

---

# 04 WPF – Windows Presentation Foundation

---

# Agenda

---

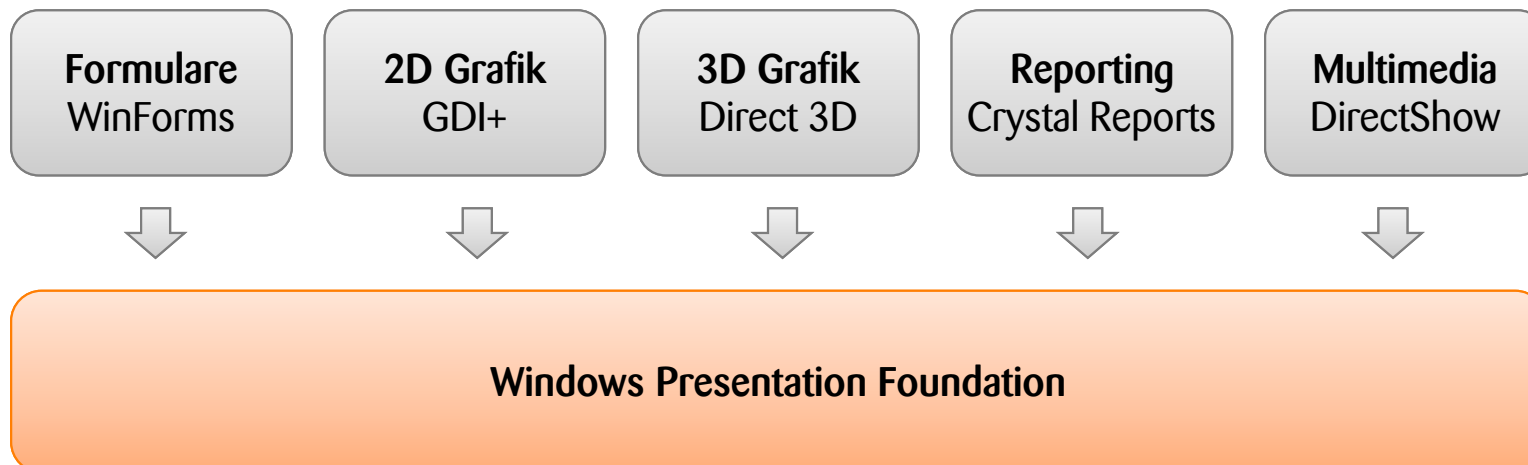
- Einführung
- XAML-Syntax
- Layout-Management
- WPF-Controls
- Data-Binding
- Value-Converter
- Eingabvalidierung
- MVVM-Pattern

## Übersicht

- WPF ist das zukünftige UI-Framework zur Entwicklung von .NET-Applikationen
- Trennung von Darstellung und Verhalten
  - Darstellung und Verhalten sind lose gekoppelt
  - Designer und Entwickler arbeiten in separaten Modellen
  - Design-Tool arbeitet auf XML-basierten Dateien
- Namensraum: `System.Windows`

## Vereinheitlichung verschiedener APIs

- Einfache Integration
- Konsistentes Programmiermodell
- Durchgängiges Layout
- Weniger Abhängigkeiten zu anderen Frameworks



## Neue Funktionen im Überblick

- **Stufenlose Skalierung**
  - GUI-Elemente lassen sich beliebig skalieren
  - Bei Umstellung der Bildschirmauflösung bleiben Fenster gleich
- **Vektororientierte Oberflächen**
  - Keine Pixel, alle Konturen der Elemente bleiben scharf
- **Bessere Performanz**
  - WPF benutzt zur Berechnung die Hardwarebeschleunigung der Grafikkarte
  - Die Berechnungen können notfalls auch durch einen SW-Renderer ausgeführt werden, wenn z.B. keine Hardwarebeschleunigung vorhanden ist

## Neue Funktionen im Überblick

- Wählbares Design der Steuerelemente
  - Alle Steuerelemente sind „skinnable“, d.h. jedem Steuerelement kann das Aussehen individuell bestimmt werden
  - Z.B. hat ein Knopf eine Eigenschaft `Content` statt `Text`. Dadurch kann einem Knopf Text oder Grafik zugeordnet werden
- Deklarative Programmierung – XAML
  - Vorteil: GUI-Definition kann in einem lesbaren, einfachen Format erstellt werden
- Umfangreiche Grafikerunterstützung
  - 2D- und native 3D-Grafiken inkl. Lichtquellen und Kameraperspektiven
  - Nicht zu verwechseln mit DirectX oder OpenGL

## Nützliche Links

### ■ WPF MSDN

- <https://docs.microsoft.com/de-de/dotnet/framework/wpf>
- Offizielle Microsoft WPF-Dokumentation

### ■ WPF-Toolkit

- <https://github.com/xceedsoftware/wpftoolkit>
- Zusätzliche WPF-Controls

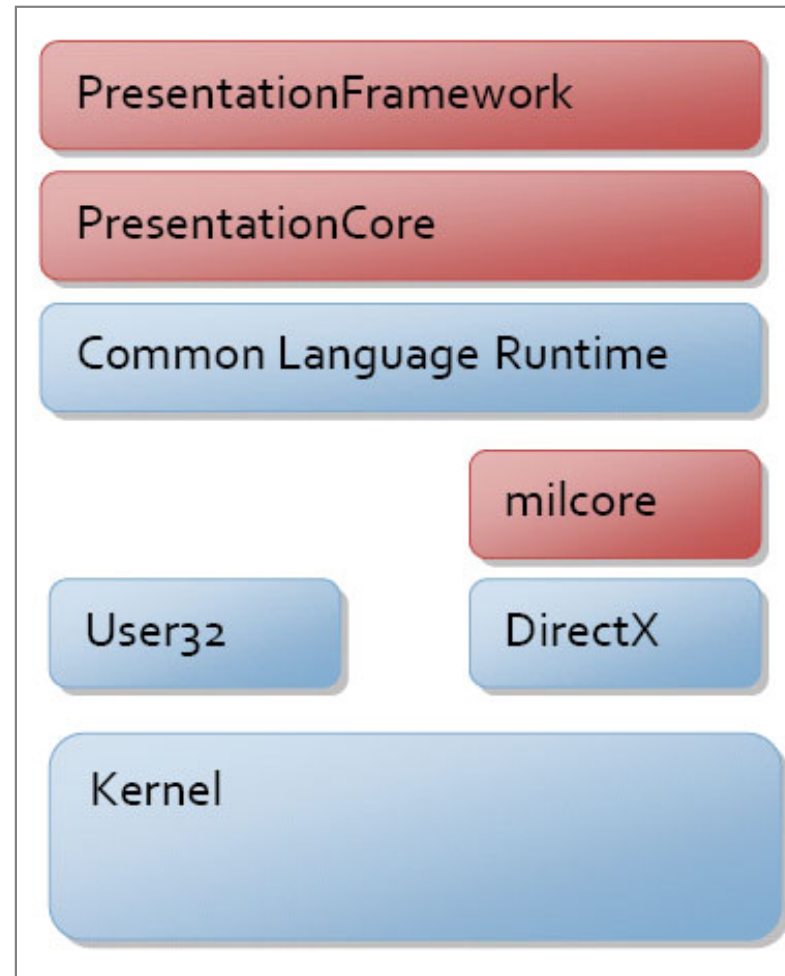
### ■ WPF-Tutorial

- <http://www.wpftutorial.net/>
- Gute Einführung in WPF von Christian Moser (2011)

## WPF-Architektur

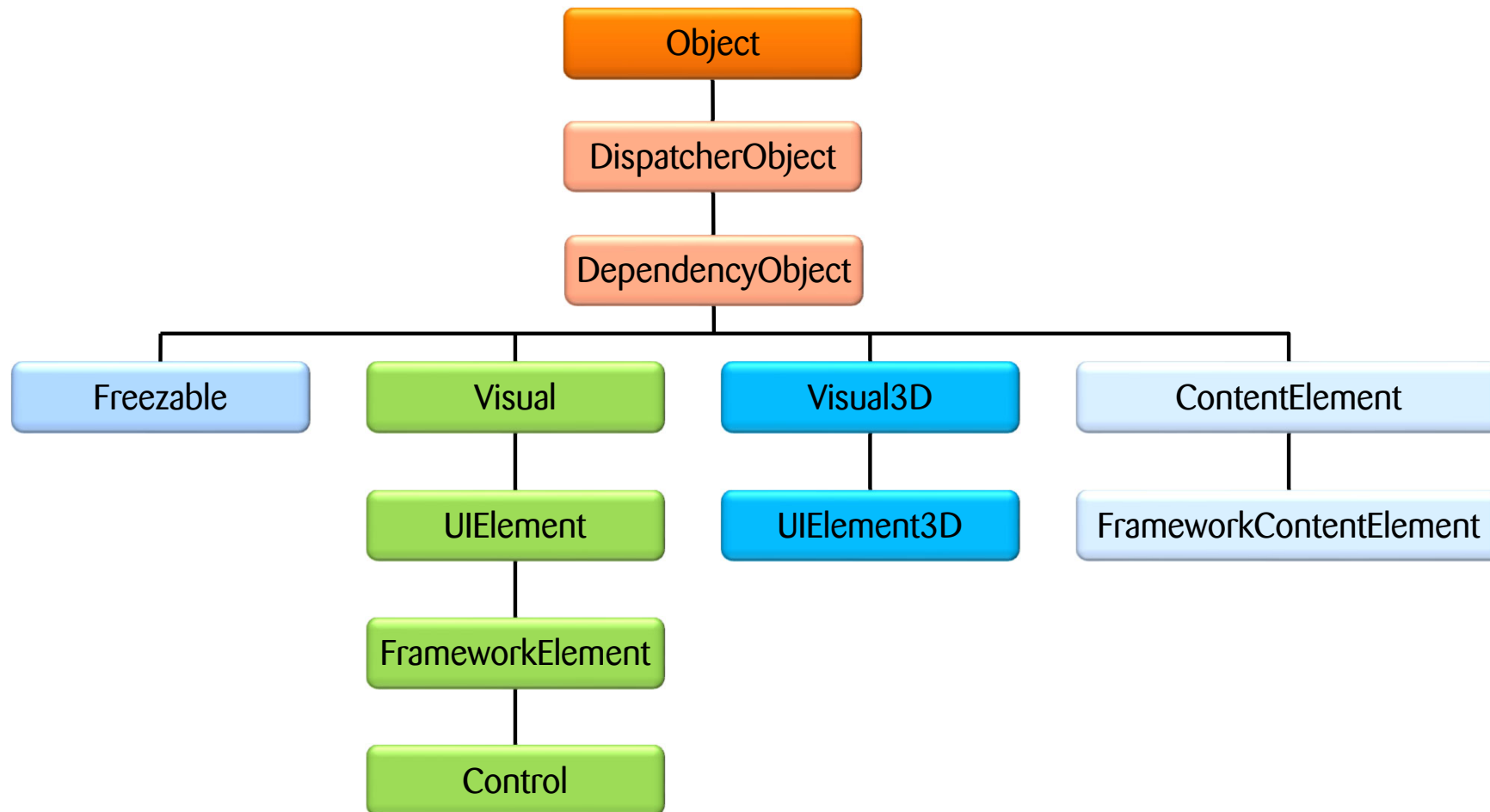
### ■ Hauptkomponenten

- PresentationFramework
- PresentationCore
- Milcore (unmanaged)  
enge Integration mit DirectX
- Nutzt DirectX im Hintergrund  
für effiziente Visualisierung mit  
Hardware- und Softwarerendering





## Die WPF-Hauptklassen



## Die WPF-Hauptklassen

Klasse	Beschreibung
DispatcherObject	Von dieser Basisklasse leiten die meisten WPF-Klassen ab.
DependencyObject	Basisklasse für alle Klassen, welche Dependency Properties unterstützen.
Freezable	Basisklasse für alle Objekte, welche im Read-Only-Status aus Performanzgründen “gefrohren” werden können. Gefrorene Objekte können nicht mehr aufgetaut werden. Meistens grundlegende Graphik-Klassen wie Brush, Pen, Animation Klassen, usw.

## Die WPF-Hauptklassen

Klasse	Beschreibung
Visual	Basisklasse für alle 2D-Objekte.
UIElement	Basisklasse für alle 2D-Objekte, die Routed Events, Command Binding, Layout und Fokus unterstützen.
FrameworkElement	Basisklasse, die Unterstützung für Styles, Data Binding, Resources und gemeinsame Grundfunktionen für Windows-basierte Controls wie Tooltips und Kontextmenüs bietet.
Control	Basisklasse für Controls wie Button, ListBox und StatusBar. Die Klasse Control fügt viele Properties zur Basisklasse FrameworkElement hinzu. Z.B. Foreground, Background und FontSize, sowie die Möglichkeit das Objekt komplett anders zu gestalten.
Visual3D	Basisklasse für alle 3D-Objekte.
UIElement3D	Basisklasse für alle 3D-Objekte mit Support für Routed Events, Command Binding und Focus.
ContentElement	Basisklasse für Content Elemente, ähnlich der Klasse UIElement. Unterstützt grundlegende Eingabemöglichkeiten von Tastatur, Maus, Drag-and-Drop.
FrameworkContentElement	Analoge Klasse zu FrameworkElement für Content.

