

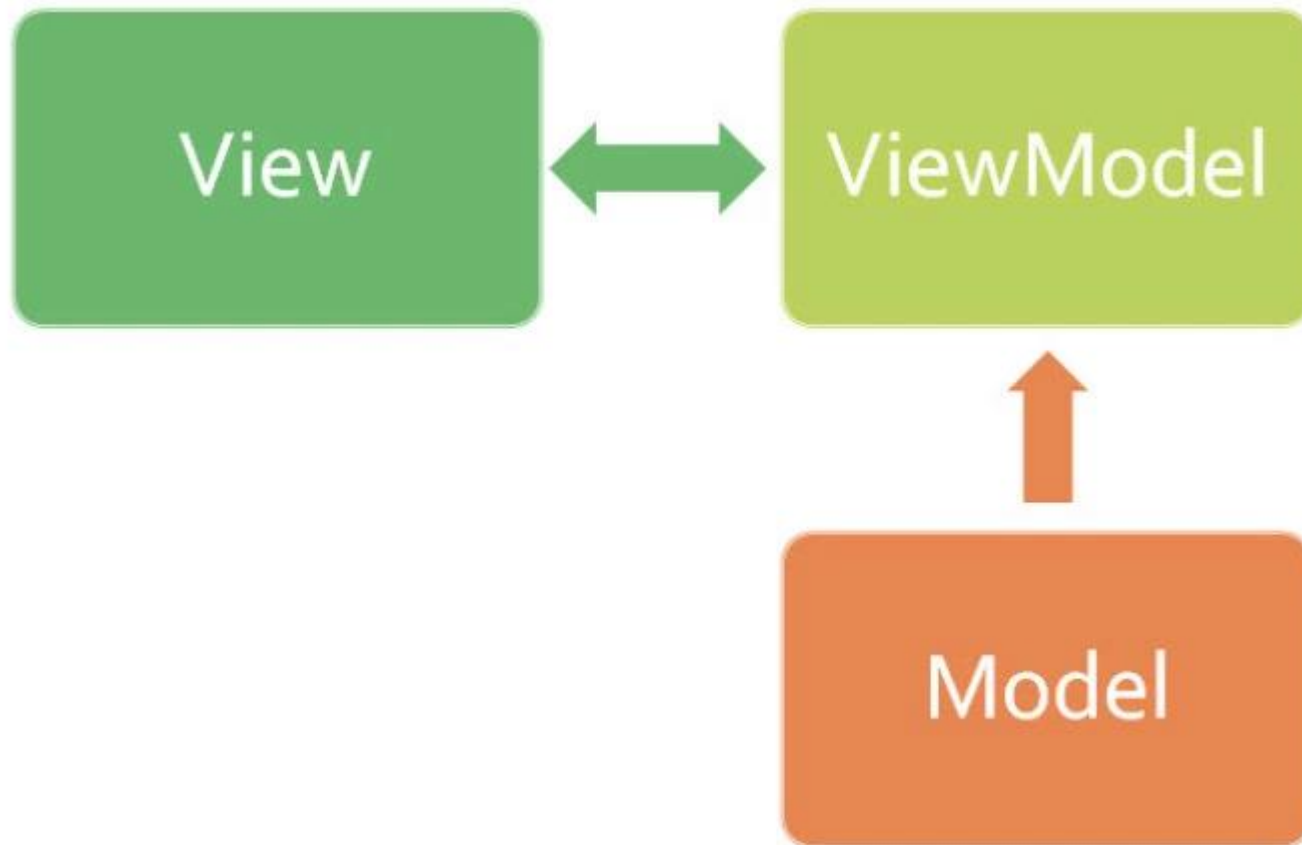
# UITWERKING OPGAVE

Model View ViewModel

# MVVM

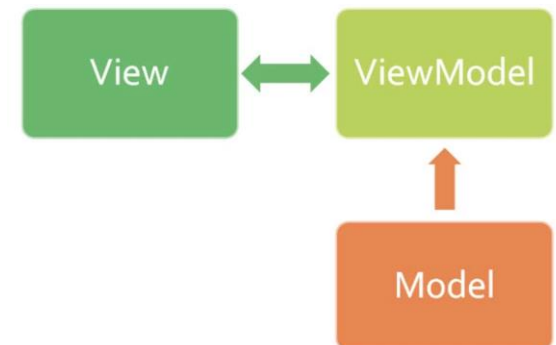
- Voor
  - ▣ *DataBinding* in twee richtingen
- Nu
  - ▣ Structuur inbouwen
    - Design Pattern ⇒ MVVM
    - Zelf opbouwen van scratch
  - ▣ Niet alleen *Binding* van *Data* maar ook van *Commands*
- Na
  - ▣ Database
  - ▣ MVVM Light Framework

# MVVM



# Model

- Het *Model* beschrijft de data of objecten waarmee gewerkt wordt.
- Het *Model* bestaat volledig zelfstandig en refereert nooit naar *View* of *ViewModel* (omgekeerd wel).

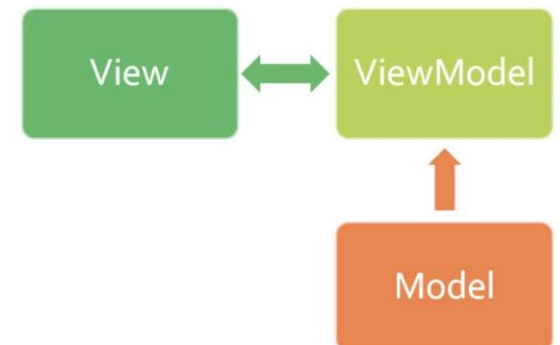


# Model

- Het model bevat de klasse *Student* met de eigenschappen *Naam*, *Jaar*, *Groep*.
- Het model beschrijft in de klasse *Student* niet de lay-out (font, size, style) van een student-object.
- Het model is een XML-file met afzonderlijke studentobjecten.
- Het model is een data access laag (*Dapper*, *LinQToSQL*, *EF*) die de koppeling legt naar een tabel *Student* in een database.

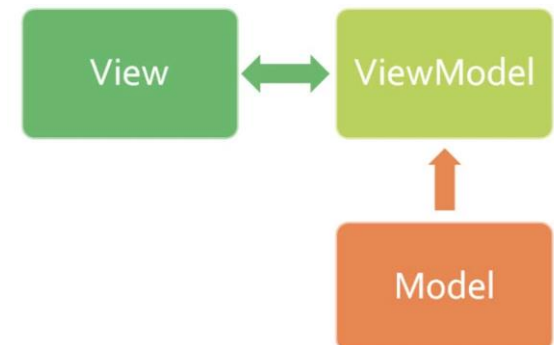
# View

- De **View** is de presentatielaag of de visuele weergave uitgewerkt in XAML.
- De view bevat geen/weinig logica in de code-behind.
- Data i/d **View** wordt aangeboden door het **ViewModel** aangeboden
- De View communiceert alleen met het ViewModel via **Binding**.



