    <ListBox ItemsSource="{Binding AddedPersons}"

             ItemTemplate="{StaticResource PersonTemplate}">
    </ListBox>
```

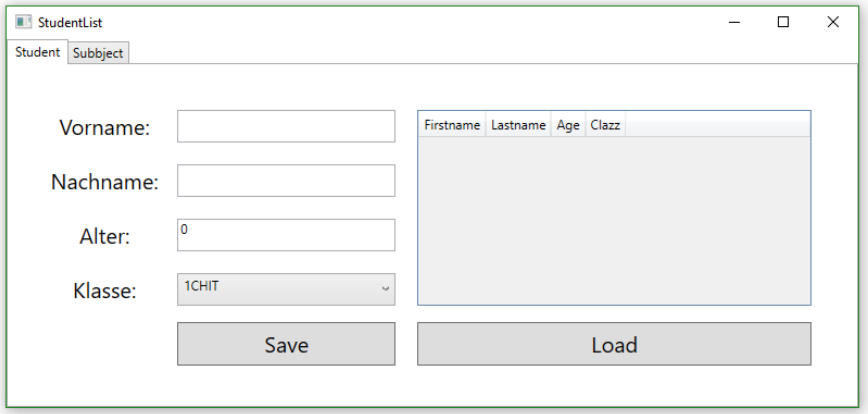


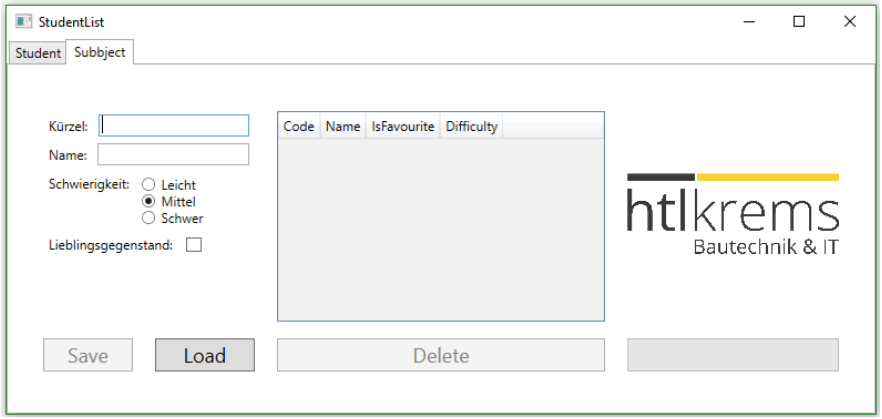
Student & Subject Example

Use a TabControl to manage Students & Subjects

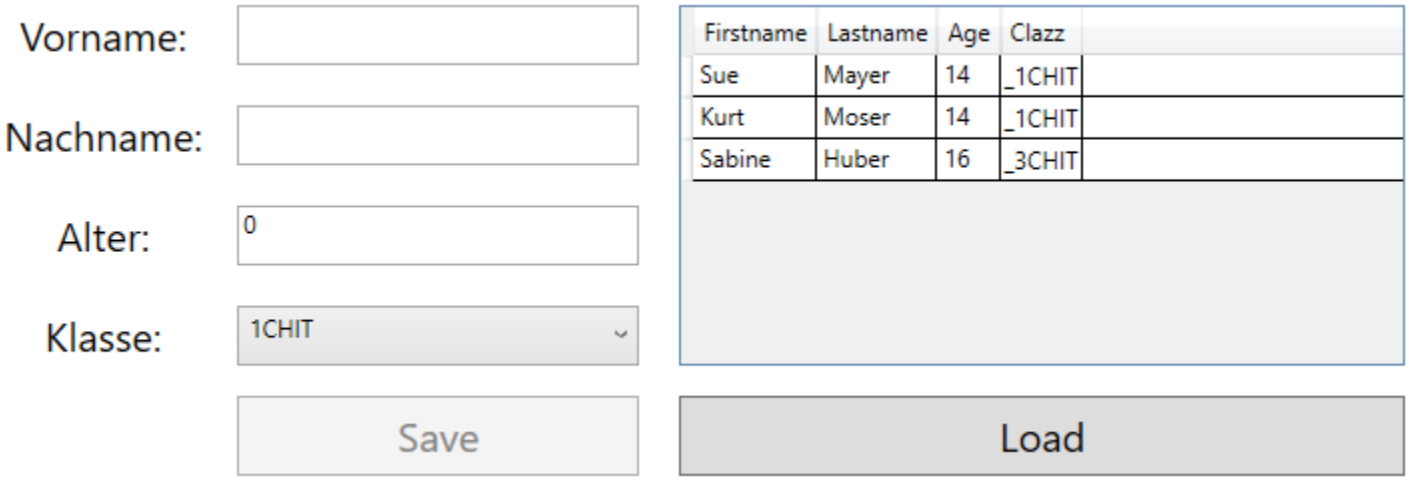
TabControl

- TabItem Student
  - Set the DataContext for TabItem Student to StudentVM
- TabItem Subject
  - Set the DataContext for TabItem Subject to SubjectVM






```
<Grid>
    <TabControl  >
        <TabItem Header="Student" DataContext="StudentViewModel">
            <Grid Margin="20 30" ...>
        </TabItem>
        <TabItem Header="Subbject" DataContext="SubjectViewModel">
            <Grid Margin="20 30"...>
        </TabItem>
    </TabControl>
</Grid>
```



Studenten

Label, TextBox, Button, ComboBox, ListBox

Student

- Create a new Form, enter FirstName, LastName, Age and his/her Class like 1Bhit, 2Bhit, 3Bhit, …
- Use Label and TextBox for the input
- Use an Enum & a ComboBox for the Class

- Save all Students in an ObservableCollection
- Show all Students in a DataGrid

Student – Load from csv file

Firstname	Lastname	Age	Clazz
Sue	Mayer	14	_1CHIT
Kurt	Moser	14	_1CHIT
Sabine	Huber	16	_3CHIT

Load

- Use a ObservableCollection „Students“ in the ViewModel
  - Add a Student
- Save in a csv-file
  - Use a to csv-method in the student
- Load the csv-file
  - Load all students from the csv to a the ObservableCollection „Students“

```
<Button Grid.Column="1" Grid.Row="4" Margin="0 5" FontSize="20"
        Command="{Binding student.AddCommand}">Save</Button>
<DataGrid Grid.Column="2" Grid.RowSpan="4" Margin="20 10" CanUserAddRows="False"
          ItemsSource="{Binding student.StudentCollection, Mode=OneWay}" />
<Button Grid.Column="2" Grid.Row="4" Margin="20 5" FontSize="20"
        Command="{Binding student.LoadCommand}">Load</Button>
```

StudentViewModel - Command

```
public class ViewModel : INotifyPropertyChanged
{

    public ICommand AddCommand { get; private set; }
    public ICommand LoadCommand { get; private set; }

    public ViewModel()
    {

        AddCommand = new AddCommand(this);
        LoadCommand = new LoadCommand(this);
    }
```

LoadCommand

```
public ICommand AddCommand { get; private set; }
public ICommand LoadCommand { get; private set; }

public ViewModel()
{
    AddCommand = new AddCommand(this);
    LoadCommand = new LoadCommand(this);
```

```
class LoadCommand:ICommand
{
    ViewModel parent;
    public event EventHandler CanExecuteChanged;

    public LoadCommand(ViewModel parent)...