## Logischer und visueller Elementbaum

### ■ Elemente

- In WPF werden die Elemente hierarchisch angeordnet
- Die Steuerelemente zeichnen sich nicht selber, sondern werden durch WPF gezeichnet
- Elemente können aus visuellen und nicht-visuellen Bausteinen zusammen gesetzt sein

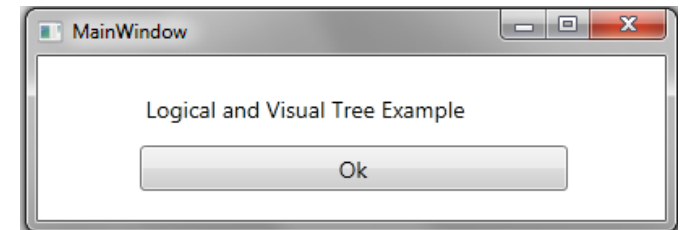
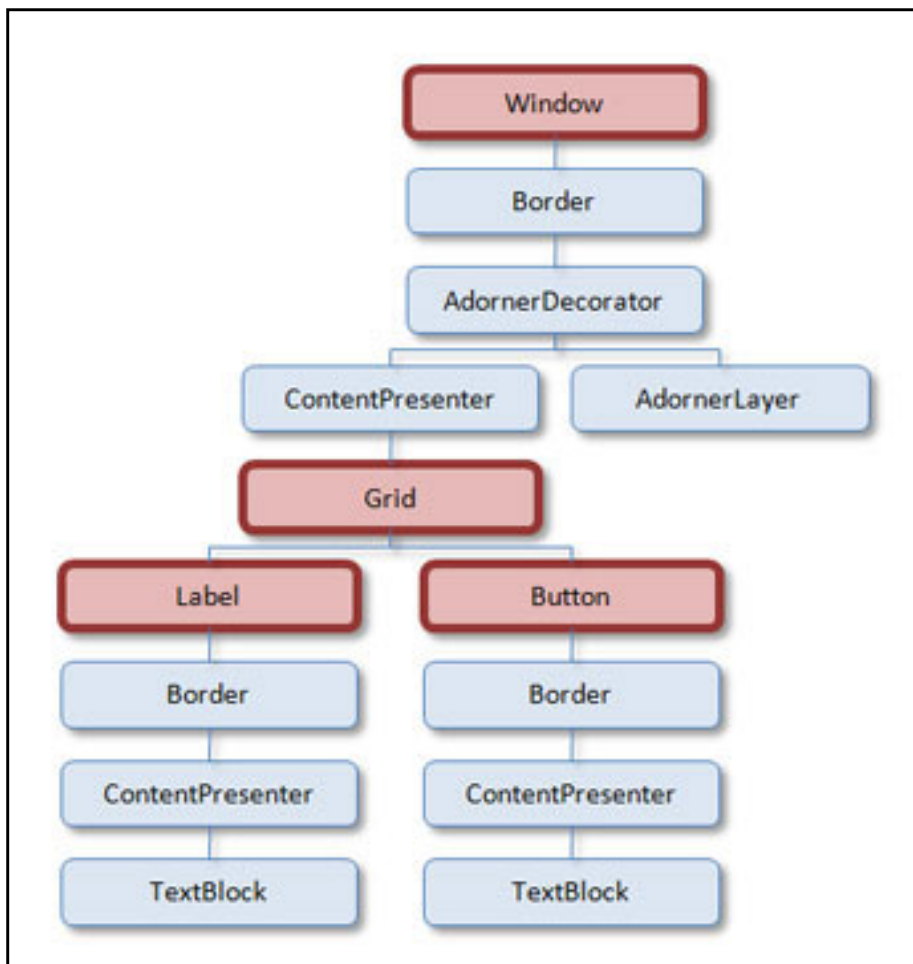
### ■ Logischer Elementbaum (logical tree)

- Enthält alle Steuerelemente, die eingefügt wurden

### ■ Visueller Elementbaum (visual tree)

- Enthält alle sichtbaren Elemente, die auf dem GUI dargestellt werden
- Wird benutzt fürs Rendering, Event Routing, Lokalisieren von Ressourcen

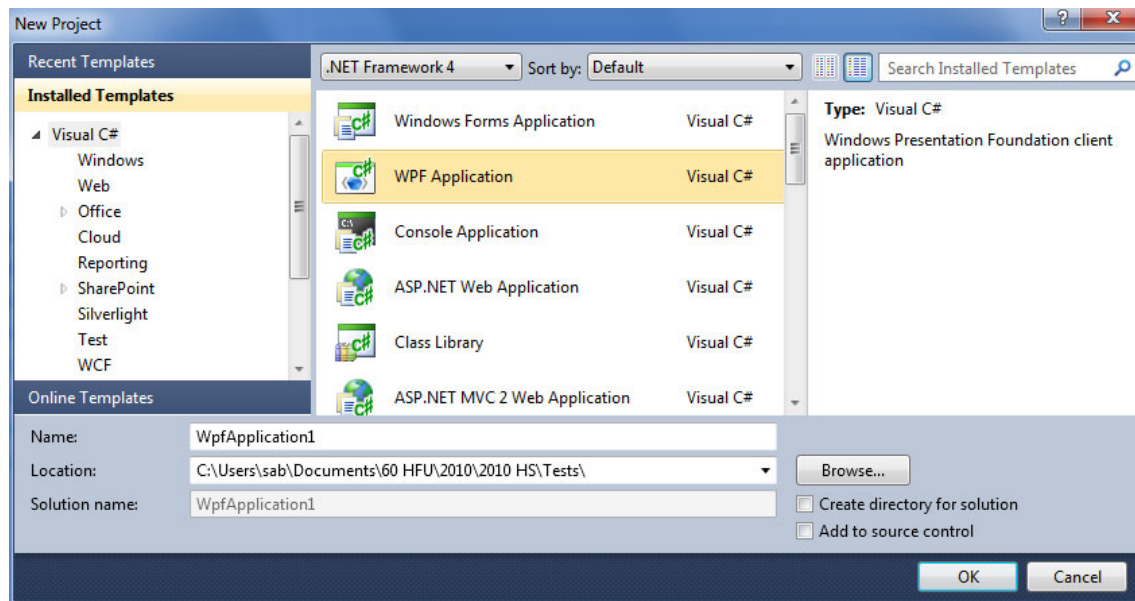
## Logischer und visueller Elementbaum - Beispiel



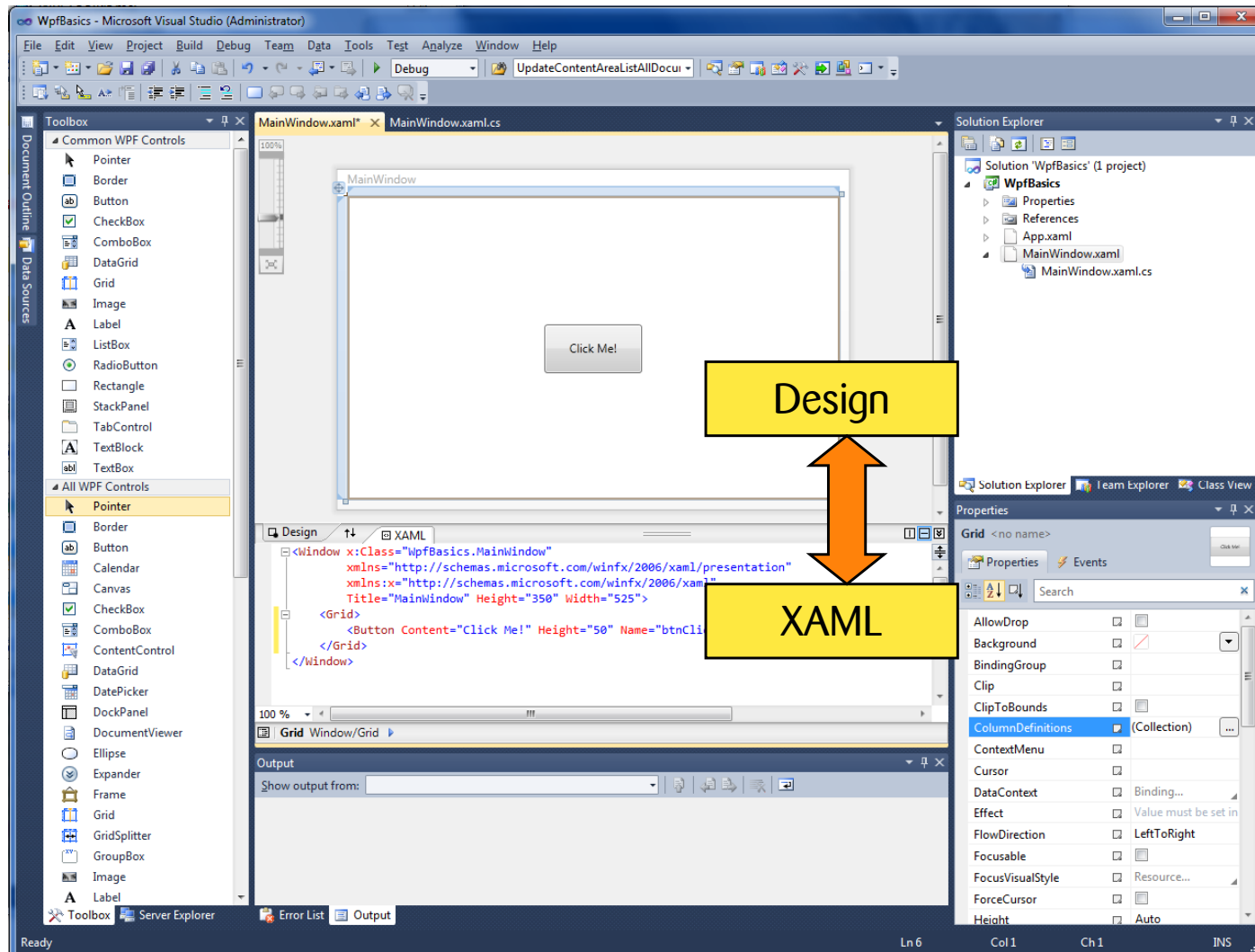
- Logical Tree**
  - Vererbung von Properties
  - Routed Events
  - Auflösen von DynamicResources
  - ElementName DataBinding

- Visual Tree**
  - Rendern der Elemente
  - Hit Testing
  - IsEnabled
  - Opacity
  - Transformation

## Neues WPF-Projekt erstellen



## Design und Markup Language XAML



---

## XAML - eXtensible Application Markup Language

- Deklarative Markup-Sprache zum Instanziieren und Initialisieren von Objekten mit hierarchischen Beziehungen
- Wird mit Code-Behind verknüpft (Programmlogik)
- XAML-Dateien sind grundsätzlich XML-Dateien
- XAML-Beispiel: Ein Knopf

```
<StackPanel>  
    <Button Content="Click Me"/>  
</StackPanel>
```

- XAML wird auch in WF und Silverlight eingesetzt
- Links:
  - <https://docs.microsoft.com/de-de/dotnet/framework/wpf/advanced/xaml-syntax-in-detail>



## Grundlegender Aufbau einer XAML-Datei

### ■ Beispiel: XAML-Code eines leeren Fensters

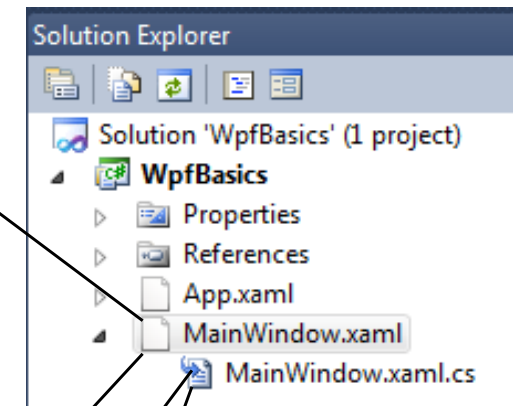
```
<Window x:Class="WpfXAMLMarkupExtension.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">

    <Grid>

    </Grid>
</Window>
```