# ViewModel

- Het *ViewModel* fungeert als lijm (glue) tussen de *View* en het *Model*.
- *ViewModel* bevat
  - ▣ Informatie over de *Models* die de *View* kan gebruiken voor *DataBinding*.
  - ▣ *Commands* die de *View* kan gebruiken om te interageren met de data in het model.



# MVVM

- 😊 *Separation of concerns (maintainability en extensibility)*
- 😊 Door de ontkoppeling van de view is het creëren van het ontwerp voor de UI makkelijker uit te besteden aan een designer.
- 😊 UI ontwerp en programmatie kunnen in principe parallel plaatsvinden en zo kan er tijdswinst geboekt worden.

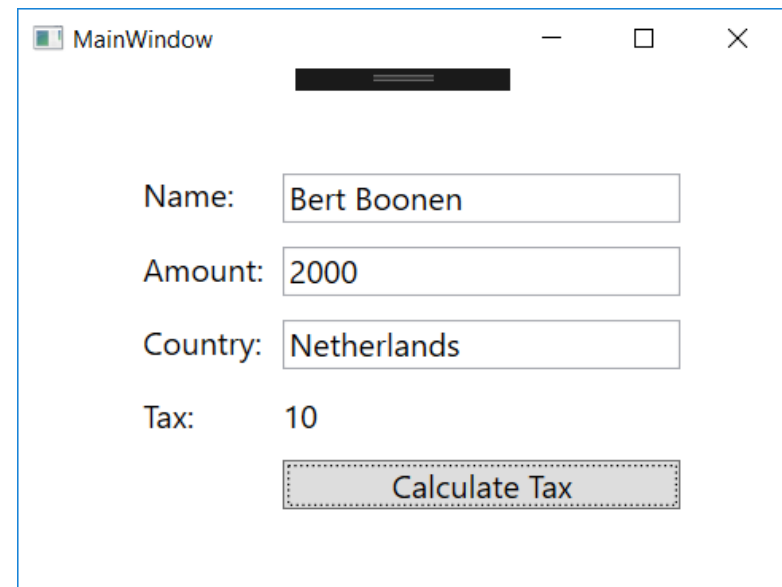


# MVVM

- 😊 Functionaliteiten van het *ViewModel* kunnen geautomatiseerd getest worden met unittesten (grotere applicaties) en dit onafhankelijk van de *View*.
- 😞 *Overkill* voor eenvoudige applicaties
- 😞 *DataBindings* zijn moeilijker te debuggen

# Voorbeeld WpfMVVMFirst

- File | New WPF Project
- WpfMVVMFirst
- Maak 3 mappen
  - Model
  - View
  - ViewModel
- Doe *MainWindow* weg



The screenshot shows a WPF application window titled "MainWindow". The window contains a form with the following elements:

- A black header bar.
- Four input fields with labels: "Name:" (containing "Bert Boonen"), "Amount:" (containing "2000"), "Country:" (containing "Netherlands"), and "Tax:" (containing "10").
- A "Calculate Tax" button at the bottom.

# MainWindow

- Creëer *MainWindow* in *View-map*
- Let op: aparte namespace (rebuild)

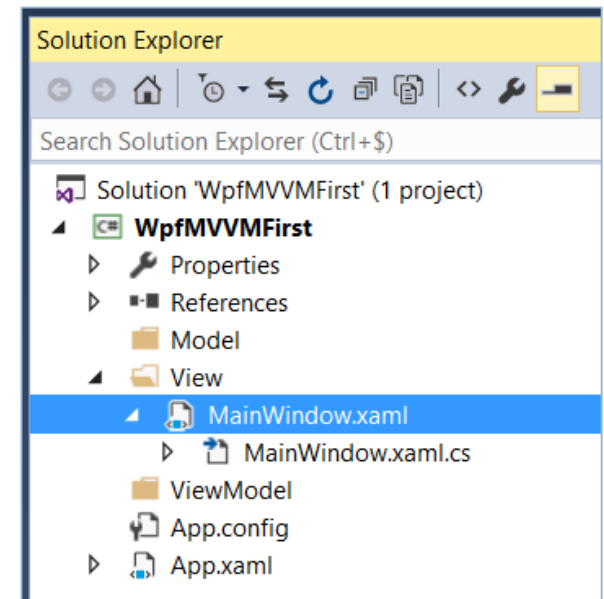
```
namespace WpfMVVMFirst.View
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

# StartUri

## □ Pas StartUri aan in App.Xaml

```
<Application x:Class="WpfMVVMFirst.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:WpfMVVMFirst"
    StartupUri="View/MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```



# Model

## □ Klasse *Customer* in map *Model*

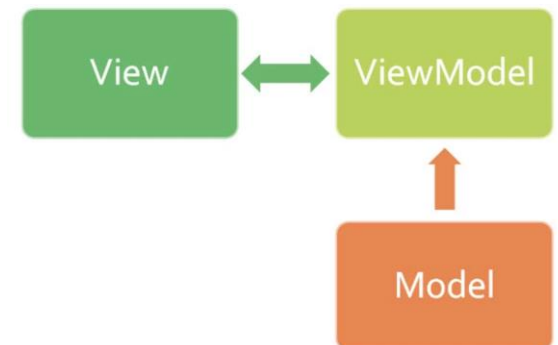
```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace WpfMVVMFirst.Model
{
    class Customer: INotifyPropertyChanged
    {
        private string name;
        private int amount;
        private string country;

        private int tax;
    }
}
```

# Model

- *Customer* implementeert de interface *INotifyPropertyChanged*
- Dankzij de interface weet de bovenliggende *ViewModel* wanneer een property van een instantie wijzigt en bijgevolg andere data moeten doorgespeeld worden naar de bound controls in de *View*.