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        parent.LoadFromCSV();
    }
}
```

```
<Button Grid.Column="2" Grid.Row="4" Margin="20 5" FontSize="20"
        Command="{Binding student.LoadCommand}">Load</Button>
```

Class Student

```
public enum EClass {_1CHIT, _2CHIT, _3CHIT, _4CHIT, _5CHIT }

public class Student
{
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    public int Age { get; set; } = 0;
    public EClass Clazz { get; set; }

    public Student(string firstname, string lastname, int age, EClass clazz)
    {
        this.Firstname = firstname;
        this.Lastname = lastname;
        this.Age = age;
        this.Clazz = clazz;
    }
    public Student(){  }
    public string ToCsv()...
    public void LoadFromCSVLine()...
}
```

- Properties & Constructor
- ToCsv / ReadCsv

Access a csv file

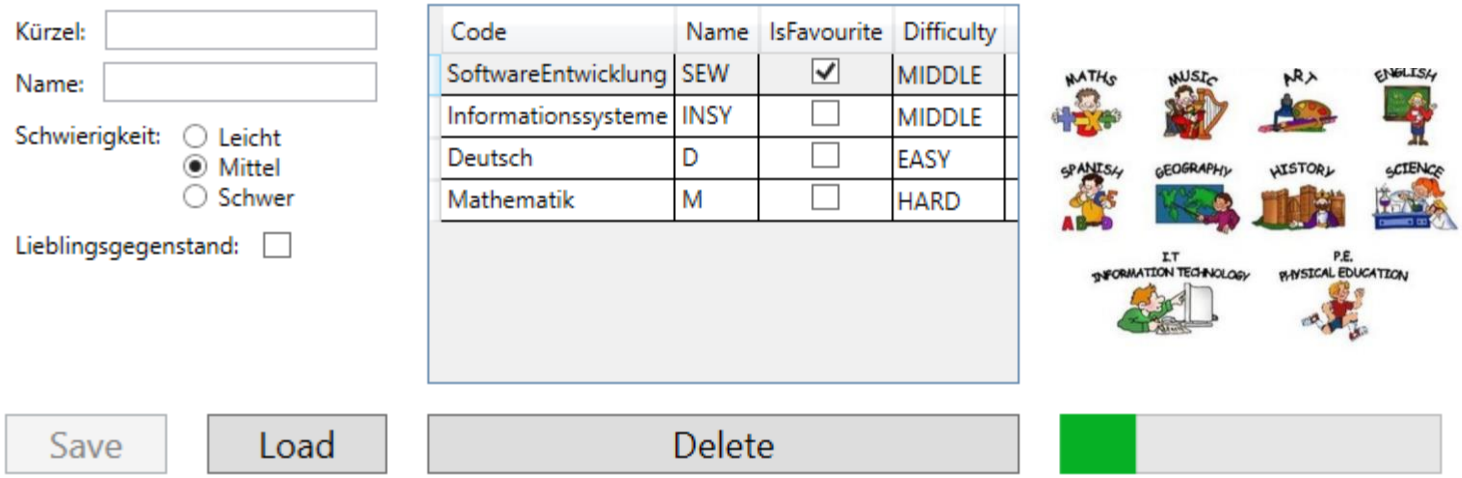
```
public void Save(string path, Student s)
{
    StreamWriter sw = new StreamWriter("students.csv", true);
    sw.WriteLine(s.ToCSV());
    sw.Close();
}

public List<Student> Read(string path)
{
    StreamReader sr = new StreamReader(path);
    List<Student> students = new List<Student>();
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        string[] splittedLine = line.Split(';');
        students.Add(new Student(splittedLine[0],
            splittedLine[1],
            Convert.ToInt32(splittedLine[2]),
            (Classes)Enum.Parse(typeof(Classes),
            splittedLine[3])));
    }
    sr.Close();
    return students;
}
```

Add File.Exits(Path) condition to CanExecute to the Load Method,

can only be executed, if the File exists

Student ViewModel

```
public class StudentViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public ObservableCollection<Student> StudentCollection { get; private set; }
    private Student currentStudent;
    public string Path { get; set; } = "Students.csv";

    public ICommand AddCommand { get; private set; }
    public ICommand LoadCommand { get; private set; }

    public StudentViewModel()...

    public string CurrentFirstname...

    public string CurrentLastname...

    public int CurrentAge...

    public int CurrentClazz...

    public Student CurrentStudent...

    private void OnPropertyChanged(string Property)...

    public void SaveInCSV()...

    public void LoadFromCSV()...
}
```



Manage Subjects

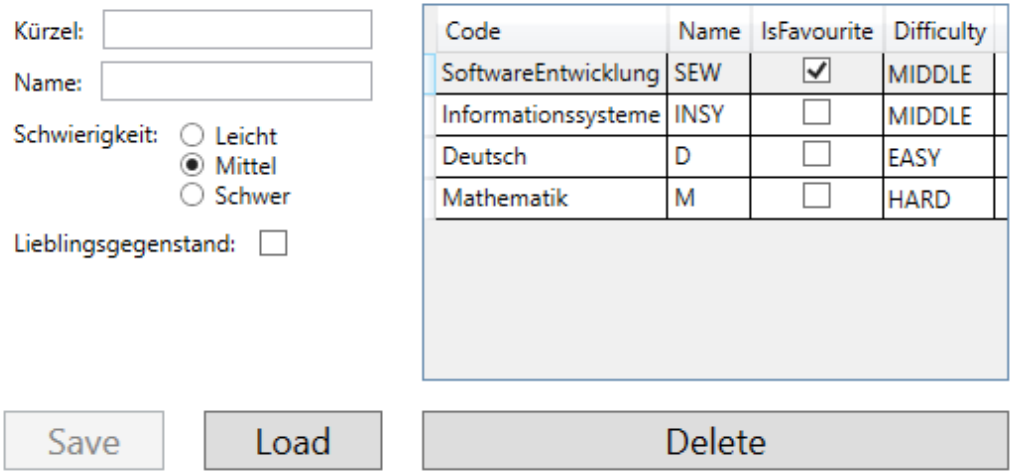
DataGrid, RadioButton, CheckBox, PictureBox

Subject

```
public enum EDifficulty { EASY, MIDDLE, HARD }

public class Subject
{
    public string Code { get; set; }
    public string Name { get; set; }
    public bool IsFavourite { get; set; }
    public EDifficulty Difficulty { get; set; }

    public Subject()
    {
        Code = "";
        Name = "";
        IsFavourite = false;
        Difficulty = EDifficulty.MIDDLE;
    }

    public string ToCsv()...
    public void ReadFromCSV()...
}
```

ObservableCollection Subjects

- Save Subject to a ObservableCollection
- Delete selected Subject from Collection



- Save all Subjects to csv file
- Load all Subjects from csv file

Add Picture & ProgressBar

- PictureBox: SchoolLogo

- Add a ProgressBar
  - Amount of saved Subjects (max 20)

Read Subjects

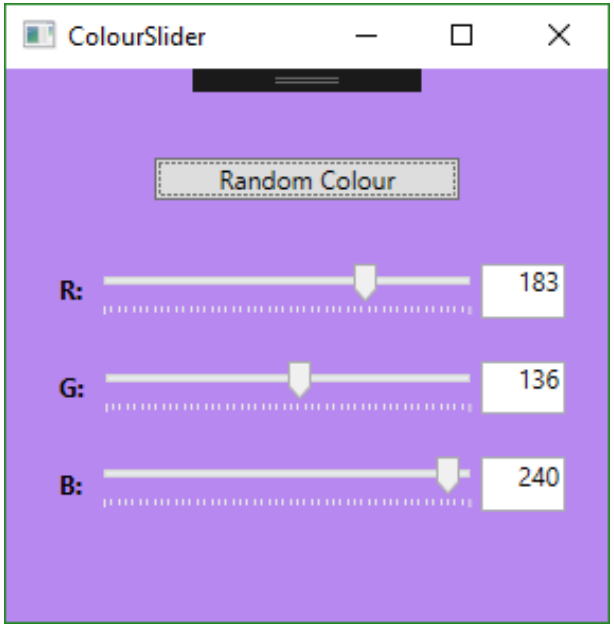
```
public static List<Subject> ReadSubjects(string path)
{
    List<Subject> subjects = new List<Subject>();
    StreamReader sr = new StreamReader(path);
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        string[] splittedLine = line.Split(';');
        subjects.Add(new Subject(splittedLine[0],
            splittedLine[1],
            (EDifficulty)Enum.Parse(typeof(EDifficulty),
            splittedLine[2]),
            Convert.ToBoolean(splittedLine[3])));
    }
    sr.Close();
    return subjects;