### ■ Code-Behind-Datei – Analog WinForms

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



## XAML-Syntax

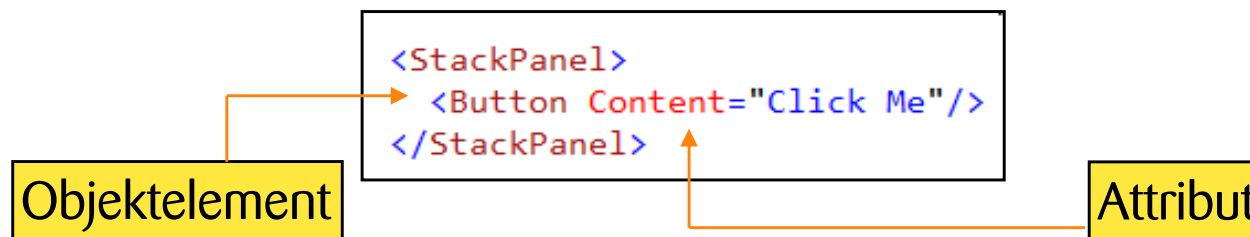
### ■ Objektelemente

- Ein Objektelement deklariert in der Regel eine Instanz eines Typs, welcher in Assemblys als Basis für XAML definiert ist.
- Syntax: `<Objektname> . . . </Objektname>`

### ■ Attributsyntax (Eigenschaften/Properties)

- Die Objekteigenschaften können oft über Attribute gesetzt werden
- Syntax: `Attributname="Wert"`

### ■ Beispiel:



## XAML-Syntax

- Eigenschaftenelementsyntax (**Property Element**):
  - Bei einigen Eigenschaften eines Objektelements kann keine Attributsyntax verwendet werden, da die Informationen nicht in der Zeichenfolge Platz haben
  - Syntax: `<Typname.Eigenschaftename>`

- Beispiel

Elementeigenschaft

```
<Button>
  <Button.Background>
    <SolidColorBrush Color="Blue"/>
  </Button.Background>
  <Button.Foreground>
    <SolidColorBrush Color="Red"/>
  </Button.Foreground>
  <Button.Content>
    This is a button
  </Button.Content>
</Button>
```

## XAML-Syntax

- Auflistungssyntax (**Collection**):
  - Wird in der Regel verwendet, wenn die Eigenschaft eine Auflistung ist
  - Ermöglicht ein besser lesbares Markup
- Beispiel:
  - Festlegen der Eigenschaft `GradientStops` einer `LinearGradientBrush`

LinearGradientBrush



```
<LinearGradientBrush>
  <LinearGradientBrush.GradientStops>
    <!-- no explicit new GradientStopCollection, parser knows how to find or create -->
    <GradientStop Offset="0.0" Color="Red" />
    <GradientStop Offset="1.0" Color="Blue" />
  </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

## XAML-Syntax

- XAML-Inhaltseigenschaften (**Content Properties**):
  - XAML ermöglicht das Festlegen einer Eigenschaft als XAML-Inhaltseigenschaft  
D.h. Definition von: **KlasseX.EigenschaftXY := Inhalt**
  - Ermöglicht ein besser lesbares Markup und reduziert Code
  - Beispiellassen: `Border`, `ViewBox`

- Beispiel: Klasse `Border`

- Die Klasse `Border` spezifiziert die Eigenschaft `Child` als Inhaltseigenschaft (**Content Property**)

```
<Border>
  <TextBox Width="300"/>
</Border>
<!--explicit equivalent-->
<Border>
  <Border.Child>
    <TextBox Width="300"/>
  </Border.Child>
</Border>
```

Beide Code-Segmente  
sind equivalent

---

## XAML-Syntax

### ■ Textinhalt (Text Content):

- Eine kleine Anzahl von XAML-Elementen kann Text direkt als Inhalt verarbeiten
- Eine Bedingung muss erfüllt sein:
  - Die Klasse muss eine Inhaltseigenschaft deklarieren, der ein Text zugewiesen werden kann
  - Typ ist eine bekannte XAML-Sprachprimitive
  - Typ muss einen Typkonverter deklarieren. Textinhalt wird als Initialisierungstext für diesen Typkonverter verwendet.

### ■ Beispiel: Button

- Dieses Beispiel ist eine Kombination von Inhaltseigenschaften und Auflistungssyntax

```
<StackPanel>  
  <Button>First Button</Button>  
  <Button>Second Button</Button>  
</StackPanel>
```

## XAML-Syntax

- Attributsyntax (Ereignisse / Events):
  - Die Attributsyntax kann auch für Ereignisse verwendet werden
  - Attributname entspricht dem Ereignisnamen
  - Attributwert ist der Name des Ereignis-Handlers
- Beispiel: Button-ClickHandler

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="ExampleNamespace.ExamplePage">
  <Button Click="Button_Click" >Click Me!</Button>
</Page>
```

Event

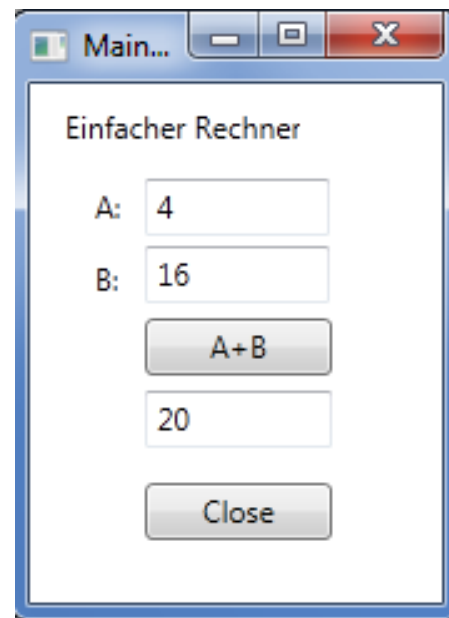
Event Handler

# Übung 4.1



## Einfacher Taschenrechner

- Erstelle eine einfache Taschenrechner-Applikation mit WPF. Der Taschenrechner soll:
- Zwei Eingabefelder haben für die Zahlen A und B
- Eine Funktion «Addition» haben





## XAML-Syntax

- Markup-Erweiterungen (**Markup Extensions**):
  - XAML-Sprachkonzept
  - Werden in Attributwerten mit geschweiften Klammern markiert: `{ markupextension }`
  - Meistens zur Datenbindung (**Binding**) zwischen GUI-Element und Datenquelle oder als Referenz auf Ressourcen verwendet (`StaticResource` und `DynamicResource`)

## XAML-Syntax

### ■ Beispiel zu Markup-Erweiterungen

```
<Window x:Class="WpfXAMLMarkupExtension.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <Style TargetType="Border" x:Key="WindowsBackground" >
            <Setter Property="Background" Value="AliceBlue" />
            <Setter Property="TextBlock.FontSize" Value="20"/>
        </Style>
    </Window.Resources>

    <Grid>
        <Border Style="{StaticResource WindowsBackground}">
            <TextBlock Text="MarkupExtension Beispiel" />
        </Border>
    </Grid>
</Window>
```

Markup-Extension  
mit Referenz  
auf Style-Definition  
(StaticResource)

## XAML-Syntax

- Liste der WPF Markup-Erweiterungen (.NET 4.0)
  - Binding Markup Extension
  - ColorConvertedBitmap Markup Extension
  - ComponentResourceKey Markup Extension
  - DynamicResource Markup Extension
  - RelativeSource MarkupExtension
  - StaticResource Markup Extension
  - TemplateBinding Markup Extension
  - ThemeDictionary Markup Extension
  - PropertyPath XAML Syntax
  - PresentationOptions: Freeze Attribute
- Weitere Details unter: <https://docs.microsoft.com/de-de/dotnet/framework/wpf/advanced/wpf-xaml-extensions>

## XAML-Syntax

### ■ Typkonverter

- Der Typkonverter ermöglicht die automatische Konvertierung eines Attributwerts als String in einen primitiven Datentyp, sowie in Objekttypen
- Ein Beispiel ist die Thickness-Struktur, bei welcher ein Typkonverter existiert

## XAML-Syntax

### ■ Typkonverter - Beispiel Thickness-Struktur:

- Definiert den Abstand um ein Rechteck herum
- Properties: Left, Top, Right, Bottom
- Setzen der Werte mittels Typkonvertierung und Attributsyntax:

```
<Button Margin="10,20,10,30" Content="Click me"/>
```

- Setzen der Werte mittels Eigenschaftenelementsyntax:

```
<Button Content="Click me">  
  <Button.Margin>  
    <Thickness Left="10" Top="20" Right="10" Bottom="30"/>  
  </Button.Margin>  
</Button>
```

---

## XAML-Syntax

- XAML-Stammelemente (Root Elements)
  - Eine XAML-Datei darf nur ein Root-Element haben
  - Es gibt eine Reihe Root-Elemente je nach WPF-Anwendung:
    - Window (Seite)
    - Page (Seite)
    - ResourceDictionary (externes Dictionary)
    - Application (Anwendungsdefinition)
- Beispiel: Windows Root Element

```
<Window x:Class="WpfXamlMarkupExtension.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">

    <Grid>

    </Grid>
</Window>
```

---

## XAML-Syntax

- XAML-Stammelemente (**Root Elements**)
  - Im Root-Element sind auch die Attribute `xmlns` und `xmlns:x` vorhanden
  - Diese geben dem XAML-Prozessor an, welche XAML-Namespaces die Typdefinitionen für Basistypen enthalten, auf die das Markup als Elemente verweist
- `xmlns`
  - Gibt den XAML-Standardnamensraum an
  - Objekte innerhalb des Standardnamensraums können ohne Präfix adressiert werden

```
<Window x:Class="WpfXAMLMarkupExtension.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
```

## XAML-Syntax

- XAML-Stammelemente (**Root Elements**)
  - Attribute für Namespaces: **xmlns** und **xmlns:x**
  - Diese geben dem XAML-Prozessor an, welche XAML-Namespaces die Typdefinitionen für Basistypen enthalten, auf die das Markup als Elemente verweist
- **xmlns**
  - XAML-Standardnamensraum referenziert meistens den WPF-Namespace:  
<http://schemas.microsoft.com/winfx/2006/xaml/presentation>
  - Objekte innerhalb des Standardnamensraums können ohne Präfix adressiert werden
- **xmlns:x**
  - Das Attribut gibt einen zusätzlichen XAML-Namespace an, welcher den XAML-Language-Namespace referenziert:  
<http://schemas.microsoft.com/winfx/2006/xaml>



## XAML-Syntax

### ■ Der «x:»-Präfix

- Das «x:»-Präfix adressiert die XAML-Sprachkonstrukte im XAML-Namensraum
- Die meist benutzten «x:»-Präfix-Programmierkonstrukte sind:
  - `x:Key`: Legt einen eindeutigen Schlüssel für jede Ressource in einem `ResourceDictionary` (wird wohl ca. 90% der «x:»-Markups in WPF-Anwendungen sein)
  - `x:Class`: Gibt den CLR-Namensraum und -Klassennamen für die Klasse an, die den Code-Behind für eine XAML-Seite angibt
  - `x:Name`: Gibt einen Laufzeitobjektnamen für die Instanz an, die nach der Verarbeitung eines Objektelements im Laufzeitcode vorhanden ist

## XAML-Syntax

### ■ Ereignisse und XAML-Code-Behind

- Die meisten WPF-Anwendungen bestehen sowohl aus XAML-Markup als auch aus Code-Behind

XAML

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="ExampleNamespace.ExamplePage">
  <Button Click="Button_Click" >Click Me!</Button>
</Page>
```

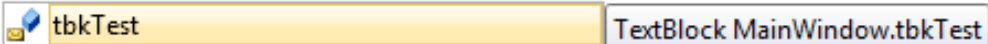
Code-Behind

```
namespace ExampleNamespace
{
    public partial class ExamplePage
    {
        void Button_Click(object sender, RoutedEventArgs e)
        {
            Button b = e.Source as Button;
            b.Foreground = Brushes.Red;
        }
    }
}
```

## XAML-Syntax

### ■ Benannte XAML-Elemente

```
<Window x:Class="WpfXamlCodeBehind.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <StackPanel>
        <TextBlock Name="tbkTest" Text="Hallo" FontSize="20"/>
        <Button Click="Button_Click" Content="Click me" FontSize="20"/>
    </StackPanel>
</Window>
```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    tb
}