# Model

```
public event PropertyChangedEventHandler PropertyChanged;

// Deze methode wordt opgeroepen in de setter van elke property.
// [CallerMemberName Attribute] is nieuw in NET Framework 4.5.
// Dit attribuut zorgt automatisch voor bepalen van de calling propertyName!
// Laat toe om bij de properties NotifyPropertyChanged() op te roepen ipv
// OnPropertyChanged("naam property")

private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

# Model

- *PropertyChanged* event wordt getriggerd telkens een property via de setter van waarde verandert
- Idem voor *Amount* en *Country*

```
public string Name
{
    get
    {
        return name;
    }

    set
    {
        name = value;
        NotifyPropertyChanged();
    }
}
```



# Model

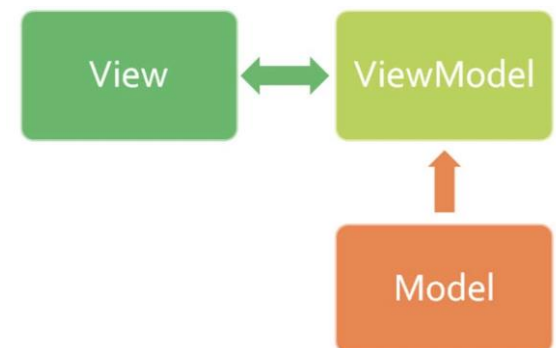
```
public Customer(string name, int amount, string country)
{
    Name = name;
    Amount = amount;
    Country = country;
}

public int Tax
{
    get
    {
        return tax;
    }
}
```

# ViewModel

- Maak klasse *CustomerViewModel* aan in de map *ViewModel*
- *CustomerViewModel* bestaat uit properties om instanties van één of meer *Models* door te spelen aan de *View* (hier alleen *Customer*)
- Door ook hier *INotifyPropertyChanged* te implementeren blijft de *View* permanent op de hoogte bij wijzigingen in de doorgegeven objecten.

The screenshot shows a window titled 'MainWindow' with a light blue border. Inside, there's a form with the following elements: a text box for 'Name' containing 'Bert Boonen', a text box for 'Amount' containing '2000', a text box for 'Country' containing 'Netherlands', a label 'Tax:' followed by the value '10', and a button labeled 'Calculate Tax' with a dashed border.



# BaseViewModel.cs

- Code zoals de implementatie van *INotifyPropertyChanged* die in elke *ViewModel*-klasse terugkomt, verhuist bij voorkeur naar een aparte klasse zoals *BaseViewModel*.
- Door de *ViewModel*-klassen dan te laten overerven van *BaseViewModel*, vermijden we herhaling in de code.

# BaseViewModel.cs

```
using System.Runtime.CompilerServices;
using System.ComponentModel;
using System;

namespace WpfMVVMFirst.ViewModel
{
    class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public void NotifyPropertyChanged([CallerMemberName] String propertyName
= "")
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

# ViewModel

```
using WpfMVVMFirst.Model;
using System.Windows.Input;

namespace WpfMVVMFirst.ViewModel
{
    class CustomerViewModel : BaseViewModel
    {
        private Customer customer;

        public Customer Customer
        {
            get { return customer; }
            set
            {
                customer = value;
                NotifyPropertyChanged();
            }
        }
    }
}
```

# ViewModel

```
public CustomerViewModel()  
{  
    LadenCustomer();  
}
```

```
private void LadenCustomer()  
{  
    Customer = new Customer("Bert Boonen", 2000, "Netherlands");  
}
```

# View (uit notities)

MainWindow

(70.4) 1\*

Name:

Amount:

Country:

Tax:

Calculate Tax

# View

```
<Style TargetType="TextBlock">
<Style TargetType="TextBox">
<Style TargetType="Button">

<Grid VerticalAlignment="Center"
HorizontalAlignment="Center" >

<TextBlock Text="Name:" />
<TextBox Grid.Column="1" Text="" />

<Button Grid.Column="1" Grid.Row="4"
Content="Calculate Tax" />
```



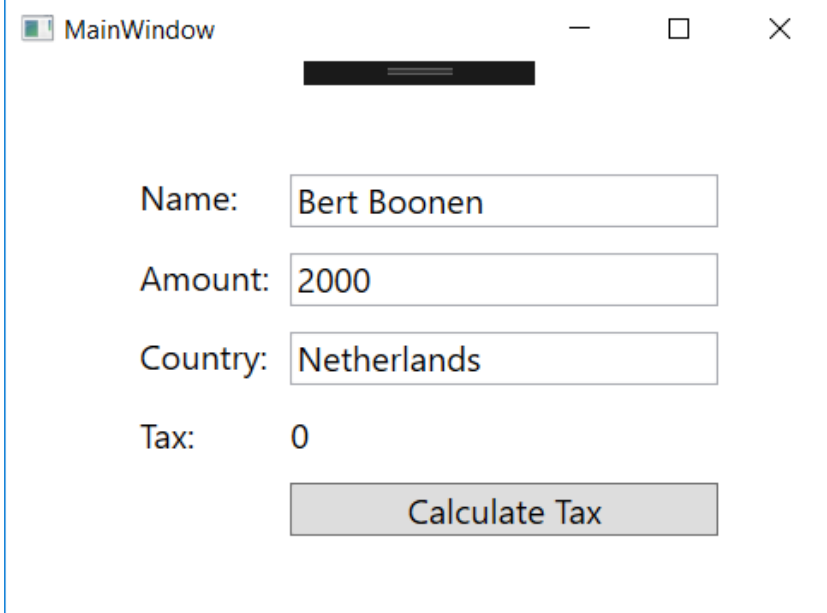
# DataBinding

- *View koppelen aan ViewModel*
- *DataContext binden aan het ViewModel*

```
<Window ...  
    xmlns:local="clr-namespace:WpfMVVMFirst.View"  
    xmlns:viewmodel="clr-namespace:WpfMVVMFirst.ViewModel"  
    ...  
  
<Window.Resources>  
    <viewmodel:CustomerViewModel x:Key="CustomerViewModel" />  
</Window.Resources>  
  
<Grid ...  
    DataContext="{StaticResource CustomerViewModel}" >
```

# View

```
<TextBox Grid.Column="1" Text="{Binding Customer.Name}" />
<TextBox Grid.Column="1" Grid.Row="1"
          Text="{Binding Customer.Amount}" />
<TextBox Grid.Column="1" Grid.Row="2"
          Text="{Binding Customer.Country}" />
<TextBlock Grid.Column="1" Grid.Row="3"
            Text="{Binding Customer.Tax}" />
```



The screenshot shows a WPF application window titled "MainWindow". It contains a form with four input fields and a button. The fields are labeled "Name:", "Amount:", "Country:", and "Tax:". The "Name:" field contains "Bert Boonen", the "Amount:" field contains "2000", and the "Country:" field contains "Netherlands". The "Tax:" field contains "0". Below the fields is a button labeled "Calculate Tax".