}
```

Save Subjects

```
public static void WriteSubjects(List<Subject> subjects, string path) {
    StreamWriter sw = new StreamWriter(path);
    foreach (Subject subject in subjects) {
        sw.WriteLine(subject.ToCSV());
    }
    sw.Close();
}

private string ToCSV() {
    return $"{this.Contraction};{this.Name};{(int)this.Difficulty};{this.FavoriteSubject}";
}

public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    sb.Append($"{Name} ({Contraction}) is {Difficulty.ToString().ToLower()} and ");
    if (!FavoriteSubject) { sb.Append("not ");}
    sb.Append("one of my favorite subjects");
    return sb.ToString();
}
```

SubjectViewModel

```
public class SubjectViewModel:INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private ObservableCollection<Subject> subjectCollection;
    private Subject current;
    public ICommand SaveCommand { get; set; }
    public ICommand DeleteCommand { get; set; }
    public ICommand LoadCommand { get; set; }
    private int selectedIndex;
    public string Path { get; set; }

    public SubjectViewModel()...
    private void OnPropertyChanged([CallerMemberName]string Property = null)...
    public ObservableCollection<Subject> SubjectCollection...
    public string CurrentCode...
    public string CurrentName...
    public bool CurrentFavourite...
    public bool CurrentEasy...
    public bool CurrentMiddle...
    public bool CurrentHard...
    public Subject Current...
    public void AddCurrent()...
    public int CollectionCount...
    public int SelectedIndex...
    public void SaveAsCSV()...
    public void LoadFromCSV()...
    public void RemoveAt(int index)...
}
```



BackgroundColor Sliders

Create 3 Sliders to set the Background-Color

Xaml – Slider for RedValue

- Add 3 Slider to the GUI

```
    Title="ColourSlider" Height="300" Width="300">
<Window.DataContext>
    <local:SliderVM></local:SliderVM>
</Window.DataContext>
<StackPanel Margin="10" VerticalAlignment="Center">
    <Button x:Name="button" Content="Random Colour" Margin="60,0,60,20"
            Command="{Binding RandomColour}" />
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="R:"/>
        <TextBox Text="{Binding RedValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />
        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding RedValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_red" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
```

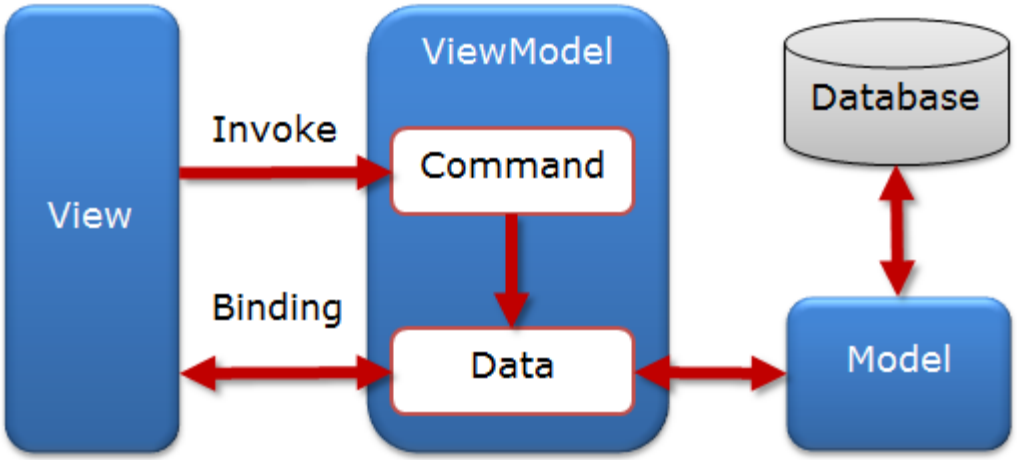
Slider for Green & Blue Values

```
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="G:"/>
        <TextBox Text="{Binding GreenValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />

        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding GreenValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_green" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="B:"/>
        <TextBox Text="{Binding BlueValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />
        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding BlueValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_blue" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
</StackPanel>
```

SliderViewModel

```
class SliderVM:INotifyPropertyChanged
{
    private int redValue;
    public int RedValue...
    private int greenValue;
    public int GreenValue...

    private int blueValue;
    public int BlueValue...

    public RelayCommand RandomColour
    {
        get
        {
            return new RelayCommand(
                o =>
                {
                    Random rnd = new Random();
                    RedValue = rnd.Next(1, 255);
                    GreenValue = rnd.Next(1, 255);
                    BlueValue = rnd.Next(1, 255);
                },
                o => true
            );
        }
    }
```

- Use BaseVM instead of IProperyNotifyChanged
- Set DataContext
- Bind the Properties and the Command to the UI



WPF AViewModel & ICommand

Use a BaseClass for all ViewModel-Classes
Use a RelayCommand, which implements ICommand, instantiate RelayCommand as you need it

AViewModel to Copy

```
abstract class ANotifyPropertyChanged : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;

    public void OnPropertyChanged([CallerMemberName] string property = null)
        => PropertyChanged(this, new PropertyChangedEventArgs(property));
}
```

```
//To Copy
 abstract class ANotifyPropertyChanged : INotifyPropertyChanged {
        public event PropertyChangedEventHandler PropertyChanged;
        public void OnPropertyChanged([CallerMemberName] string property = null)
            => PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
```

```
public event PropertyChangedEventHandler PropertyChanged;

[NotifyPropertyChangedInvocator]
2 Verweise
protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

ConcreteCommand

- Write an AddPerson Method
- Create a PersonAddCommand

PersonsVM

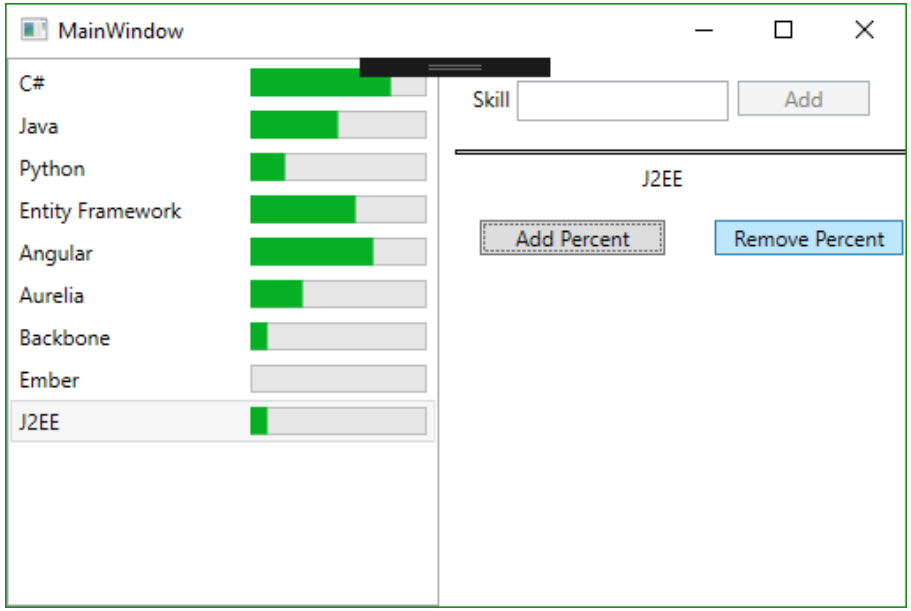
```
public void AddPerson(Person p)
{
    People.Add(new PersonVM(p));
}

public ICommand PersonAddCommand
{
    get
    {
        return new PersonAddCommand(this);
    }
}
```

```
class PersonAddCommand : ICommand
{
    private PersonsVM personsVM;
    public PersonAddCommand(PersonsVM personsVM)
        { this.personsVM = personsVM; }

    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        personsVM.AddPerson(new Person());
    }
}
```

```
<Button Command="{Binding PersonAddCommand}" />
```