```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    tbkTest.Text = "WPF ist COOL!";
}
```

# Layout Management

## «Layout Management Control»-Übersicht

### ■ Design:



### ■ XAML:

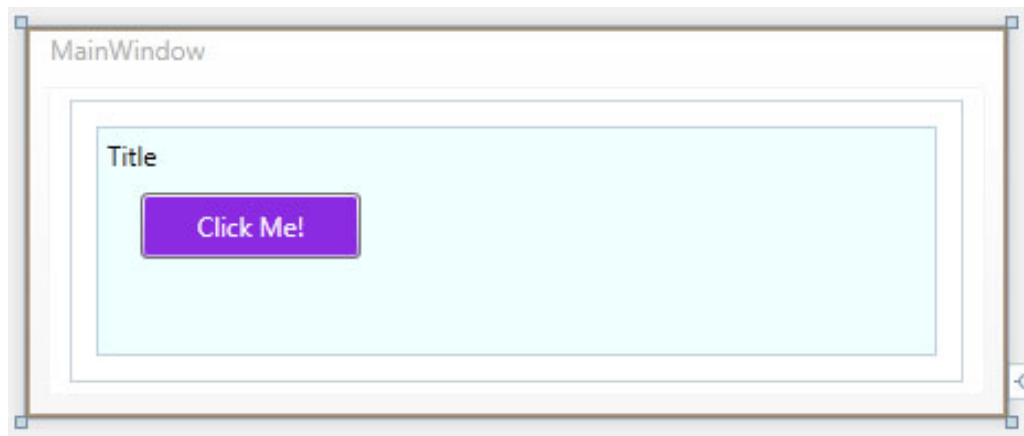
```
<UserControl x:Class="LayoutControls.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="200" d:DesignWidth="300" xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">

    <Grid x:Name="LayoutRoot" Background="White">
        <Canvas Background="AliceBlue" Width="200" Height="100">
            <Button Content="OK" Width="100" Height="40" Canvas.Top="40" Canvas.Left="50"> </Button>
            <sdk:Label Canvas.Left="10" Canvas.Top="10" Height="28" Name="label1" Width="180"
                HorizontalAlignment="Center" Content="Silverlight ist cool!" FontSize="12"/>
        </Canvas>
    </Grid>
</UserControl>
```

# Layout Management

## Canvas – Freie Platzierung der Elemente

### ■ Design:



### ■ XAML:

```
<Window x:Class="Canvas.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="177" Width="445">
  <Grid Height="128" Width="405">
    <Canvas Background="Azure" Margin="12">
      <Label Name="lblTitle" Content="Title" />
      <Button Name="btnClickMe" Background="BlueViolet" Content="Click Me!"
        Width="100" Height="30" Foreground="White" Canvas.Left="20" Canvas.Top="30" />
    </Canvas>
  </Grid>
</Window>
```

# Layout Management

## Grid – Anordnen an Zeilen und Spalten

### ■ XAML:

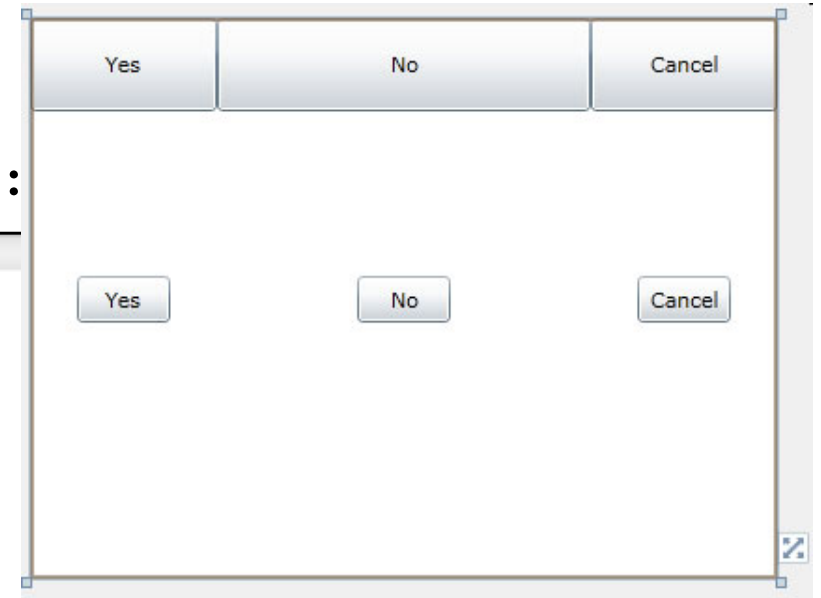
```
<UserControl x:Class="LayoutGrid.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="400">

  <Grid x:Name="LayoutRoot" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition Height="50" />
      <RowDefinition Height="*" />
      <RowDefinition Height="50" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="100" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="100" />
    </Grid.ColumnDefinitions>
    <Button Content="Yes" Grid.Row="0" Grid.Column="0"></Button>
    <Button Content="No" Grid.Row="0" Grid.Column="1"></Button>
    <Button Content="Cancel" Grid.Row="0" Grid.Column="2"></Button>

    <Button Content="Yes" Grid.Row="1" Grid.Column="0" Height="25" Width="50"></Button>
    <Button Content="No" Grid.Row="1" Grid.Column="1" Height="25" Width="50"></Button>
    <Button Content="Cancel" Grid.Row="1" Grid.Column="2" Height="25" Width="50"></Button>

  </Grid>
</UserControl>
```

### Design:



# Layout Management

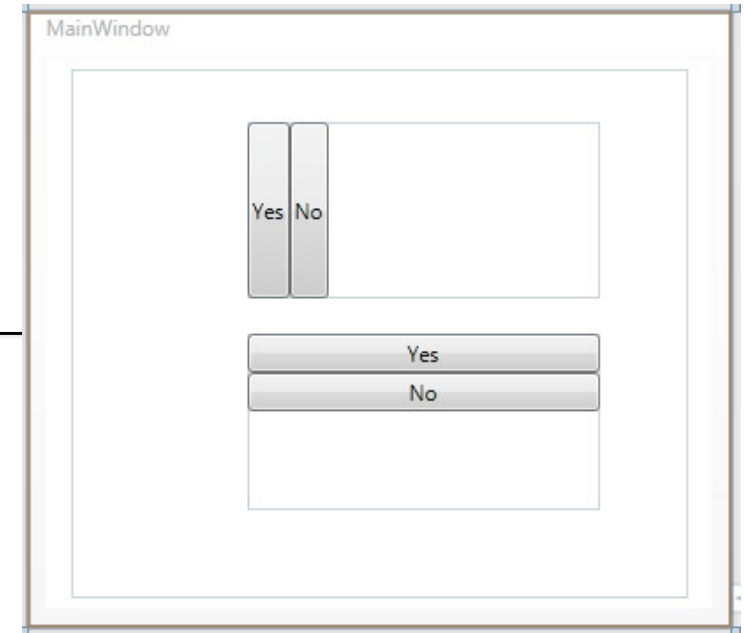
## Stackpanel – Nacheinander anordnen

### ■ XAML:

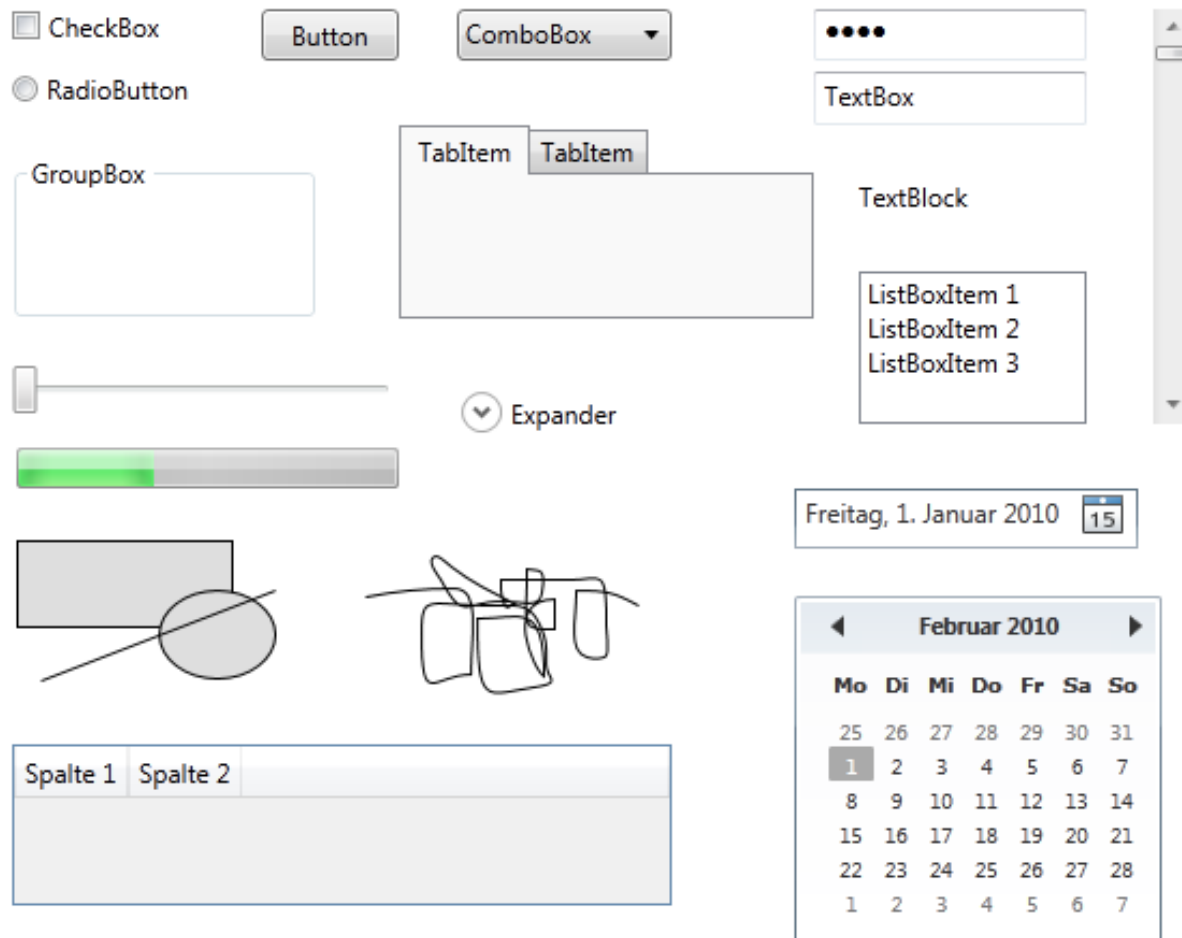
```
<UserControl x:Class="LayoutStackPanels.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel Orientation="Horizontal" Height="100" HorizontalAlignment="Left"
            Margin="126,30,0,0" Name="stackPanel1" VerticalAlignment="Top" Width="200">
            <Button Content="Yes"></Button>
            <Button Content="No"></Button>
        </StackPanel>
        <StackPanel Orientation="Vertical" Height="100" HorizontalAlignment="Left"
            Margin="126,153,0,0" Name="stackPanel2" VerticalAlignment="Top" Width="200">
            <Button Content="Yes"></Button>
            <Button Content="No"></Button>
        </StackPanel>
    </Grid>
</UserControl>
```

### Design:



## Standard-Controls





## TreeView

### ■ Navigationsbaum

- Ein TreeView-Control stellt einen Navigationsbaum dar
- Er besteht aus verschachtelten TreeViewItem
- Bsp: TreeView in XAML und Node mit C#-Code hinzufügen

```
TreeViewItem tvMainItem = new TreeViewItem() { Header = "Bands" };  
tvCarBrands.Items.Add(tvMainItem);  
tvMainItem.FontSize = 18;
```



▸ Bands

- Bsp. Fortsetzung: Elemente zum Node hinzufügen

```
List<string> carBrandList = new List<string>() { "Audi", "BMW", "Ferrari", "Mercedes", "Nissan", "Peugeot", "Porsche" };
```

```
foreach (string brand in carBrandList)  
{  
    TreeViewItem tvItem = new TreeViewItem() { Header = brand };  
    tvMainItem.Items.Add(tvItem);  
}
```



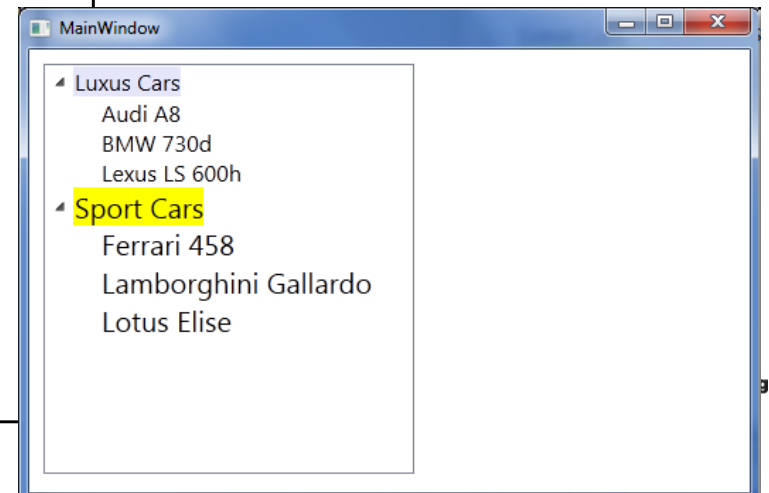
▸ Bands  
Audi  
BMW  
Ferrari  
Mercedes  
Nissan  
Peugeot  
Porsche

## TreeView

### ■ Beispiel: Navigationsbaum mit Fahrzeugen - XAML