Label	Value
Name:	Bert Boonen
Amount:	2000
Country:	Netherlands
Tax:	0

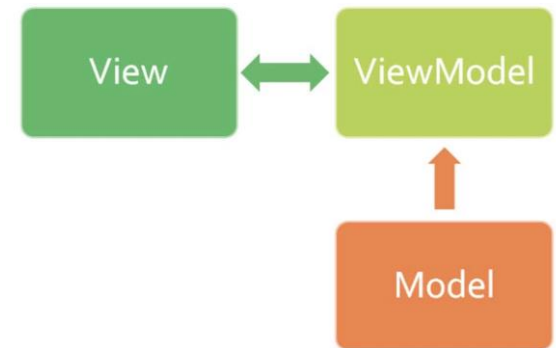
Calculate Tax

# Commands

- Berekenen van Tax via *Binding*
- Dus niet in de CodeBehind
- Methode om Tax te berekenen in Model
- Methode in *Model* wordt via *ViewModel* opgeroepen na *Binding*

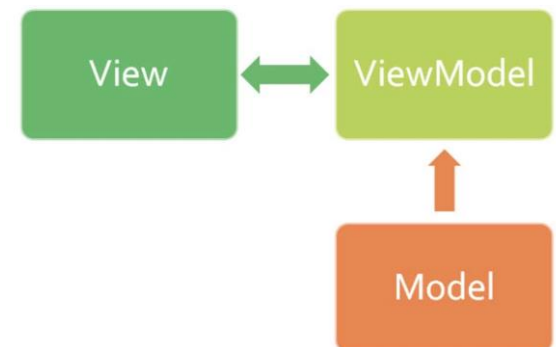
The screenshot shows a window titled 'MainWindow' with a standard Windows title bar. Inside the window, there is a form with the following fields and controls:

- 'Name:' followed by a text box containing 'Bert Boonen'.
- 'Amount:' followed by a text box containing '2000'.
- 'Country:' followed by a text box containing 'Netherlands'.
- 'Tax:' followed by a text box containing '0'.
- A 'Calculate Tax' button at the bottom right.



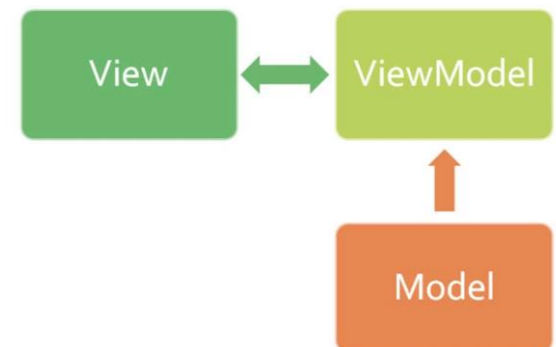
# Customer.cs

```
public void CalculateTax()
{
    if (Amount > 2000)
    {
        tax = 20;
    }
    else if (Amount > 1000)
    {
        tax = 10;
    }
    else
    {
        tax = 5;
    }
}
```



# Command Property

- De communicatie tussen *View* en *ViewModel* gebeurt door middel van *Commands* (hier alleen *CalculateTaxCommand*).
- Elk *Command* is gebaseerd op een klasse die de interface *ICommand* implementeert (naar analogie met *Customer* die *INotifyPropertyChanged* implementeert)
- Voorlopig gebruiken we voor alle *Commands* dezelfde *BaseCommand*-klasse.



# BaseCommand.cs

```
using System;
using System.Windows.Input;

namespace WpfMVVMFirst.ViewModel
{
    class BaseCommand : ICommand
    {
        Action actie;

        public BaseCommand(Action Actie)
        {
            actie = Actie;
        }

        public event EventHandler CanExecuteChanged;

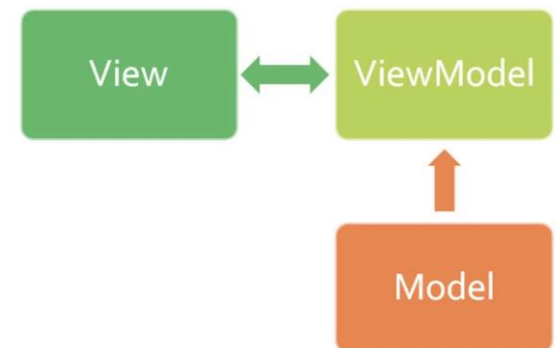
        public bool CanExecute(object parameter)
        {
            return true;
        }

        public void Execute(object parameter)
        {
            actie.Invoke();
        }
    }
}
```

# CustomerViewModel.cs

- *CalculateTaxCommand* wordt als property gedeclareerd in code *ViewModel* zodat de *View* er *Binding* mee kan doen

```
private ICommand calculateTaxCommand;  
public ICommand CalculateTaxCommand  
{  
    get  
    {  
        return calculateTaxCommand;  
    }  
    set  
    {  
        calculateTaxCommand = value;  
    }  
}
```



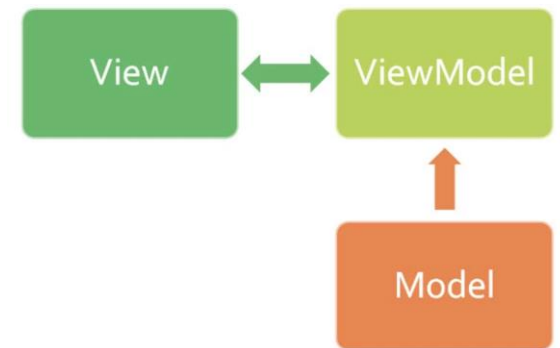
# CustomerViewModel.cs

- Bij instantiëren wordt de uit te voeren code nog geïmplementeerd.

```
public CustomerViewModel()
{
    LadenCustomer();
    KoppelenCommand();
}

private void KoppelenCommand()
{
    CalculateTaxCommand = new BaseCommand(BerekenTax);
}

private void BerekenTax()
{
    Customer.CalculateTax();
}
```