RelayCommand to Copy

```
public class RelayCommand : ICommand    {
       private Predicate<object> canExecute;
       private Action<object> execute;
       public event EventHandler CanExecuteChanged {
           add => CommandManager.RequerySuggested += value;
           remove => CommandManager.RequerySuggested -= value;
       }
       public RelayCommand(Action<object> execute, Predicate<object> canExecute){
           this.canExecute = canExecute;
           this.execute = execute;
       }
       public bool CanExecute(object parameter) => canExecute(parameter);
       public void Execute(object parameter)    => execute(parameter);
   }
```



Skill Manager

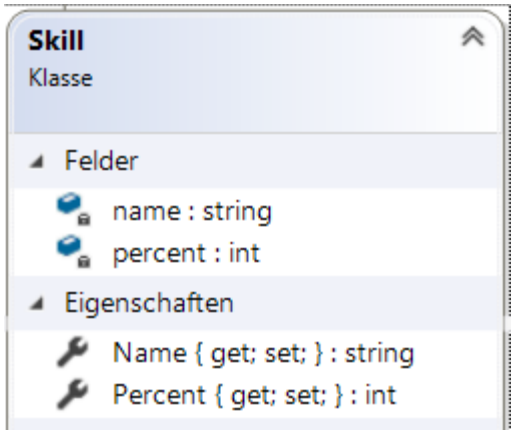
ListBox & Progressbar

Add a Skill to the SkillList (with 0% knowledge)

Increase or Decrease (10%) the percent

Skill & SkillViewModel

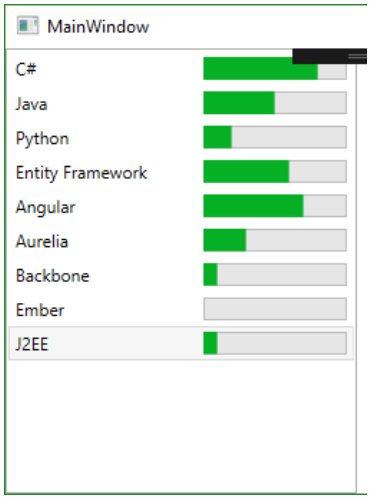
```
private Skill skill;
public Skill Skill...
private Skill selectedSkill=null;
public Skill SelectedSkill...
public ObservableCollection<Skill> Skills ...
public SkillVM()...
public SkillVM(List<Skill> skills)...

public void AddSkills(List<Skill>skills)...
public ICommand AddSkillCommand...
private void AddSkill()...
public ICommand AddPercentageCommand...
private void AddPercentage()...
public ICommand RemovePercentageCommand...
private void RemovePercentage()...
```



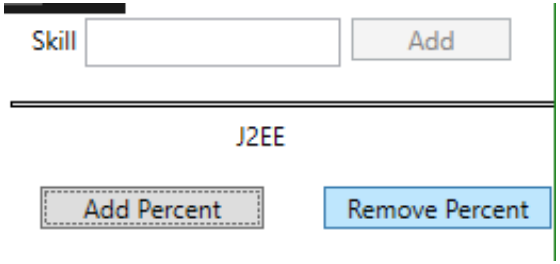

```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
```

Binding a ListBox to a SkillList

```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
<Grid>
    <ListBox Name="lb_skills"  HorizontalContentAlignment="Stretch"
             Margin="0,0,264.4,-0.2"
             ItemsSource="{Binding Skills,UpdateSourceTrigger=PropertyChanged}"
             SelectedItem="{Binding SelectedSkill,Mode=TwoWay}" >
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Grid Margin="0,2">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="100" />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Name}" />
                    <ProgressBar Grid.Column="1" Minimum="0" Maximum="100" Value="{Binding Percent}" />
                </Grid>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
```



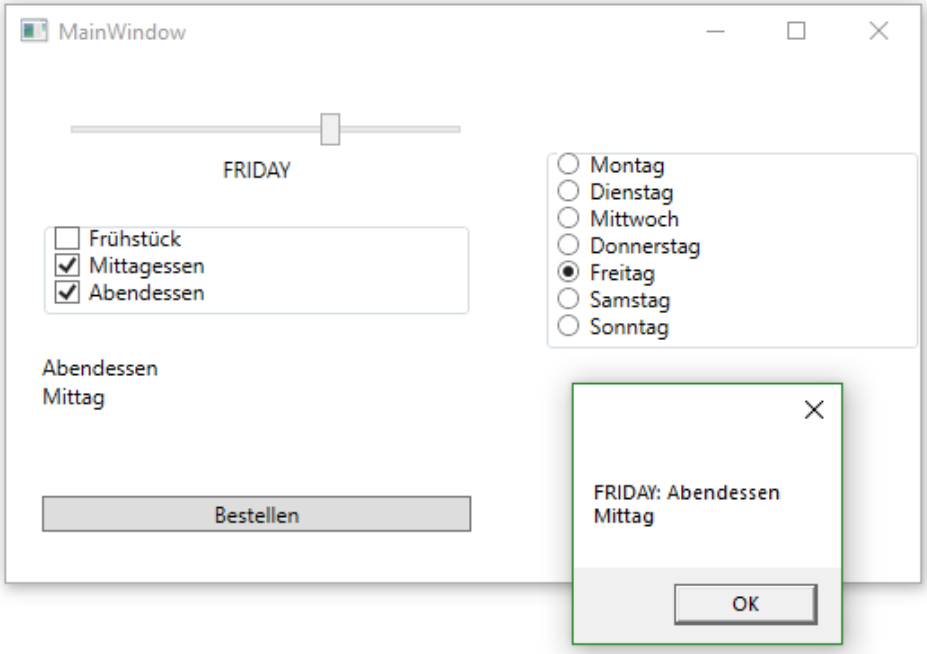
Binding: Label, TextBox & Button




```
<Label Content="Skill" HorizontalAlignment="Left" Margin="259,10,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="289,13,0,0" TextWrapping="Wrap"
        Text="{Binding Skill.Name,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"
        VerticalAlignment="Top" Width="120"/>
<Button Content="Add" HorizontalAlignment="Left" Margin="414,13,0,0"
       VerticalAlignment="Top" Width="75"
       Command="{Binding AddSkillCommand}"/>
<Rectangle Fill="#FFF4F4F5" HorizontalAlignment="Left" Height="3" Margin="254,52,0,0"
          Stroke="Black" VerticalAlignment="Top" Width="264"/>
<TextBlock HorizontalAlignment="Left" Margin="360,60,0,0" TextWrapping="Wrap"
          Text="{Binding SelectedSkill.Name}" VerticalAlignment="Top"/>
<Button Content="Add Percent" HorizontalAlignment="Left" Margin="268,92,0,0"
       VerticalAlignment="Top" Width="105"
       Command="{Binding AddPercentageCommand}"/>
<Button Content="Remove Percent" HorizontalAlignment="Left" Margin="401,92,0,0"
       VerticalAlignment="Top" Width="107"
       Command="{Binding RemovePercentageCommand}" />
```

Add Data…

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        List<Skill> skills = new List<Skill>()
        {
            new Skill() {Name = "C#", Percent = 80},
            new Skill() {Name = "Java", Percent = 50},
            new Skill() {Name = "Python", Percent = 20},
            new Skill() {Name = "Entity Framework", Percent = 60},
            new Skill() {Name = "Angular", Percent = 70},
            new Skill() {Name = "Aurelia", Percent = 30},
            new Skill() {Name = "Backbone", Percent = 10},
            new Skill() {Name = "Ember", Percent = 0},
        };
        var skillVm = DataContext as SkillVM;
        skillVm.AddSkills(skills);
    }
}
```



Order Meal EnumConverter Example

Create an enum EWeekday

… bind it to a Slider & some RadioButtons

Order 1 to 3 Meals a Weekday,
show the order in a MessageBox

Binding RadioButton on Enum

- Enum EWeekdayw in VM:

```
private EWeekdays eweekday;

public EWeekdays Weekday {
    get { return eweekday; }
    set {
        eweekday = value;
        OnPropertyChanged();
    }
}
```

- Set a Value Converter

```
<Grid.Resources>
    <local:EnumBooleanConverter x:Key="EnumBooleanConverter" />
</Grid.Resources>
```

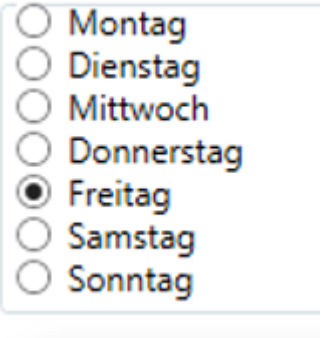
- Binding the RadioButtons

```
<RadioButton Content="Montag" GroupName="weekdays"
             IsChecked="{Binding Path=Weekday,
                         Converter={StaticResource EnumBooleanConverter},
                         ConverterParameter={x:Static local:EWeekdays.MONDAY}}"  />
```

Enum ValueConverter

```
class EnumBooleanConverter : IValueConverter
{
    1-Verweis
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        //checks if Selection from RadioButtonCheckBoxVM has
        //the same value as the ConverterParameter. Returns true or false

        return (EWeekdays)value == (EWeekdays)parameter;
        //return ((Enum)value).HasFlag((Enum)parameter);

    }

    1-Verweis
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        //If the radiobutton is checked, it returns the ConverterParameter
        return value.Equals(true) ? parameter : Binding.DoNothing;
    }
}
```

Bind all RadioButtons

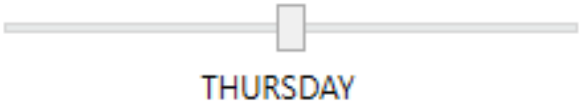



```
<Grid.Resources>
    <local:EnumBooleanConverter x:Key="EnumBooleanConverter" />
</Grid.Resources>
```

```
<GroupBox>
    <StackPanel >
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.MONDAY}}" Content="Montag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.TUESDAY}}" Content="Dienstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.WEDNESDAY}}" Content="Mittwoch" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.THURSDAY}}" Content="Donnerstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.FRIDAY}}" Content="Freitag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.SATURDAY}}" Content="Samstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.SUNDAY}}" Content="Sonntag" GroupName="weekdays" />
    </StackPanel>
</GroupBox>
```

Slider & Enum

- Slider




```
public class SliderVM:AViewModel
{
    private EWeekdays eweekday;

    public EWeekdays EWeekday
    {
        get
        {
            return eweekday;
        }
        set
        {
            eweekday = value;
            CallPropertyChanged("EWeekday");
        }
    }
    public SliderVM()
    {
        eweekday = EWeekdays.FR;
    }
}
```

- with Converter

```
public class EnumIntConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (EWeekdays)System.Convert.ToInt32(value);
    }
}
```

XAML with Slider & Enum Binding

- EnumIntConverter

```
    <Window.DataContext>
        <local:SliderVM></local:SliderVM>
    </Window.DataContext>
    <Grid>
        <Grid.Resources>
            <local:EnumIntConverter x:Key="EnumIntConverter"></local:EnumIntConverter>
        </Grid.Resources>
        <Slider IsSnapToTickEnabled="True" HorizontalAlignment="Left"
                Margin="10,10,0,0" VerticalAlignment="Top" Width="497"
                Minimum="0" Maximum="6"
            Value="{Binding
                        Path=EWeekday,
                        Converter={StaticResource EnumIntConverter},
                        Mode=TwoWay}"
                TickFrequency="1"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="10,33,0,0" TextWrapping="Wrap"
                 Text="{Binding Path=EWeekday}" VerticalAlignment="Top" Width="497"/>
    </Grid>
</Window>
```

Example - PatientList

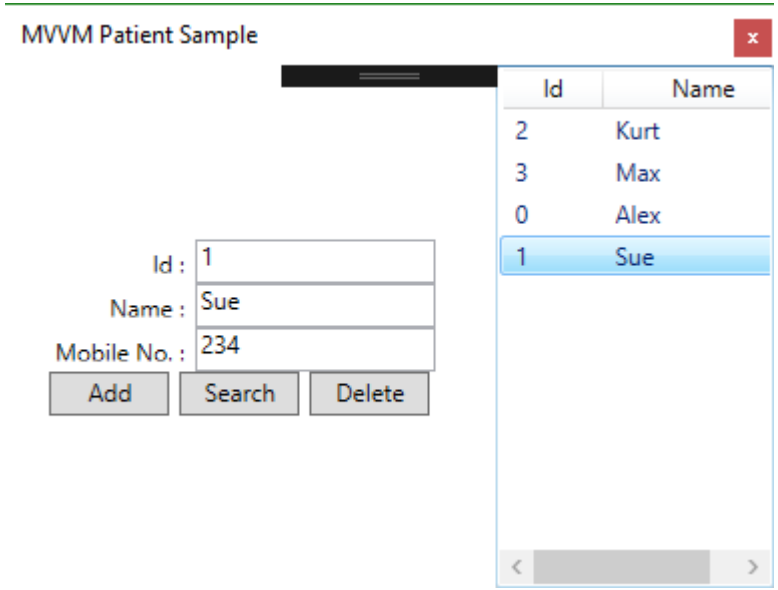
https://www.codeproject.com/Articles/435478/MVVM-Demo

Use Databinding and Command Binding
with BaseVM and ConcreteCommand Classes.

PatientList

- Write your own PatientList:
  - Create a Patient -> add to list
  - Remove a Patient -> selected from list
  - Search for a Patient by Id

- Use Data Binding
  & Command Binding



Patient & PatientVM

```
public class Patient
{
    /// <summary>
    /// Gets or Sets Unique integer ID for the Patient
    /// </summary>
    8 Verweise
    public int Id { get; set; }

    /// <summary>
    /// Gets or Sets Name of the Patient
    /// </summary>
    5 Verweise
    public string Name { get; set; }

    /// <summary>
    /// Gets or Sets MobileNumber of the Patient
    /// </summary>
    5 Verweise
    public Int64 MobileNumber { get; set; }
}
```

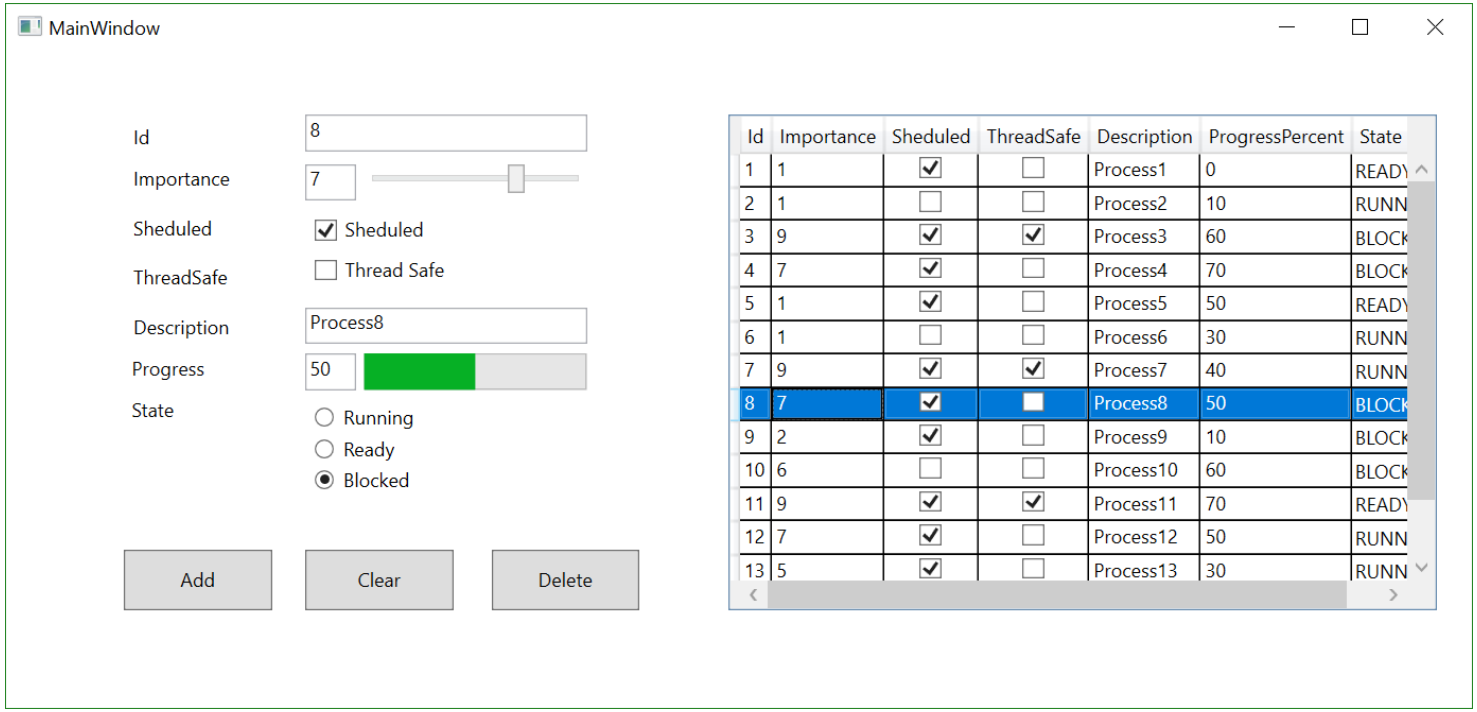
```
public class PatientDetailViewModel: INotifyPropertyChange
{                                   : BaseViewModel