```
<TreeView Margin="10,10,0,0" Name="tvCars"
  HorizontalAlignment="Left" VerticalAlignment="Top" Width="262" Height="289">

  <TreeViewItem Header="Luxus Cars" FontSize="16" Background="Lavender">
    <TreeViewItem Header="Audi A8"></TreeViewItem>
    <TreeViewItem Header="BMW 730d "></TreeViewItem>
    <TreeViewItem Header="Lexus LS 600h"></TreeViewItem>
  </TreeViewItem>
  <TreeViewItem Header="Sport Cars" FontSize="20" Background="Yellow">
    <TreeViewItem Header="Ferrari 458"></TreeViewItem>
    <TreeViewItem Header="Lamborghini Gallardo"></TreeViewItem>
    <TreeViewItem Header="Lotus Elise"></TreeViewItem>
  </TreeViewItem>
</TreeView>
```



## Aussehen der Controls verändern

### ■ Control-Templates

- Styles werden als Ressourcen deklariert und können von WPF-Controls referenziert werden.
- So kann z.B. das Aussehen eines Knopfes angepasst werden



- Quelle: <http://jobijoy.blogspot.com/2008/07/cool-control-template-for-wpf-button.html>

## Aussehen der Controls verändern

### ■ Beispiel: XAML (Ausschnitt)

ControlTemplate (Vorlage)

```
<Page.Resources>
  <ControlTemplate x:Key="ButtonControlTemplate1" TargetType="{x:Type Button}">
    <Grid Width="Auto" Height="Auto">
      <Grid.RowDefinitions>
        <RowDefinition Height="0.1*" />
        <RowDefinition Height="0.8*" />
        <RowDefinition Height="0.1*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="0.1*" />
        <ColumnDefinition Width="0.8*" />
        <ColumnDefinition Width="0.1*" />
      </Grid.ColumnDefinitions>
      <Ellipse StrokeThickness="0" Grid.ColumnSpan="3" Grid.RowSpan="3">
        <Ellipse.Fill>
          <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="#FF000000" Offset="0" />
            <GradientStop Color="#FFFFFFFF" Offset="0.472" />
            <GradientStop Color="#FF000000" Offset="1" />
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
      <Ellipse StrokeThickness="0" Grid.ColumnSpan="3" Grid.RowSpan="3" Margin="1
```



Referenz auf Ressource

```
<Button Content="Normal" Template="{DynamicResource ButtonControlTemplate1}"
  HorizontalAlignment="Stretch" Width="Auto" FontSize="15" Grid.ColumnSpan="1" Grid.Row="3"/>
```

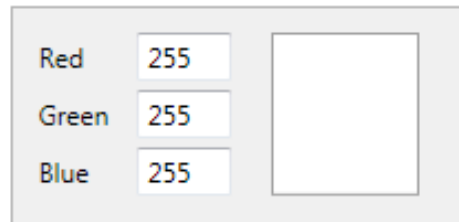
Höhere Berufsbildung  
Uster

**HBU**

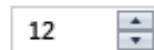
Folie 44

## UserControls vs. CustomControls

- UserControls fassen mehrere Controls zu einer wiederverwendbaren Einheit zusammen (Composition)



- CustomControls erweitern ein bestehendes Control um weitere Funktionen (Extension)



## Flexible Komposition

- Die wichtigste Aufgabe von WPF-Elementen ist die Darstellung von Inhalt
- Inhalt können Daten oder andere Elemente sein
- Dies erlaubt eine sehr flexible Komposition
- Deutlich weniger UserControls werden benötigt

```
<Button Content="Hello" />
```

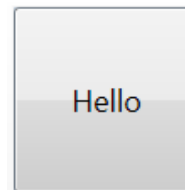
```
<Button>
```

```
    <Button.Content>
```

```
        <Image Source="winlogo.png" />
```

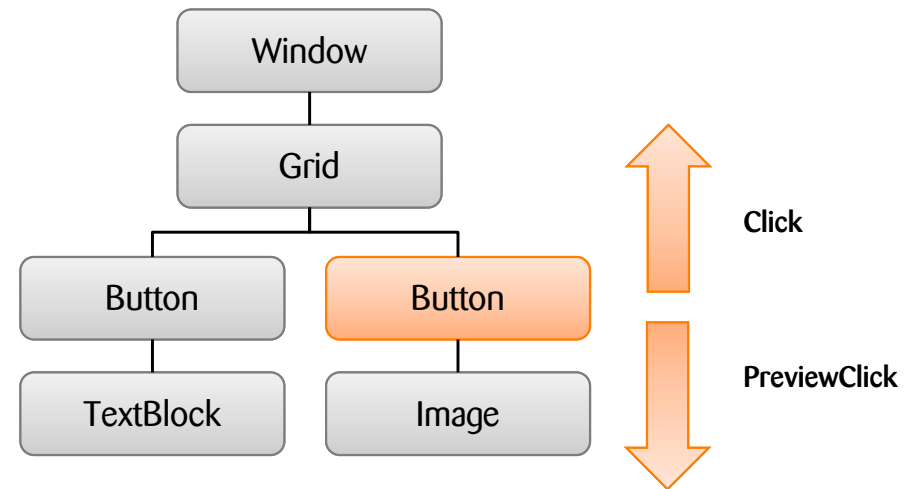
```
    </Button.Content>
```

```
</Button>
```



# Routed Events

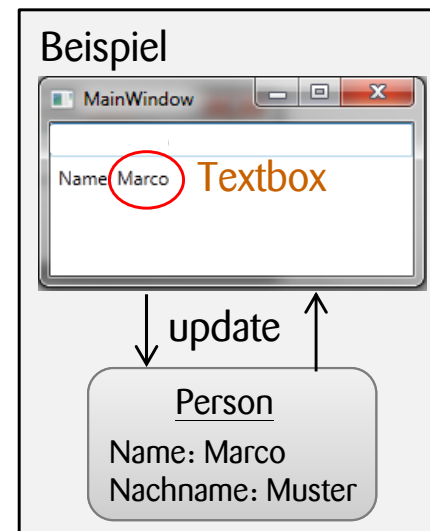
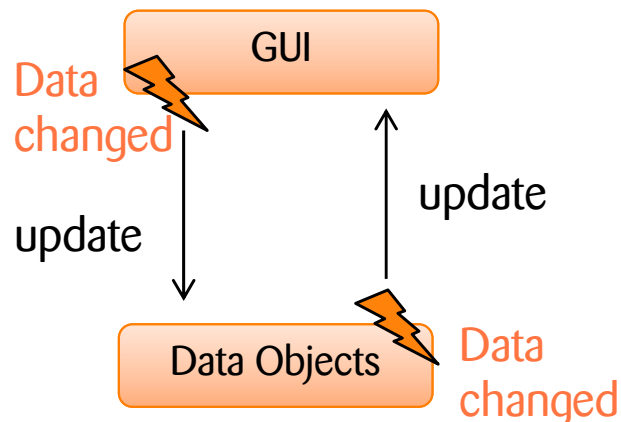
- Routed Events werden auf einem WPF-Element ausgelöst und traversieren den LogicalTree nach oben und unten
- Und einen PreviewEvent mit der RoutingStrategy Tunneling (Event steigt hinunter)
- Es gibt jeweils einen Event mit der RoutingStrategy Bubbling (Event steigt bis zum obersten Element)
- Ermöglicht ein Event in mehreren GUI-Elementen im jeweiligen Handler zu verarbeiten



# Data-binding

## Was ist Data-Binding?

- Verbindung zwischen GUI und Datenobjekte
- Notifikation bei Änderung
  - Änderungen in Datenobjekt → GUI notifizieren
  - Änderungen in GUI → Datenobjekt aktualisieren
- WPF-Data-Binding wird realisiert durch:
  - WPF-GUI: Dependency-Properties
  - Data-Objects: CLR-Objects / XML-Data

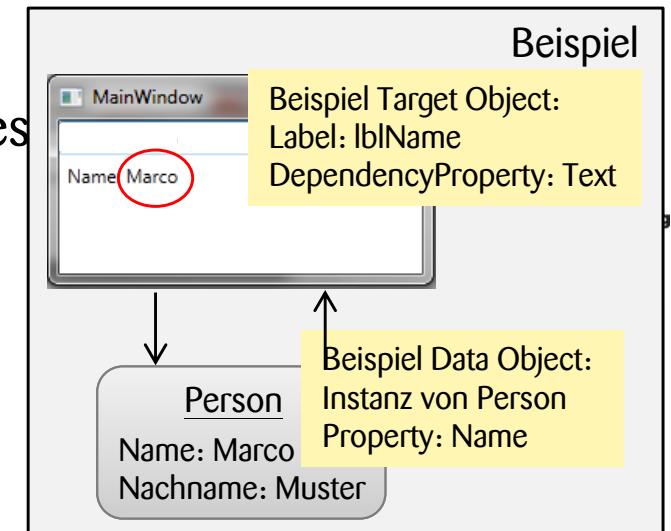
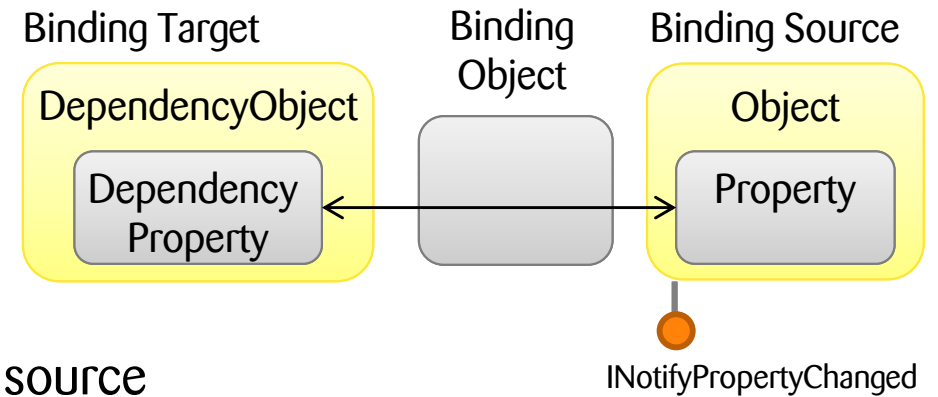




# Data-binding

## Grundlegende Data-Binding-Konzepte

- Data-Bindings bestehen i.d.R. aus
  - Ziel-Objekt: binding target object
  - Ziel-Eigenschaft: target property
  - Binding-Quelle: binding source
  - Pfad zum Wert: path to the value in the binding source
- Target property
  - Muss ein Dependency-Property sein
  - Die meisten UIElement-Properties sind Dependency-Properties
  - Nur Klassen, die von DependencyObject erben, können Dependency-Properties definieren
- Mögliche Binding-Objects
  - CLR/ADO.NET/XML und Dependency Objects
- INotifyPropertyChanged
  - Source-Objekt muss INotifyPropertyChanged implementieren



## Datenflussrichtungen

### ■ Property: Mode

- Definiert die Richtung des Data-Bindings

### ■ Enumeration:

```
public enum BindingMode
```

```
Namespace: System.Windows.Data
```

### ■ OneWay

- Änderungen in der Source werden im Target automatisch aktualisiert

### ■ TwoWay

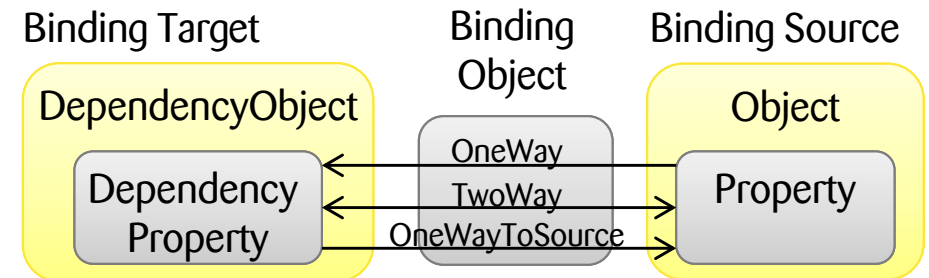
- Änderungen in der Source oder Target werden autom. im anderen Objekt aktualisiert

### ■ OneWayToSource

- Änderungen im Target werden in der Source automatisch aktualisiert

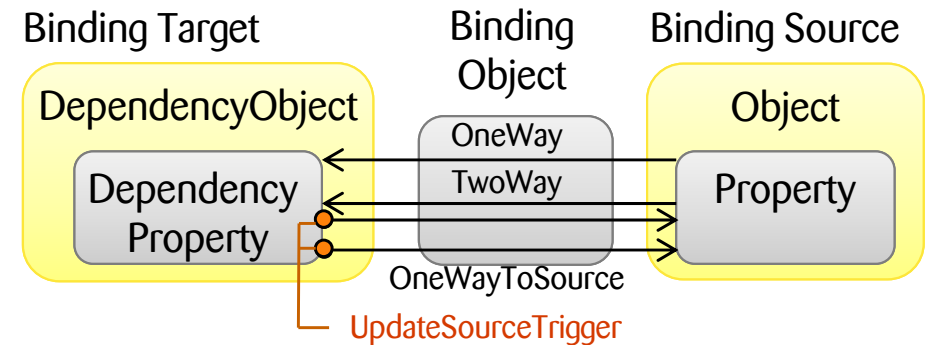
### ■ OneTime

- Das Target wird von der Source initialisiert, aber nicht weiter aktualisiert



## Update-Triggers

- **Property: UpdateSourceTrigger**
  - Definiert den Modus für die Aktualisierung der Source
- **Beispiel: Aktualisieren Textbox → Source**



UpdateSource Trigger Wert	Die Source wird aktualisiert	Beispielanwendung
LostFocus	Wenn die Textbox den Fokus verliert	Textbox mit einer Validierungslogik
PropertyChanged	Wenn in der Textbox getippt wird	Textbox für eine Chat-Applikation
Explicit	Wenn die Applikation UpdateSource aufruft	Formular mit mehreren Textboxen zum ausfüllen

## Einfaches Code-Beispiel (1/2)

- Klasse MyColorDataSource als DataSource
  - Button Background wird aus Property ColorName gelesen

```
namespace DataBindingBasic
{
    public class MyColorDataSource
    {
        public string ColorName { get; set; }
    }

    public partial class MainWindow : Window
    {
        public MyColorDataSource myDataSource;
        public MainWindow()
        {
            myDataSource = new MyColorDataSource() { ColorName = "Yellow" };
            InitializeComponent();
            button1.DataContext = myDataSource;
        }
    }
}
```

The diagram illustrates the data binding setup. On the left, the C# code defines a `MyColorDataSource` class with a `ColorName` property. The `MainWindow` class initializes this property to "Yellow" and sets the `DataContext` of `button1` to `myDataSource`. On the right, a visual representation of the `MainWindow` is shown as a window titled "Basic Databinding Sample" containing a yellow "Button". A yellow box labeled "Binding Source" points to the `MyColorDataSource` class. Another yellow box labeled "Binding Target" points to the "Button". A yellow box labeled "DataContext setzen" points to the line `button1.DataContext = myDataSource;` in the code, with an arrow indicating the flow of data from the source to the target.

Binding Source

Binding Target

DataContext setzen

Bildungszentrum Uster  
Höhere Berufsbildung  
Uster  
**HBU**

Folie 52

© M. Sabbatella

## Einfaches Code-Beispiel (2/2)

- Data-Binding auf Klasse MyColorDataSource in XAML

The screenshot displays the XAML code for a WPF application and its runtime output. The XAML code is as follows:

```
<Window x:Class="DataBindingBasic.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:c="clr-namespace:DataBindingBasic"
  Title="MainWindow" Height="163" Width="417">
  <Grid>
    <Grid.Resources>
      <c:MyColorDataSource x:Key="mySource"/>
    </Grid.Resources>
    <Button DataContext="mySource" Background="{Binding Path=ColorName, Mode=OneWay}"
      Content="Button" Height="41" HorizontalAlignment="Left"
      Margin="81,43,0,0" Name="button1" VerticalAlignment="Top" Width="243">
    </Button>
    <Label Content="Basic Databinding Sample" Height="28"
      HorizontalAlignment="Left" Margin="81,0,0,0"
      Name="label1" VerticalAlignment="Top" Width="243" />
  </Grid>
</Window>
```

Annotations in the image highlight the following parts of the code:

- Referenz auf Namespace:** Points to the `xmlns:c="clr-namespace:DataBindingBasic"` attribute.
- Binding Source & Key:** Points to the `<c:MyColorDataSource x:Key="mySource"/>` resource declaration.
- Binding Target:** Points to the `DataContext="mySource"` attribute on the `<Button>` element.
- Target Property & Source Path:** Points to the `Background="{Binding Path=ColorName, Mode=OneWay}"` attribute on the `<Button>` element.

The runtime output shows a window titled "MainWindow" with the text "Basic Databinding Sample" and a yellow button labeled "Button".

## Datenquellen

- Andere WPF-Elemente

```
{Binding Path=Text, ElementName=textBox}
```

- Übergeordnete WPF-Elemente

```
{Binding Path=Text,  
    RelativeSource={RelativeSource  
        Mode=FindAncestor,  
        AncestorType=ListBox}}
```

- DataContext

```
{Binding Text}
```

- Explizite Objekte

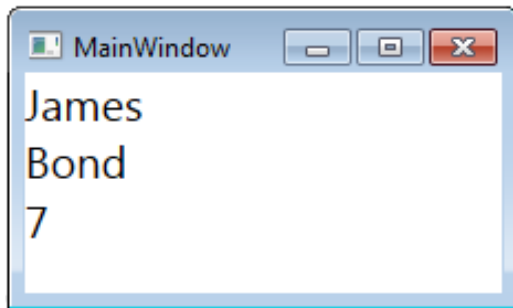
```
{Binding Path=Text Source={StaticResource myObject}}
```

# Data-binding-Samples

---

## Einfaches Binden einer Datenquelle (1/2)

- Ziel: Anzeige einer Person auf GUI



- Vorlage der Datenquelle ist die Klasse `Person`

```
namespace WpfDataBindingSamples
{
    public class Person
    {
        public string Name { get; set; }
        public string FirstName { get; set; }
        public int LuckyNumber { get; set; }
    }
}
```

# Data-binding-Samples

## Einfaches Binden einer Datenquelle (2/2)

- Nur XAML-Code, kein C#-Code-Behind

```
<Window x:Class="WpfDataBindingSamples.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:src="clr-namespace:WpfDataBindingSamples"
  Title="MainWindow" Height="153" Width="189" FontSize="20">
  <Window.Resources>
    <src:Person x:Key="MyPerson" FirstName="James" Name="Bond" LuckyNumber="007"/>
  </Window.Resources>
  <Grid >
    <StackPanel>
      <TextBlock Text="{Binding Source={StaticResource MyPerson}, Path=FirstName}" />
      <TextBlock Text="{Binding Source={StaticResource MyPerson}, Path=Name}" />
      <TextBlock Text="{Binding Source={StaticResource MyPerson}, Path=LuckyNumber}" />
    </StackPanel>
  </Grid>
</Window>
```

Namespace referenzieren, um Klasse Person benutzen zu können

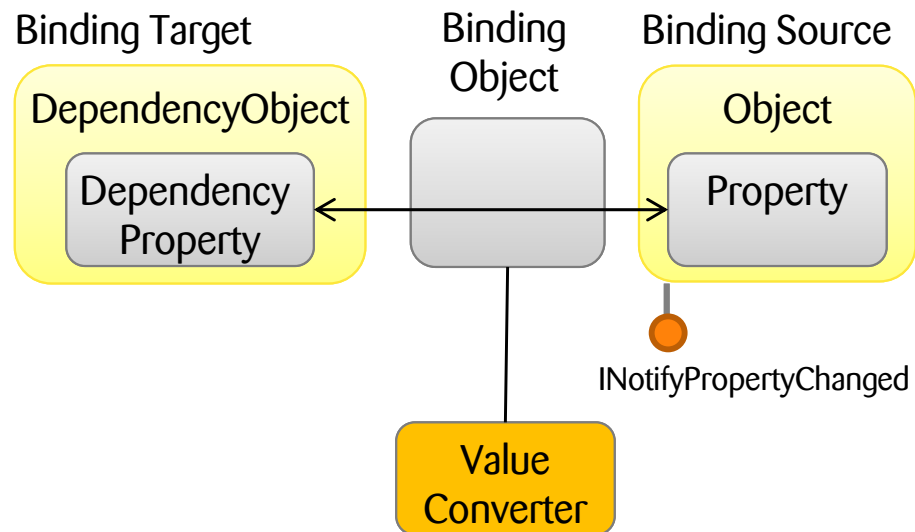
Person-Objekt instanziiieren

Data-Binding auf MyPerson



# Value-Converter

- Ein Value-Converter kann unpassende Datenformate in beide Richtungen konvertieren
- Er wird im Binding-Kontext gesetzt



## Interface

- Ein Value-Converter muss das Interface `IValueConverter` implementieren

Methoden	Beschreibung
Convert	Konvertiert den Wert von der Datenquelle (Binding Source) zum GUI (Binding Target).
ConvertBack	Konvertiert den Wert zurück vom GUI (Binding Target) zur Datenquelle (Binding Source).

# Value-Converter

## Code-Beispiel

```
<Button Content="{Binding Name, Converter={h:BoolToVisibilityConverter}}" />
```

XAML

```
public class BoolToVisibilityConverter : IValueConverter
{
    #region IValueConverter Members

    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return (bool) value ? Visibility.Visible : Visibility.Collapsed;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

C#

## Einfache Validierung von Benutzereingaben

### ■ Control-Template

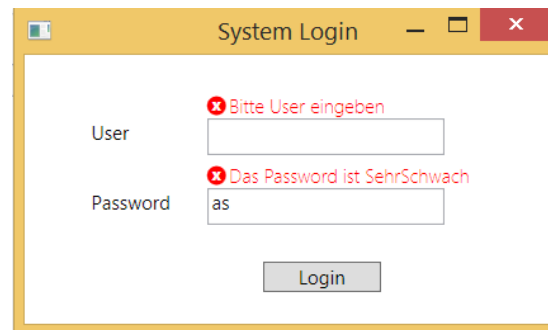
- Zur Darstellung der Fehlermeldung muss ein `ControlTemplate` erstellt werden

### ■ Validation-Rule

- Zur Prüfung der Eingabe muss eine Validator-Klasse implementiert werden, welche von der abstrakten Klasse `ValidationRule` ableitet
- Die Methode `ValidationResult.Validate()` überschreiben

### ■ Beispiel: Login-Dialog

- Validator für User
- Validator für Passwort
- Gemeinsames `ControlTemplate` für Fehlermeldung



# Eingaben validieren

---

## LoginUserValidator

```
namespace ValidatorSample
{
    public class LoginUserValidator : ValidationRule
    {
        public override ValidationResult Validate(object value, System.Globalization.CultureInfo cultureInfo)
        {
            string inputString = value == null ? string.Empty : value.ToString().Trim();

            if (string.IsNullOrEmpty(inputString))
                return new ValidationResult(false, "Bitte User eingeben");

            if (inputString.Length > 8)
                return new ValidationResult(
                    (false, "User kann maximal 8 Zeichen lang sein.");

            return ValidationResult.ValidResult;
        }
    }
}
```

# Eingaben validieren

## LoginPasswordValidator (1/2)

```
using System;
using System.Windows.Controls;
using System.Text.RegularExpressions;

namespace ValidatorSample
{
    public enum PasswordScore
    {
        Leer = 0,
        SehrSchwach = 1,
        Schwach = 2,
        Mittel = 3,
        Stark = 4,
        SehrStark = 5
    }

    public class LoginPasswordValidator : ValidationRule
    {
        public override ValidationResult Validate(object value, System.Globalization.CultureInfo cultureInfo)
        {
            string inputString = value == null ? String.Empty : value.ToString().Trim();

            if (String.IsNullOrEmpty(inputString))
                return new ValidationResult(false, "Passwort darf nicht leer sein.");

            PasswordScore score = CheckStrength(inputString);
            if (score < (PasswordScore) 3)
            {
                return new ValidationResult(false, String.Format("Das Passwort ist {0}", score));
            }

            return ValidationResult.ValidResult;
        }
    }
}
```

Passwortstärken

Passwort-Validator-Klasse

# Eingaben validieren

## LoginPasswordValidator (2/2)

```
public static PasswordScore CheckStrength(string password)
{
    int score = 1;

    if (password.Length < 1)
        return PasswordScore.Leer;
    if (password.Length < 4)
        return PasswordScore.SehrSchwach;

    if (password.Length >= 8)
        score++;
    if (password.Length >= 12)
        score++;
    if (Regex.Match(password, @"\/\d+\/", RegexOptions.ECMAScript).Success)
        score++;
    if (Regex.Match(password, @"\/[a-z]\/", RegexOptions.ECMAScript).Success &&
        Regex.Match(password, @"\/[A-Z]\/", RegexOptions.ECMAScript).Success)
        score++;
    if (Regex.Match(password, @"\/.[!,@,#,$,%,&,*?,_~, -,£,(,)]\/", RegexOptions.ECMAScript).Success)
        score++;

    return (PasswordScore)score;
}
```