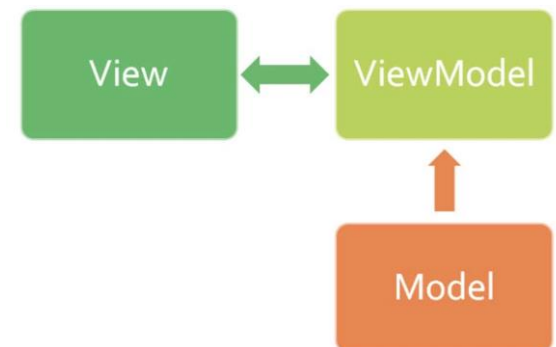




# Binding

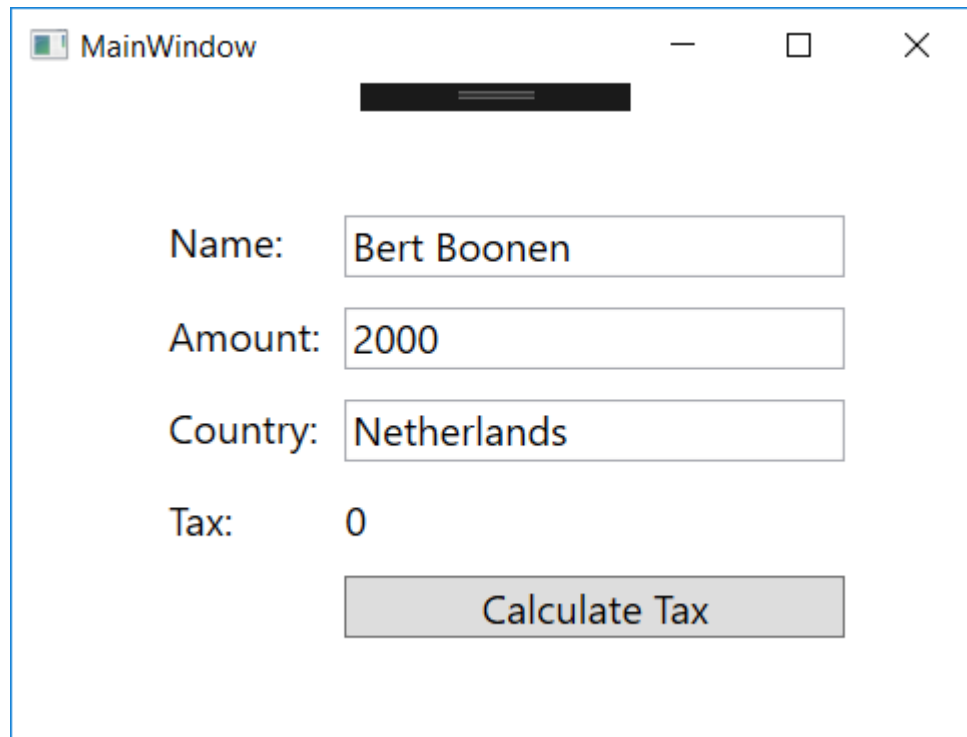
- Het uit te voeren **Command** wordt via *Binding* doorgegeven aan het Command-attribuut van de Button

```
<Button Grid.Column="1" Grid.Row="4"  
        Content="Calculate Tax"  
        Command="{Binding CalculateTaxCommand}" />
```



# Proberen

- Waarom werkt het niet?



The screenshot shows a Windows application window titled "MainWindow". Inside the window, there is a form with the following fields and controls:

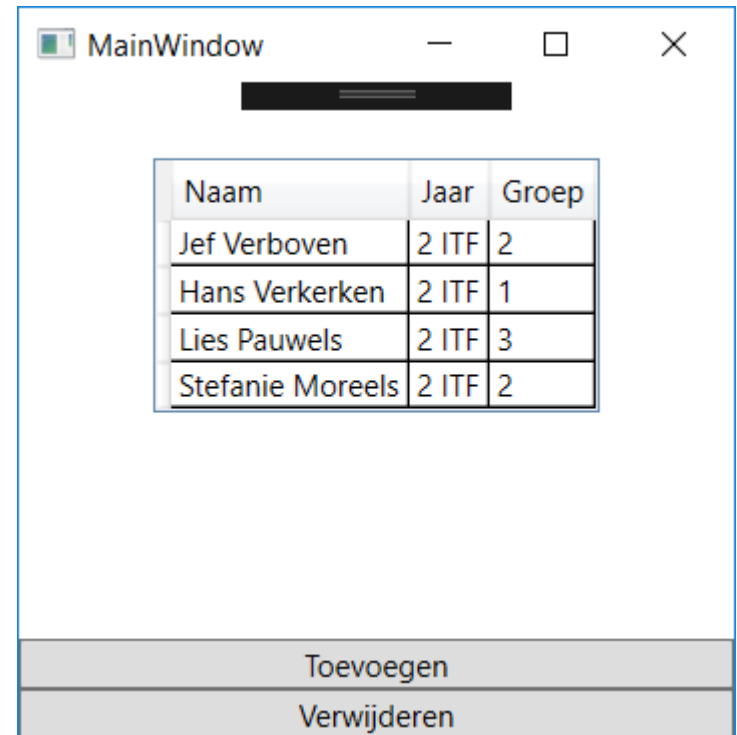
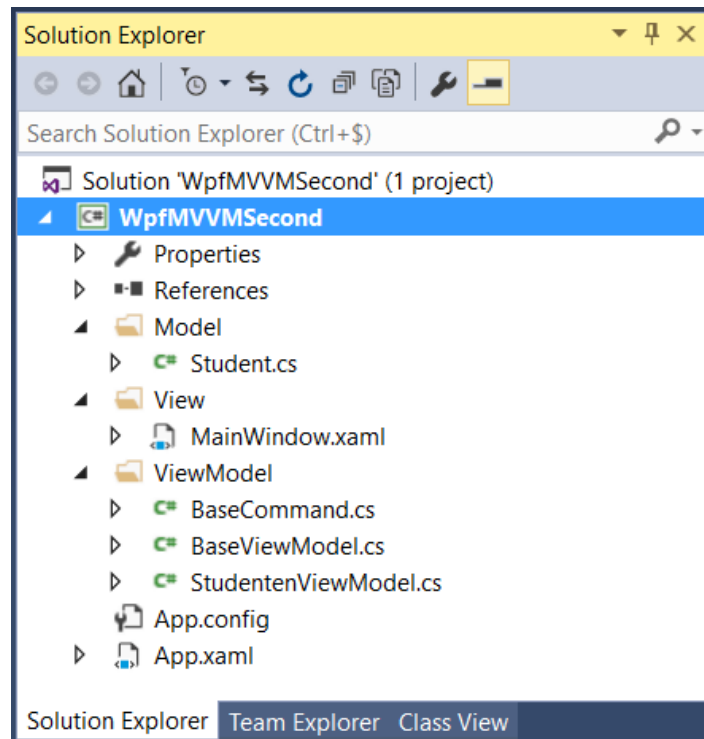
- A black rectangular bar at the top center.
- A "Name:" label followed by a text input field containing "Bert Boonen".
- An "Amount:" label followed by a text input field containing "2000".
- A "Country:" label followed by a text input field containing "Netherlands".
- A "Tax:" label followed by a text input field containing "0".
- A "Calculate Tax" button located below the "Tax" field.