    private readonly Patient patient;

    public string Name
    {
        get { return patient.Name; }
        set
        {
            patient.Name = value;
            OnPropertyChanged("Name");
        }
    }

    /// <summary>
    /// Gets or Sets Patient MobileNumber. Ready
    /// Impelments INotifyPropertyChanged which e
    /// </summary>
    5 Verweise
    public Int64 MobileNumber
    {
        get { return patient.MobileNumber; }
        set
        {
            patient.MobileNumber = value;
            OnPropertyChanged("MobileNumber");
        }
    }
```

PatientManager uses PatientRepo

```
/// <summary> Implements Business Logic related to Patient.
3 Verweise
public class PatientManager
{
    readonly PatientRepository patientRepository;

    /// <summary> Initialises all the private variables
    1-Verweis
    public PatientManager()...

    /// <summary> Add Patient
    1-Verweis
    public bool Add(Patient patient)...

    /// <summary> Remove Patient
    1-Verweis
    public bool Remove(int id)...

    /// <summary> Search for a patient
    1-Verweis
    public Patient Search(int id)...
}
```

PatientRepo

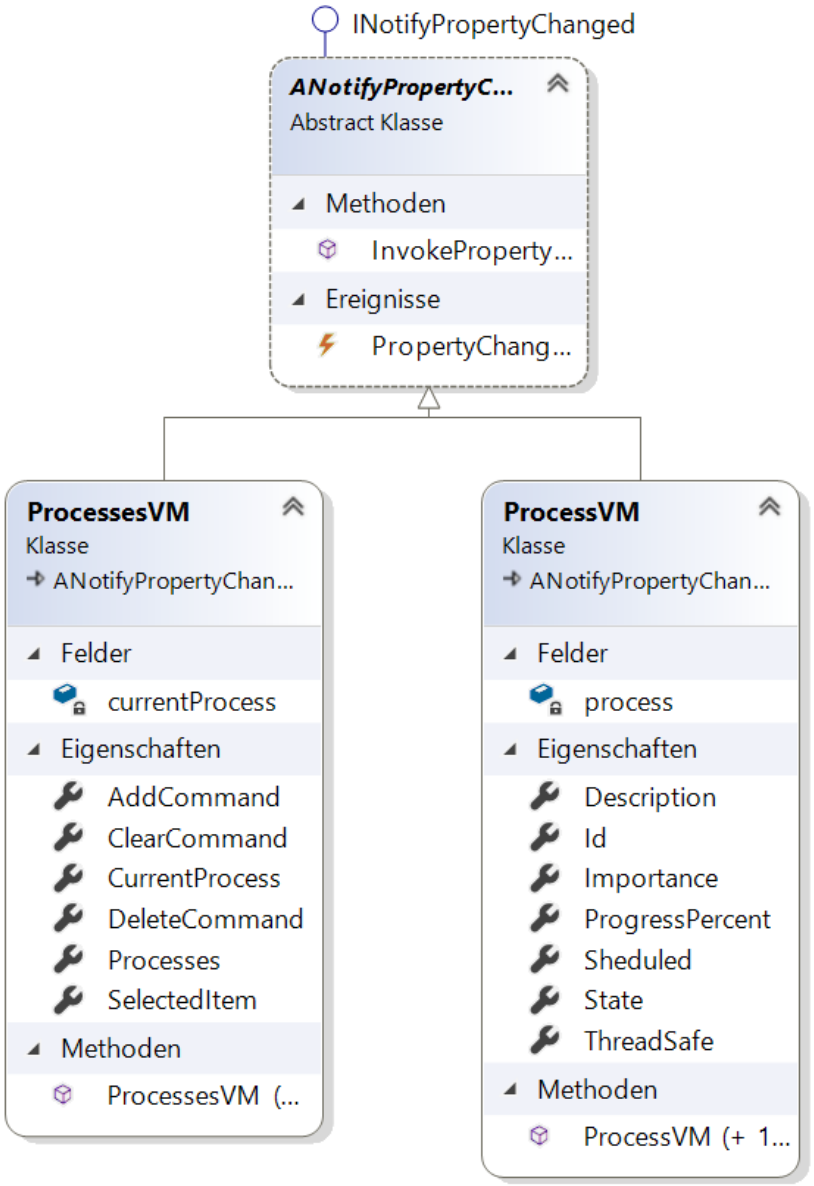
```
/// <summary>
/// PatientRepository provides mechanism to interact with storage.
/// Uses a temp collection for storage. Can be extended to store in DB.
/// </summary>
2 Verweise
public class PatientRepository
{
    //Maintains the patient collection locally
    private static List<Patient> patients = new List<Patient>();

    /// <summary> Add a patient
    1-Verweis
    internal void Add(Patient patient)...

    /// <summary> Remove a patient based on
    1-Verweis
    internal void Remove(Patient patient)...

    /// <summary> Search for the patient with Patient ID
    3 Verweise
    internal Patient Search(int id)...