Passwortstärke überprüfen

# Eingaben validieren

## View (1/2)

```
<Window x:Class="ValidatorSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:ValidatorSample"
        Title="System Login" Height="207.075" Width="347.033">
    <Window.DataContext>
        <local:CredentialsViewModel></local:CredentialsViewModel>
    </Window.DataContext>
    <Window.Resources>
        <ControlTemplate x:Key="ValidationErrorTemplate">
            <DockPanel>
                <StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
                    <Grid Width="12" Height="12">
                        <Ellipse Width="12" Height="12" Fill="Red" HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        <TextBlock Foreground="White" FontWeight="Heavy"
                                FontSize="8" HorizontalAlignment="Center"
                                VerticalAlignment="Center" TextAlignment="Center"
                                Tooltip="{Binding ElementName=ErrorAdorner,
                                Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"
                                >X</TextBlock>
                    </Grid>
                    <TextBlock Foreground="Red" FontWeight="12" Margin="2,0,0,0"
                                Text="{Binding ElementName=ErrorAdorner,
                                Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"
                                ></TextBlock>
                </StackPanel>
                <AdornedElementPlaceholder x:Name="ErrorAdorner" ></AdornedElementPlaceholder>
            </DockPanel>
        </ControlTemplate>
    </Window.Resources>
```

Data Context

Control Template

Weisses X

Roten Kreis  
zeichnen

Tooltip Text = Error-Meldung

Text = Error-Meldung



# Eingaben validieren

## View (2/2)

```
<Grid>
  <Canvas HorizontalAlignment="Left" Height="149" Margin="9,10,0,-65.4" VerticalAlignment="Top" Width="328">
    <Label Content="User" Canvas.Left="28" Canvas.Top="26" Width="61"/>
    <Label Content="Password" Canvas.Left="28" Canvas.Top="70"/>
    <TextBox Name="TextBoxUser" Validation.ErrorTemplate="{StaticResource ValidationErrorTemplate}"
      Height="23" Canvas.Left="106" TextWrapping="Wrap" Canvas.Top="30" Width="150">
      <TextBox.Text>
        <Binding Path="User" Mode="TwoWay" UpdateSourceTrigger="LostFocus">
          <Binding.ValidationRules>
            <local:LoginUserValidator></local:LoginUserValidator>
          </Binding.ValidationRules>
        </Binding>
      </TextBox.Text>
    </TextBox>
    <TextBox Name="TextBoxPassword" Validation.ErrorTemplate="{StaticResource ValidationErrorTemplate}"
      Height="23" Canvas.Left="106" TextWrapping="Wrap" Canvas.Top="74" Width="150">
      <TextBox.Text>
        <Binding Path="Password" Mode="TwoWay" UpdateSourceTrigger="LostFocus">
          <Binding.ValidationRules>
            <local:LoginPasswordValidator></local:LoginPasswordValidator>
          </Binding.ValidationRules>
        </Binding>
      </TextBox.Text>
    </TextBox>
    <Button Name="ButtonLogin" Content="Login" Canvas.Left="141" Canvas.Top="120" Width="75"/>
  </Canvas>
</Grid>
</Window>
```

Referenz auf ValidationErrorTemplate

Data Binding auf User

Validator Binding auf LoginUserValidator

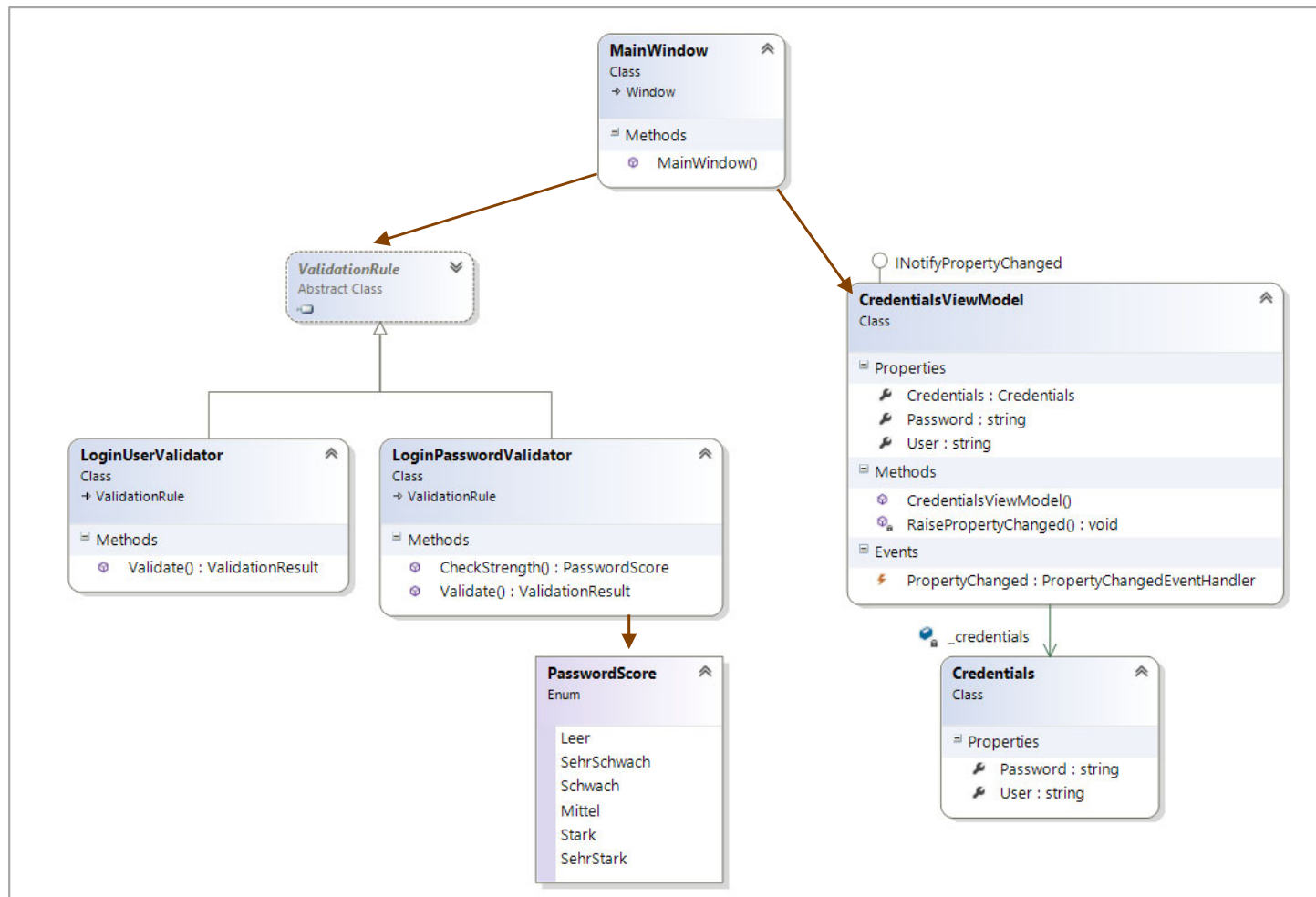
Referenz auf ValidationErrorTemplate

Data Binding auf Password

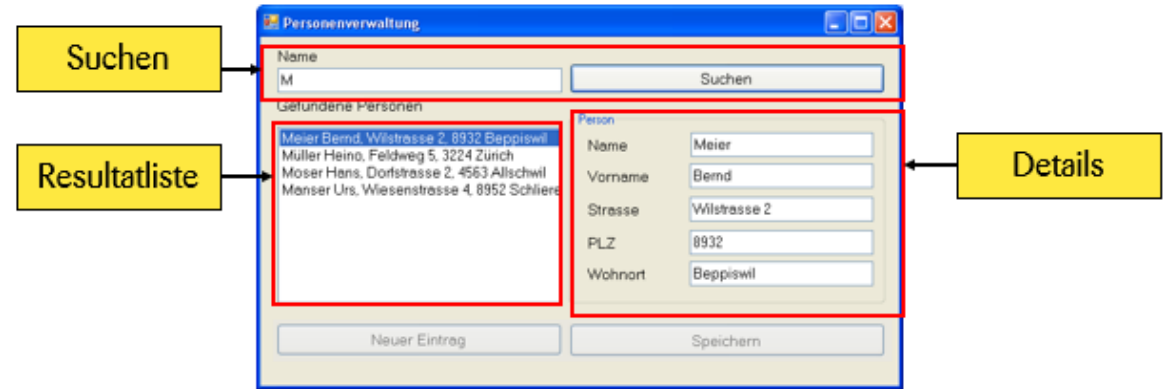
Validator Binding auf LoginPasswordValidator

# Eingaben validieren

## View (2/2)



# Übung 4.2

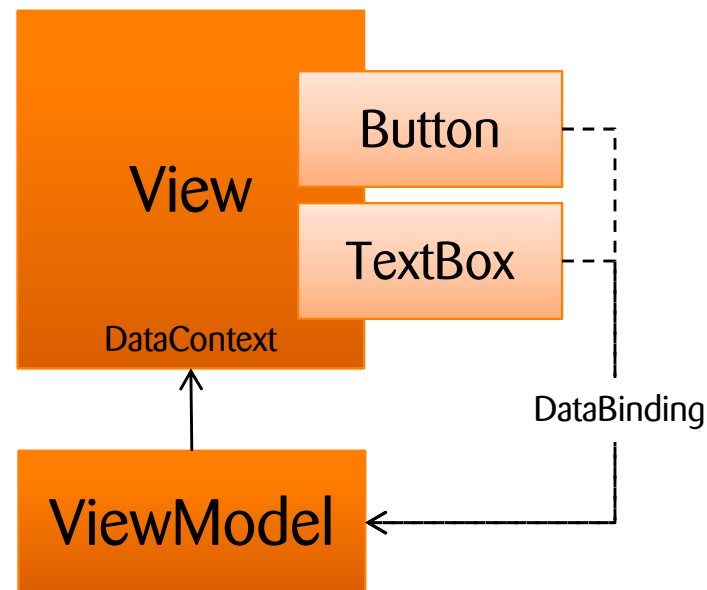


## Personalverwaltung Release 2.0

- Verwende die WinForms-Applikation «Personalverwaltung» aus dem Kapitel ADO.NET und peppe das GUI mit WPF auf, indem das WinForms-GUI mit einem WPF-GUI ersetzt wird.
- Achte beim Umbau darauf, wie viel oder wie wenig angepasst werden muss.
- Wie in Kapitel 3 Applikationsarchitektur gelernt: wurde bei Release 1.0 eine geeignete Architektur gewählt, muss jetzt beim Umbau relativ wenig angepasst werden.

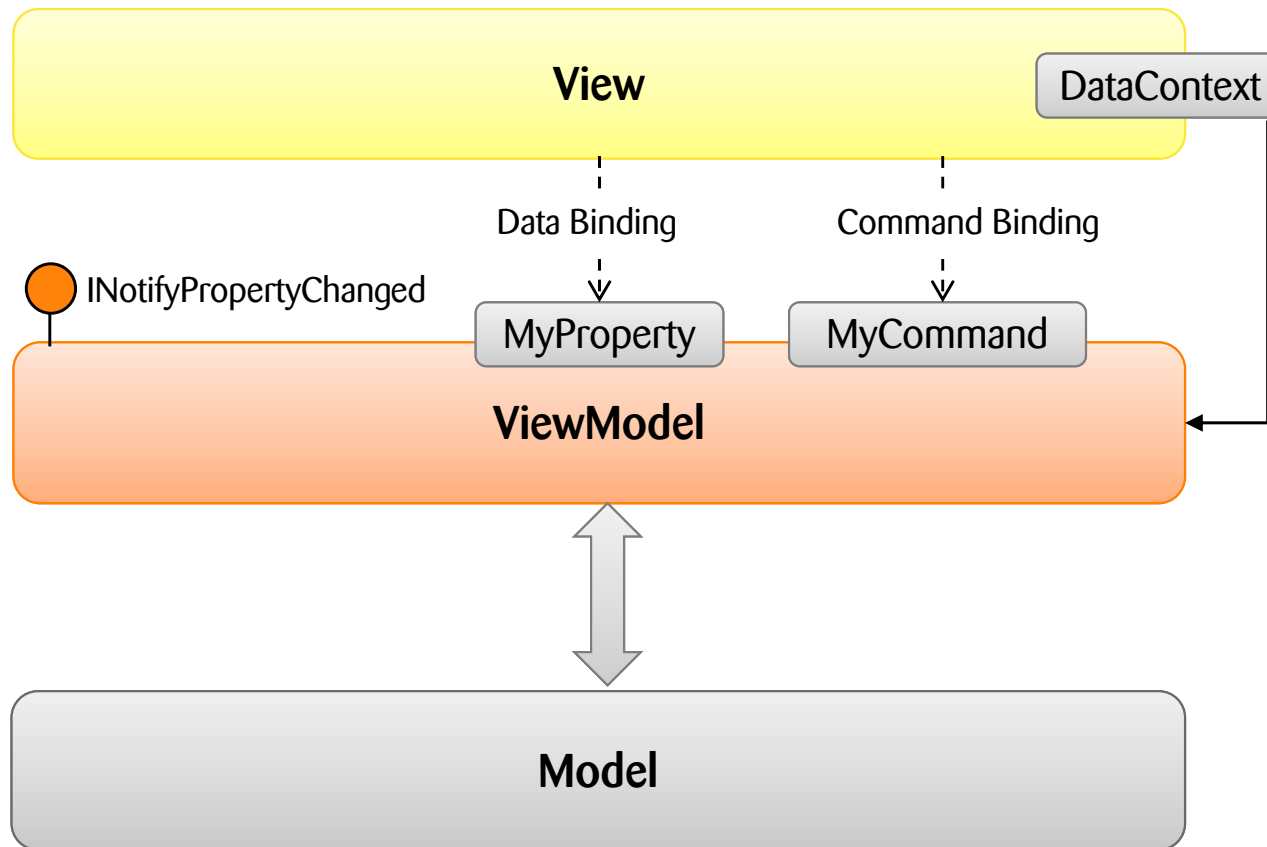
# Das Model-View-ViewModel-Pattern

- Ein WPF-Pattern zum Anbinden von Daten an die View
- Entstehung:
  - Jedes Element hat einen DataContext
  - Man möchte oft mehrere Datenobjekte in einer View anzeigen
  - Das ViewModel aggregiert diese zu einem Objekt, das dann über den DataContext zugänglich ist
- Vorteile:
  - Saubere Trennung von Präsentation und Logik
  - UI-Logik wird durch Unittests testbar