# NotifyPropertyChanged

```
public void CalculateTax()
{
    if (Amount > 2000)
    {
        tax = 20;
    }
    else if (Amount > 1000)
    {
        tax = 10;
    }
    else
    {
        tax = 5;
    }
    NotifyPropertyChanged("Tax");
}
```

# Voorbeeld WpfMVVMSecond

- File | Open WPF Project
- WpfMVVMSecond



# Model

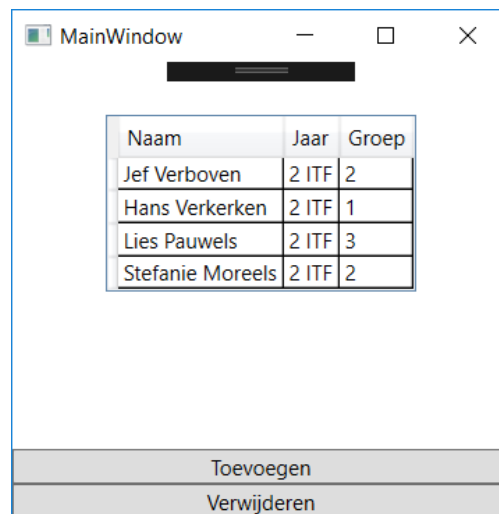
## □ Student.cs

```
namespace WpfMVVMSecond.Model
{
    class Student : INotifyPropertyChanged
    {
        private string naam;
        private string jaar;
        private int groep;

        public Student(string naam, string jaar, int groep)
        {
            Naam = naam;
            Jaar = jaar;
            Groep = groep;
        }
    }
}
```

# ViewModel

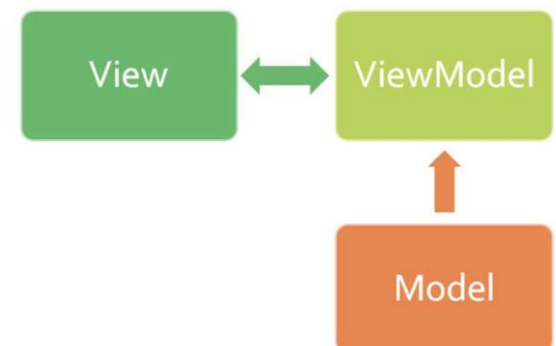
- Klasse StudentenViewModel.cs
- Dit ViewModel stelt een **collectie van Student-objecten** ter beschikking aan de View.
- We kiezen voor een **ObservableCollection** omdat dit type zorgt voor communicatie tussen View en ViewModel bij wijzigingen in de bounded data.



The screenshot shows a window titled 'MainWindow' with a table containing student information. Below the table are two buttons: 'Toevoegen' (Add) and 'Verwijderen' (Remove).

Naam	Jaar	Groep
Jef Verboven	2 ITF	2
Hans Verkerken	2 ITF	1
Lies Pauwels	2 ITF	3
Stefanie Moreels	2 ITF	2

Buttons: Toevoegen, Verwijderen



# StudentViewModel.cs

```
private ObservableCollection<Student> studenten;  
public ObservableCollection<Student> Studenten  
{  
    get  
    {  
        return studenten;  
    }  
    set  
    {  
        studenten = value;  
        NotifyPropertyChanged();  
    }  
}
```

# ViewModel

- Constructor om collectie te instantiëren en op te vullen met data

```
private int nummer;
```

```
public StudentenViewModel()  
{  
    LadenStudenten();  
}
```

```
private void LadenStudenten()  
{  
    Studenten = new ObservableCollection<Student>();  
    Studenten.Add(new Student("Jef Verboven", "2 ITF", 2));  
    Studenten.Add(new Student("Hans Verkerken", "2 ITF", 1));  
    Studenten.Add(new Student("Lies Pauwels", "2 ITF", 3));  
    Studenten.Add(new Student("Stefanie Moreels", "2 ITF", 2));  
}
```



# Binding

```
<Window ...
    xmlns:local="clr-namespace:WpfMVVSecond.View"
    xmlns:viewmodel="clr-namespace:WpfMVVMSecond.ViewModel"
    ...

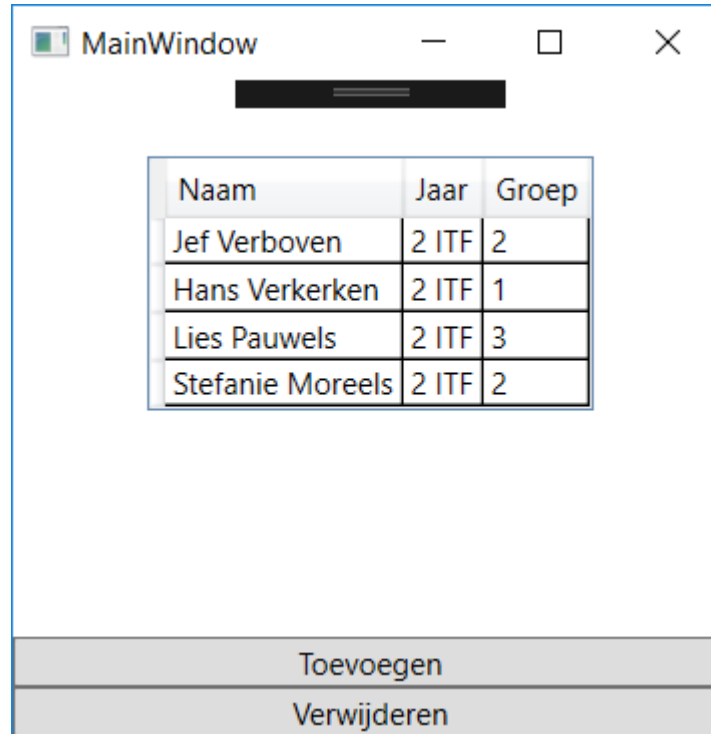
<Window.Resources>
    <viewmodel:StudentenViewModel x:Key="StudentenViewModel" />
</Window.Resources>

<DockPanel DataContext="{StaticResource StudentenViewModel}" >
    ...

<DataGrid ...
    ItemsSource="{Binding Studenten}" />
```

# MainWindow.xaml

- Wat nog i/h ViewModel?