    /// <summary> Search for the patient ID in the collection and return the Index
    1-Verweis
    private int GetIndex(int id)...
}
```

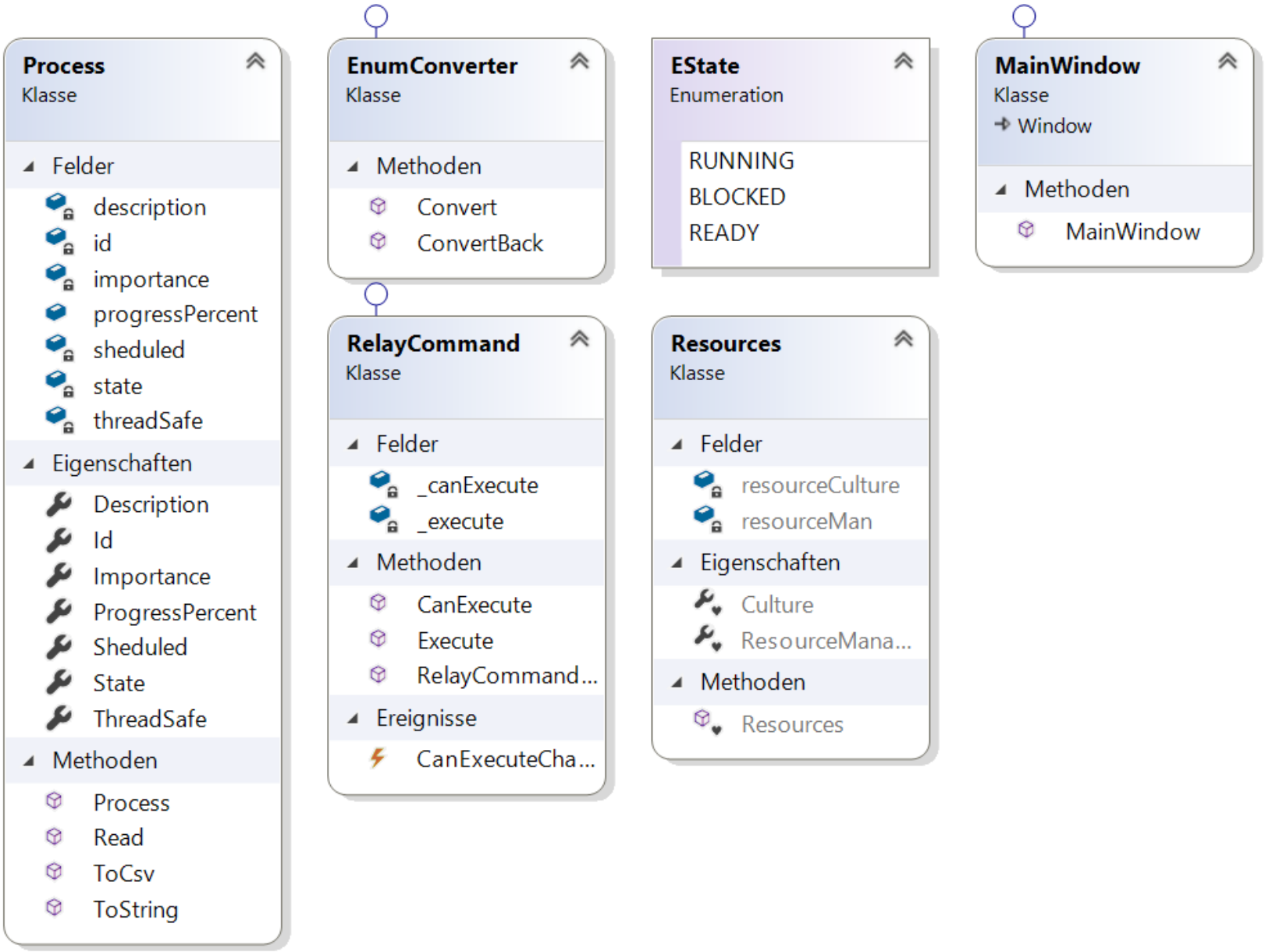


Process WPF

Load Processes from CSV, Select and Edit, Clear Fields & Add new Process to List

ProcessVM & ProcessesVM




Process
Klasse
Felder
description
id
importance
progressPercent
sheduled
state
threadSafe
Eigenschaften
Description
Id
Importance
ProgressPercent
Sheduled
State
ThreadSafe
Methoden
Process
Read
ToCsv
ToString
EnumConverter
Klasse
Methoden
Convert
ConvertBack
EState
Enumeration
RUNNING
BLOCKED
READY
MainWindow
Klasse
Window
Methoden
MainWindow
RelayCommand
Klasse
Felder
_canExecute
_execute
Methoden
CanExecute
Execute
RelayCommand...
Ereignisse
CanExecuteCha...
Resources
Klasse
Felder
resourceCulture
resourceMan
Eigenschaften
Culture
ResourceMana...
Methoden
Resources

Process

```
public class Process
   {
       private int id;

       //Value [1-10]
       private int importance;
       private bool sheduled;
       private string description;
       private bool threadSafe;
       public int progressPercent;
       private EState state;


       public int Id { get => id; set => id = value; }
       public int Importance { get => importance; set => importance = value; }
       public bool Sheduled { get => sheduled; set => sheduled = value; }
       public bool ThreadSafe { get => threadSafe; set => threadSafe = value; }
       public string Description { get => description; set => description = value; }
       public int ProgressPercent { get => progressPercent; set => progressPercent = value; }
       public EState State { get => state; set => state = value; }

       public Process(int id, int importance, bool sheduled,
           bool threadsafe, string desc, int progress, EState state)
       {
           this.id = id;
           this.importance = importance;
           this.sheduled = sheduled;
           this.threadSafe = threadsafe;
           this.description = desc;
           this.progressPercent = progress;
           this.state = state;
       }
```

CSV Read Write

```
public string ToCsv()
{
    return $"{Id};{Importance};{Sheduled};{ThreadSafe};{Description};{ProgressPercent};{State};" ;
}
1-Verweis
public static List<Process> Read()
{
    List<Process> processes = new List<Process>();
    StreamReader sr = new StreamReader("process.csv");
    string line;
    string[] words;


    while ((line = sr.ReadLine()) != null)
    {
        words = line.Split(';');

        processes.Add(new Process(Convert.ToInt32(words[0]),
            Convert.ToInt32(words[1]), Convert.ToBoolean(words[2]),
            Convert.ToBoolean(words[3]),
            words[4], Convert.ToInt32(words[5]),
            (EState)Enum.Parse(typeof(EState), words[6])));
    }
    return processes;
}
```

XAML

Copy Code

```
<Grid>
        <Label Content="Id" HorizontalAlignment="Left" Margin="75,41,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,40,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="177"/>

        <Label Content="Importance" HorizontalAlignment="Left" Margin="75,67,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,71,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="32"/>
        <Slider  HorizontalAlignment="Left" Margin="225,71,0,0" VerticalAlignment="Top" Width="140" Height="22"/>


        <Label Content="ThreadSafe" HorizontalAlignment="Left" Margin="75,129,0,0" VerticalAlignment="Top"/>
        <CheckBox Content="Thread Safe" HorizontalAlignment="Left" Margin="193,130,0,0" VerticalAlignment="Top"/>
        <Label Content="Sheduled" HorizontalAlignment="Left" Margin="75,98,0,0" VerticalAlignment="Top"/>
        <CheckBox Content="Sheduled" HorizontalAlignment="Left" Margin="193,105,0,0" VerticalAlignment="Top"/>

        <Label Content="Description" HorizontalAlignment="Left" Margin="75,160,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,161,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="177"/>

        <Label HorizontalAlignment="Left" Margin="95,234,0,0" VerticalAlignment="Top"/>

        <Label Content="Progress" HorizontalAlignment="Left" Margin="74,186,0,0" VerticalAlignment="Top"/>
        <ProgressBar HorizontalAlignment="Left" Height="23" Margin="225,190,0,0" VerticalAlignment="Top" Width="140"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,190,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="32"/>

        <Label Content="State" HorizontalAlignment="Left" Margin="74,212,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Running" HorizontalAlignment="Left" Margin="193,223,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Ready" HorizontalAlignment="Left" Margin="193,243,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Blocked" HorizontalAlignment="Left" Margin="193,262,0,0" VerticalAlignment="Top"/>

        <Button Content="Add" HorizontalAlignment="Left" Height="38" Margin="74,313,0,0" VerticalAlignment="Top" Width="93"/>
        <Button Content="Clear" HorizontalAlignment="Left" Height="38" Margin="188,313,0,0" VerticalAlignment="Top" Width="93" />
        <Button Content="Delete" HorizontalAlignment="Left" Height="38" Margin="305,313,0,0" VerticalAlignment="Top" Width="93"/>
        <DataGrid HorizontalAlignment="Left" Height="311" Margin="454,40,0,0" VerticalAlignment="Top" Width="445" />