# Commands in ViewModel

```
private ICommand addStudentCommand;  
public ICommand AddStudentCommand  
{  
    get  
    {  
        return addStudentCommand;  
    }  
  
    set  
    {  
        addStudentCommand = value;  
    }  
}
```

// verkorte schrijfwijze

```
public ICommand AddStudentCommand { get; set; }  
public ICommand DeleteStudentCommand { get; set; }
```

# Commands in ViewModel

```
public StudentenViewModel()
{
    LadenStudenten();
    KoppelenCommands();
}

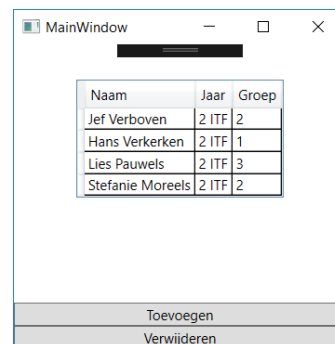
private void KoppelenCommands()
{
    AddStudentCommand = new BaseCommand(Toevoegen);
    DeleteStudentCommand = new BaseCommand(Verwijderen);
}
```

# Toevoegen

```
private void Toevoegen()  
{  
    Studenten.Add(new Student("Student " + (++nummer), "2 ITF",  
nummer));  
}
```

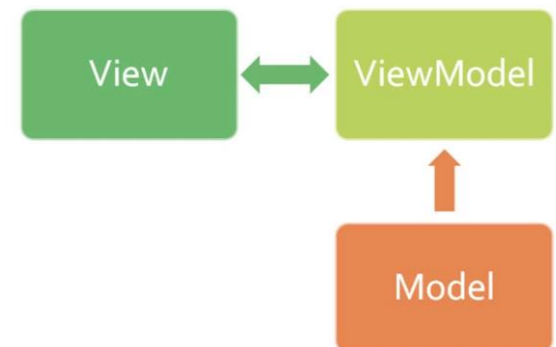
# Verwijderen

- ❑ Verwijderen geselecteerde student in DataGrid uit collectie.
- ❑ MAAR hoe kent het **ViewModel** het geselecteerde item in de **DataGrid**?
- ❑ Attribuut **SelectedItem** van de **DataGrid** via binding doorgeven aan een **SelectedItem-property** in het **ViewModel**.



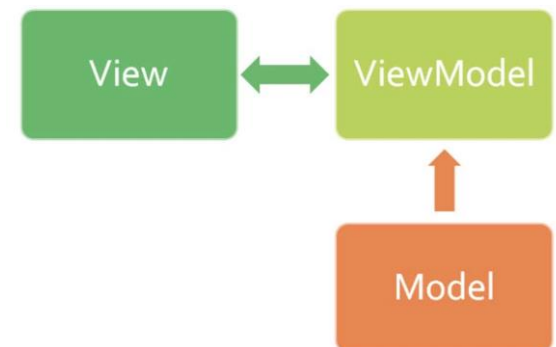
Naam	Jaar	Groep	
Jef Verboven	2	ITF	2
Hans Verkerken	2	ITF	1
Lies Pauwels	2	ITF	3
Stefanie Moreels	2	ITF	2

Toevoegen  
Verwijderen



# SelectedItem

```
private Student selectedItem;  
public Student SelectedItem  
{  
    get { return selectedItem; }  
    set  
    {  
        selectedItem = value;  
        NotifyPropertyChanged();  
    }  
}
```



# Verwijderen

```
private void Verwijderen()
{
    if (SelectedItem != null)
    {
        Studenten.Remove(SelectedItem);
    }
}
```



# Binding

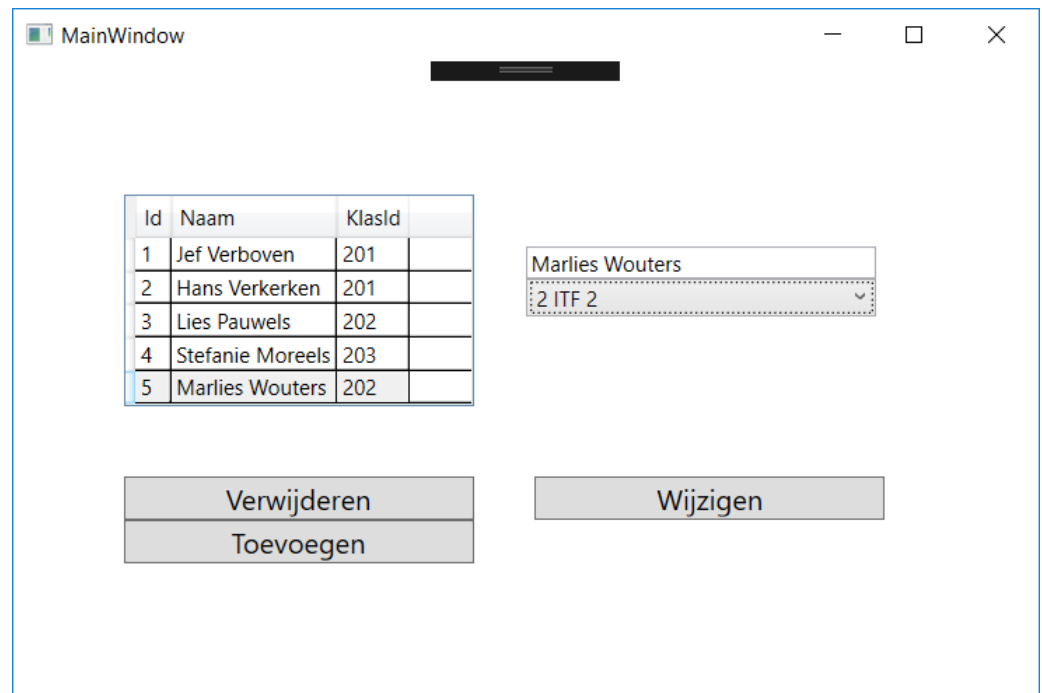
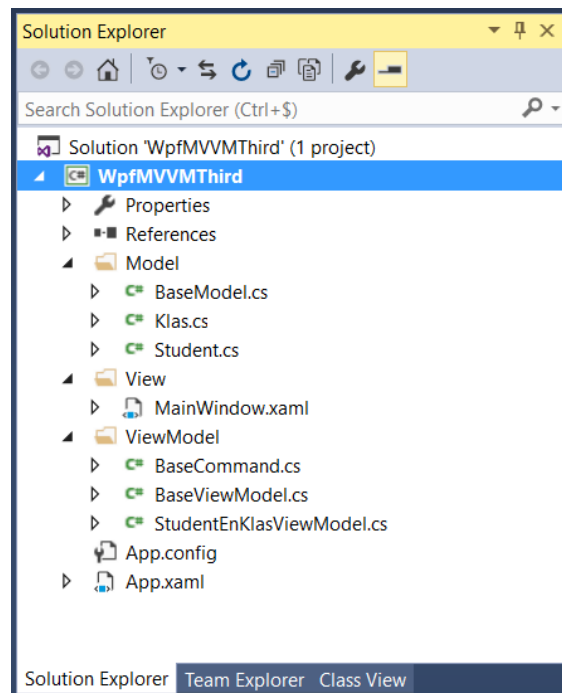
```
<DockPanel DataContext="{StaticResource StudentenViewModel}" >
    <Button Content="Verwijderen" DockPanel.Dock="Bottom"
        Command="{Binding DeleteStudentCommand}" />
    <Button Content="Toevoegen" DockPanel.Dock="Bottom"
        Command="{Binding AddStudentCommand}" />

    <DataGrid HorizontalAlignment="Center" VerticalAlignment="Top"
        Margin="30"
        SelectedItem="{Binding SelectedItem}"
        ItemsSource="{Binding Studenten}" />

</DockPanel>
```

# Voorbeeld WpfMVVMThird

- File | Open WPF Project
- *WpfMVVMThird*



# BaseModel.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace WpfMVVMThird.Model
{
    class BaseModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

# Klas.cs en Student.cs

```
namespace WpfMVVMThird.Model
{
```

```
    class Klas : BaseModel
    {
```

```
        private int id;
        private string naam;
```

```
        public Klas(int id, string naam)
        {
            Id = id;
            Naam = naam;
        }
```

```
        public int Id
        {
            get { return id; }
```

```
            set
            {
                id = value;
                NotifyPropertyChanged();
            }
        }
    }
```

```
class Student: BaseModel
{
```

```
    private int id;
    private string naam;
    private int klasId;
```

```
    public Student(int id, string naam, int klasId)
    {
        Id = id;
        Naam = naam;
        KlasId = klasId;
    }
```

# StudentEnKlasViewModel.cs

- Wat heeft de View nodig?

The screenshot shows a WPF application window titled "MainWindow". Inside the window, there is a table with student data, a search bar, a dropdown menu, and several buttons.

Id	Naam	KlasId	
1	Jef Verboven	201	
2	Hans Verkerken	201	
3	Lies Pauwels	202	
4	Stefanie Moreels	203	
5	Marlies Wouters	202	

Below the table, there are two buttons: "Verwijderen" and "Toevoegen".

To the right of the table, there is a search bar containing "Marlies Wouters" and a dropdown menu showing "2 ITF 2".

Below the search bar, there is a button labeled "Wijzigen".

# StudentEnKlasViewModel.cs

```
private ObservableCollection<Student> studenten;  
public ObservableCollection<Student> Studenten  
{  
    get  
    {  
        return studenten;  
    }  
    set  
    {  
        studenten = value;  
        NotifyPropertyChanged();  
    }  
}  
  
public ObservableCollection<Klas> Klassen { get; set; }  
  
public ICommand AddStudentCommand { get; set; }  
public ICommand DeleteStudentCommand { get; set; }
```

# Data inlezen

```
public StudentEnKlasViewModel()
{
    LadenStudenten();
    LadenKlassen();
    KoppelenCommands();
}

private void LadenStudenten()
{
    Studenten = new ObservableCollection<Student>();
    Studenten.Add(new Student(++newId, "Jef Verboven", 201));
    Studenten.Add(new Student(++newId, "Hans Verkerken", 201));
    Studenten.Add(new Student(++newId, "Lies Pauwels", 202));
    Studenten.Add(new Student(++newId, "Stefanie Moreels", 203));
}

private void LadenKlassen()
{
    Klassen = new ObservableCollection<Klas>();
    Klassen.Add(new Klas(201, "2 ITF 1"));
    Klassen.Add(new Klas(202, "2 ITF 2"));
    Klassen.Add(new Klas(203, "2 ITF 3"));
}
```

# Koppelen Commands

```
private void KoppelenCommands()
{
    AddStudentCommand = new BaseCommand(Toevoegen);
    DeleteStudentCommand = new BaseCommand(Verwijderen);
}

private void Toevoegen()
{
    Studenten.Add(new Student(++newId, "New student", 201));
    SelectedItem = Studenten[Studenten.Count - 1];
}

private void Verwijderen()
{
    if (SelectedItem != null)
    {
        Studenten.Remove(SelectedItem);
    }
}
```



# Binding

```
<Window ...
    xmlns:local="clr-namespace:WpfMVVThird.View"
    xmlns:viewmodel="clr-namespace:WpfMVVMThird.ViewModel"
    ...

<Window.Resources> ...
    <viewmodel:StudentEnKlasViewModel
        x:Key="StudentEnKlasViewModel" />
</Window.Resources>

<Grid DataContext="{StaticResource StudentEnKlasViewModel}" >
    ...

<DataGrid ItemsSource="{Binding Studenten}"
    SelectedItem="{Binding SelectedItem}" />
```

# Resultaat

MainWindow

— □ ×

=====

Id	Naam	KlasId	
1	Jef Verboven	201	
2	Hans Verkerken	201	
3	Lies Pauwels	202	
4	Stefanie Moreels	203	

▼

Verwijderen

Wijzigen

Toevoegen

# Binding

```
<Button Content="Verwijderen"  
        Command="{Binding DeleteStudentCommand}" />  
  
<Button Content="Toevoegen"  
        Command="{Binding AddStudentCommand}" />  
  
<TextBox Text="{Binding SelectedItem.Naam}" Width="200" />
```

# Resultaat

MainWindow

— □ ×

=====

Id	Naam	KlasId	
1	Jef Verboven	201	
2	Hans Verkerken	201	
4	Stefanie Moreels	203	
5	Peter Pan	201	

Peter Pan

▼

Verwijderen

Toevoegen

Wijzigen

# Binding

```
<ComboBox Width="200"  
    ItemsSource="{Binding Klassen}"  
    SelectedValuePath="Id"  
    DisplayMemberPath="Naam"  
    SelectedValue="{Binding Path=SelectedItem.KlasId}"  
/>
```

# Resultaat

MainWindow

— □ ×

=====

Id	Naam	KlasId	
1	Jef Verboven	201	
2	Hans Verkerken	201	
3	Lies Pauwels	202	
4	Stefanie Moreels	203	

Lies Pauwels

2 ITF 2 ▾

Verwijderen

Toevoegen

Wijzigen

# Aan de slag