    </Grid>
```

Solution

```
class ProcessVM : ANotifyPropertyChanged
{
    private Process process;

    public int Id{
        get => this.process.Id;
        set { this.process.Id = value; InvokePropertyChanged(); }
    }
    public int Importance{
        get => this.process.Importance;
        set { this.process.Importance = value; InvokePropertyChanged();}
    }
    public bool Sheduled{
        get => this.process.Sheduled;
        set{ this.process.Sheduled = value; InvokePropertyChanged();}
    }
    public bool ThreadSafe{
        get => this.process.ThreadSafe;
        set{ this.process.ThreadSafe = value; InvokePropertyChanged(); }
    }
    public string Description
    {
        get => this.process.Description;
        set{this.process.Description = value;InvokePropertyChanged();}
    }
    public int ProgressPercent{
        get => this.process.ProgressPercent;
        set{this.process.ProgressPercent = value;InvokePropertyChanged();}
    }
    public EState State{
        get => this.process.State;
        set{this.process.State = value;InvokePropertyChanged();}
    }

    public ProcessVM() : this(  new Process(0, 0, false, false, null, 0, EState.BLOCKED)) { }
    public ProcessVM(Process process){
        this.process = process;
    }
}
```

ProcessesVM

```
class ProcessesVM : ANotifyPropertyChanged
{
    public ObservableCollection<ProcessVM> Processes { get; private set; }

    public RelayCommand AddCommand...
    public RelayCommand ClearCommand...
    public RelayCommand DeleteCommand...

    public ProcessVM SelectedItem...
    private ProcessVM currentProcess = new ProcessVM();
    public ProcessVM CurrentProcess...

    public ProcessesVM() : this(Process.Read()) { }
    public ProcessesVM(IEnumerable<Process> processes)...
}
```

Selected Item vs CurrentProcess

```
public ProcessVM SelectedItem
{
    get => Processes.Contains(CurrentProcess) ? CurrentProcess : null;
    set => CurrentProcess = value == null ? new ProcessVM() : value;
}
private ProcessVM currentProcess = new ProcessVM();
public ProcessVM CurrentProcess
{
    get => this.currentProcess;
    set
    {
        this.currentProcess = value;
        InvokePropertyChanged();
        InvokePropertyChanged("SelectedItem");
    }
}
```

Change one Item of a list, the OC doesn‘t get that something changed, therefore use a OC<T> where T:INotifyPropertyChanged to recognize the change

Solution

```
abstract class ANotifyPropertyChanged : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void InvokePropertyChanged([CallerMemberName] string property = null)
        => PropertyChanged(this, new PropertyChangedEventArgs(property));
}
```

```
class ProcessesVM : ANotifyPropertyChanged
{
    public ObservableCollection<ProcessVM> Processes { get; private set; }

    public RelayCommand AddCommand
        => new RelayCommand(
            o => {
                Processes.Add(CurrentProcess);
                SelectedItem = null;
            }
        );
    public RelayCommand ClearCommand
        => new RelayCommand(
            o => SelectedItem = null
        );
    public RelayCommand DeleteCommand
        => new RelayCommand(
            o => Processes.Remove(CurrentProcess)
        );
```

```
    public ProcessVM SelectedItem
    {
        get => Processes.Contains(CurrentProcess) ? CurrentProcess : null;
        set => CurrentProcess = value == null ? new ProcessVM() : value;
    }
    private ProcessVM currentProcess = new ProcessVM();
    public ProcessVM CurrentProcess
    {
        get => this.currentProcess;
        set
        {
            this.currentProcess = value;
            InvokePropertyChanged();
            InvokePropertyChanged("SelectedItem");
        }
    }

    public ProcessesVM() : this(Process.Read()) { }
    public ProcessesVM(IEnumerable<Process> processes)
    {
        Processes = new ObservableCollection<ProcessVM>(
            processes.Select(p => new ProcessVM(p))
        );
    }
}
```

BoolEnumConverter

```
public class EnumConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        return ((EState)value) == ((EState)parameter);
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if ((bool)value)
            return (EState)parameter;
        else
            return Binding.DoNothing;
    }
}
```

Binding UI Element

```
<Label Content="Id" HorizontalAlignment="Left" Margin="75,41,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,40,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Id}" VerticalAlignment="Top" Width="177"/>

<Label Content="Importance" HorizontalAlignment="Left" Margin="75,67,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,71,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Importance}" VerticalAlignment="Top" Width="32"/>
<Slider HorizontalAlignment="Left" Margin="225,71,0,0" VerticalAlignment="Top" Width="140" Height="22"
       Value="{Binding CurrentProcess.Importance}"/>


<Label Content="ThreadSafe" HorizontalAlignment="Left" Margin="75,129,0,0" VerticalAlignment="Top"/>
<CheckBox Content="Thread Safe" HorizontalAlignment="Left" Margin="193,130,0,0" VerticalAlignment="Top"
         IsChecked="{Binding CurrentProcess.ThreadSafe}"/>
<Label Content="Sheduled" HorizontalAlignment="Left" Margin="75,98,0,0" VerticalAlignment="Top"/>
<CheckBox Content="Sheduled" HorizontalAlignment="Left" Margin="193,105,0,0" VerticalAlignment="Top"
         IsChecked="{Binding CurrentProcess.Sheduled}"/>

<Label Content="Description" HorizontalAlignment="Left" Margin="75,160,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,161,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Description}" VerticalAlignment="Top" Width="177"/>

<Label HorizontalAlignment="Left" Margin="95,234,0,0" VerticalAlignment="Top"/>
```

Converter Binding

```
<Window.DataContext>
    <local:ProcessesVM />
</Window.DataContext>
<Window.Resources>
    <local:EnumConverter x:Key="EnumConverter"/>
</Window.Resources>
```

```
<RadioButton Content="Running" HorizontalAlignment="Left" Margin="193,223,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.RUNNING}" />
    </RadioButton.IsChecked>
</RadioButton>
<RadioButton Content="Ready" HorizontalAlignment="Left" Margin="193,243,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.READY}" />
    </RadioButton.IsChecked>
</RadioButton>
<RadioButton Content="Blocked" HorizontalAlignment="Left" Margin="193,262,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.BLOCKED}" />
    </RadioButton.IsChecked>
</RadioButton>
```

ICommand & RelayCommand…

```
public class RelayCommand : ICommand
{
    readonly Func<Boolean> _canExecute;
    readonly Action _execute;

    0 Verweise
    public RelayCommand(Action execute) : this(execute, null) { }
    2 Verweise
    public RelayCommand(Action execute, Func<Boolean> canExecute) {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    public event EventHandler CanExecuteChanged...
    2 Verweise
    public Boolean CanExecute(Object parameter) {
        return _canExecute == null ? true : _canExecute();
    }
    2 Verweise
    public void Execute(Object parameter){
        _execute();
    }
}
```

CanExecute

- if something changes, the button has to activate and deactivate itself
- therefore the Eventhandler CanExecuteChanged has to add or remove a value

```
public event EventHandler CanExecuteChanged {
    add => CommandManager.RequerySuggested += value;
    remove => CommandManager.RequerySuggested -= value;
}
```

Using RelayCommand

ICommand
as
Returntype

would be
even better
...

```
public RelayCommand AddCommand
    => new RelayCommand(
        o => {
            Processes.Add(CurrentProcess);
            SelectedItem = null;
        }
    );
public RelayCommand ClearCommand
    => new RelayCommand(
        o => SelectedItem = null
    );
public RelayCommand DeleteCommand
    => new RelayCommand(
        o => Processes.Remove(CurrentProcess)
    );
```

Command Binding

```
<Button Content="Add"
        Command="{Binding AddCommand}"/>
<Button Content="Clear"
        Command="{Binding ClearCommand}" />
<Button Content="Delete"
        Command="{Binding DeleteCommand}"/>
<DataGrid Name="dgrProcesses"
          ItemsSource="{Binding Processes}" IsReadOnly="True"
          SelectedItem="{Binding SelectedItem}" />
```

Topics

- Explain MVVM vs MVC
  - Explain the Difference between Model and ViewModel
- Which kind of properties are in the VM-class?
- Where is the INotifyPropertyChanged needed?
- What does INotifyPropertyChanged do?
- Why use an ObservableCollection?
- How does Binding work?
- Where is the Interface ICommand needed?
- Explain the RelayCommand class
- When is the IValueConverter needed?
- Explain Binding with Enum values.