# Das Model-View-ViewModel-Pattern

## Das MVVM – Model-View-ViewModel



# Commands in WPF

---

- Commands vereinfachen das Binden von Funktionen an das UI
- Ein Command kapselt folgende 3 Funktionen in das ICommand Interface:

`Execute(object param) ;`

`CanExecute(object param) ;`

`event CanExecuteChanged;`

- Commands können flexibel an diverse UI-Elemente gebunden werden (Button, Menu, KeyboardShortcut)
- `IsEnabled` des UI-Elements wird automatisch gesteuert

# Möglichkeit 1 :

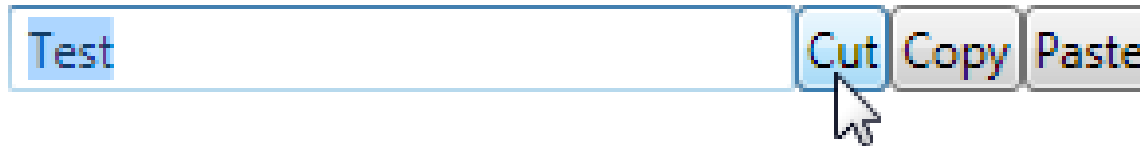
## Vordefinierte Commands von WPF verwenden

---

<b>ApplicationCommands</b>	Close, Copy, Cut, Delete, Find, Help, New, Open, Paste, Print, PrintPreview, Properties, Redo, Replace, Save, SaveAs, SelectAll, Stop, Undo
<b>ComponentCommands</b>	MoveDown, MoveLeft, MoveRight, MoveUp, ScrollByLine, ScrollPageDown, ScrollPageLeft, ScrollPageRight, ScrollPageUp, SelectToEnd, SelectToHome, SelectToPageDown, SelectToPageUp
<b>MediaCommands</b>	ChannelDown, ChannelUp, DecreaseVolume, FastForward, IncreaseVolume, MuteVolume, NextTrack, Pause, Play, PreviousTrack, Record, Rewind, Select, Stop
<b>NavigationCommands</b>	BrowseBack, BrowseForward, BrowseHome, BrowseStop, Favorites, FirstPage, GoToPage, LastPage, NextPage, PreviousPage, Refresh, Search, Zoom
<b>EditingCommands</b>	AlignCenter, AlignJustify, AlignLeft, AlignRight, CorrectSpellingError, DecreaseFontSize, DecreaseIndentation, EnterLineBreak, EnterParagraphBreak, IgnoreSpellingError, IncreaseFontSize, IncreaseIndentation, MoveDownByLine, MoveDownByPage, MoveDownByParagraph, MoveLeftByCharacter, MoveLeftByWord, MoveRightByCharacter, MoveRightByWord

# Möglichkeit 1 : Vordefinierte Commands von WPF verwenden

---



```
<TextBox x:Name="textBox" Width="200" />
```

```
<Button Command="Cut"  
        CommandTarget="{Binding ElementName=textBox}"  
        Content="Cut" />
```

```
<Button Command="Copy"  
        CommandTarget="{Binding ElementName=textBox}"  
        Content="Copy" />
```

```
<Button Command="Paste"  
        CommandTarget="{Binding ElementName=textBox}"  
        Content="Paste" />
```



# Möglichkeit 2: Eigene Commands implementieren

## ■ Selber das ICommand Interface implementieren

```
<Button Content="Say Hello!" Command="{Binding HelloCommand}" />
```

```
public partial class Window1 : Window
{
    public ICommand HelloCommand { get; set; }

    public Window1()
    {
        InitializeComponent();
        DataContext = this;

        HelloCommand = new DelegateCommand {ExecuteDelegate = SayHello};
    }

    private void SayHello(object parameter)
    {
        MessageBox.Show("Hello WPF!");
    }
}
```

Bildungszentrum Uster  
Höhere Berufsbildung  
Uster

**HBU**

Folie 73

# Universell und nützlich: Das RelayCommand

---

- Das RelayCommand ist eine generische Implementierung des Command Patterns

```
public class RelayCommand : ICommand
{
    public Action<object> ExecuteDelegate { get; set; }
    public Predicate<object> CanExecuteDelegate { get; set; }

    #region ICommand Members

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

    public event EventHandler CanExecuteChanged;

    public void Execute(object parameter)
    {
        ExecuteDelegate(parameter);
    }

    #endregion
}
```

## Was man wissen muss

### ■ **INotifyPropertyChanged**

- Fürs Data-Binding müssen Datenobjekte das Interface `INotifyPropertyChanged` implementieren

### ■ **DataContext**

- Alle WPF-Controls haben einen `DataContext` und Controls mit mehreren Elementen (z.B. `ListBox`) haben ein Attribut `ItemSource`

### ■ **ObservableCollections**

- Datenobjekte, die in Listen angezeigt werden sollen, werden in `ObservableCollection<>` und nicht in `List<>` oder anderen Collection-Klassen

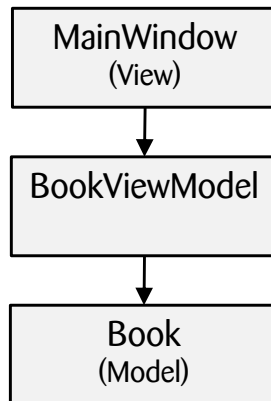
# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 1

WRONG  
WAY

### Wie es nicht funktioniert

#### ■ Architektur



#### ■ Eigenschaften

- Funktionalität: Ein Buch anzeigen und Titel ändern
- ViewModel im XAML instanziiert, **ohne** **INotifyPropertyChanged** implementiert
- Data-Binding im XAML
- DataContext im Code-Behind gesetzt

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 1

WRONG  
WAY

### ViewModel und Model

ViewModel

```
public class BookViewModel
```

```
{
```

```
    private Book _book;
```

```
    public BookViewModel()
```

```
    {
```

```
        _book = new Book() {Author = "Unbekannt", Title = "Unbekannt", Price = 0};
```

```
    }
```

```
    public Book Book
```

```
    {
```

```
        get { return _book; }
```

```
        set { _book = value; }
```

```
    }
```

```
    public string Title
```

```
    {
```

```
        get { return _book.Title; }
```

```
        set { _book.Title = value; }
```

```
    }
```

```
    public string Author
```

```
    {
```

```
        get { return _book.Author; }
```

```
        set { _book.Author = value; }
```

```
    }
```

```
    public double Price
```

```
    {
```

```
        get { return _book.Price; }
```

```
        set { _book.Price = value; }
```

```
    }
```

```
}
```

1

Model

```
public class Book
```

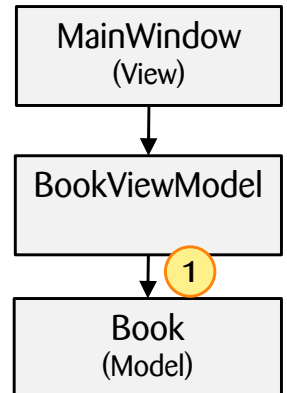
```
{
```

```
    public string Title { get; set; }
```

```
    public string Author { get; set; }
```

```
    public double Price { get; set; }
```

```
}
```



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 1

WRONG  
WAY

### View mit Bindings

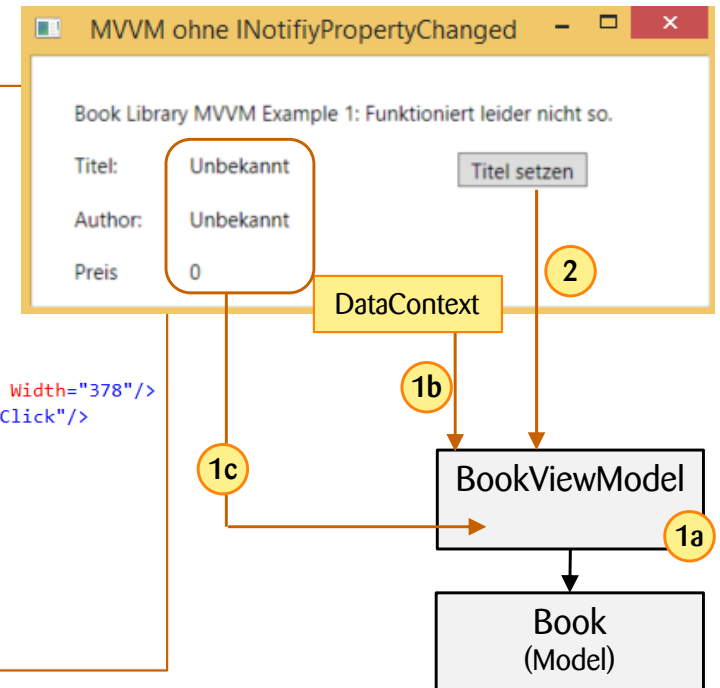
**XAML**

```
<Window x:Class="BookInventoryMvvmExample1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:BookInventoryMvvmExample1"
        Title="MVVM ohne INotifyPropertyChanged" Height="182.836" Width="378.134">
    <Window.DataContext>
        1a <!-- Wir instanzieren deklarativ ein BookViewModel und setzen den DataContext -->
        <local:BookViewModel />
    </Window.DataContext>
    <Grid>
        <Canvas HorizontalAlignment="Left" Height="134" Margin="10,10,0,0" VerticalAlignment="Top" Width="350">
            <Label Content="Book Library MVVM Example 1: Funktioniert leider nicht so." Canvas.Left="10" Canvas.Top="10" Width="378"/>
            <Button Name="SetTitle" Content="Titel setzen" Canvas.Left="238" Canvas.Top="46" Width="75" Click="SetTitle_Click"/>
            <Label Content="Titel:" Canvas.Left="10" Canvas.Top="40" Width="79"/>
            <Label Content="Author:" Canvas.Left="10" Canvas.Top="71"/>
            <Label Content="Preis" Canvas.Left="10" Canvas.Top="102"/>
            <Label Name="LabelTitle" Content="{Binding Title}" Canvas.Left="77" Canvas.Top="40" Width="138"/>
            <Label Name="LabelAuthor" Content="{Binding Author}" Canvas.Left="77" Canvas.Top="71" Width="138"/>
            <Label Name="LabelPrice" Content="{Binding Price}" Canvas.Left="77" Canvas.Top="102" Width="45"/>
        </Canvas>
    </Grid>
</Window>
```

**Code Behind**

```
namespace BookInventoryMvvmExample1
{
    public partial class MainWindow : Window
    {
        private BookViewModel _bookViewModel;
        public MainWindow()
        {
            InitializeComponent();
            // Den in XAML instanziierten DataContext referenzieren
            _bookViewModel = (BookViewModel) this.DataContext;
        }

        private void SetTitle_Click(object sender, RoutedEventArgs e)
        {
            2 _bookViewModel.Title = "MVVM - So geht's nicht!";
            // Der Titel ist auf dem BookViewModel neu gesetzt
            // aber die View wird nicht automatisch aktualisiert
        }
    }
}
```



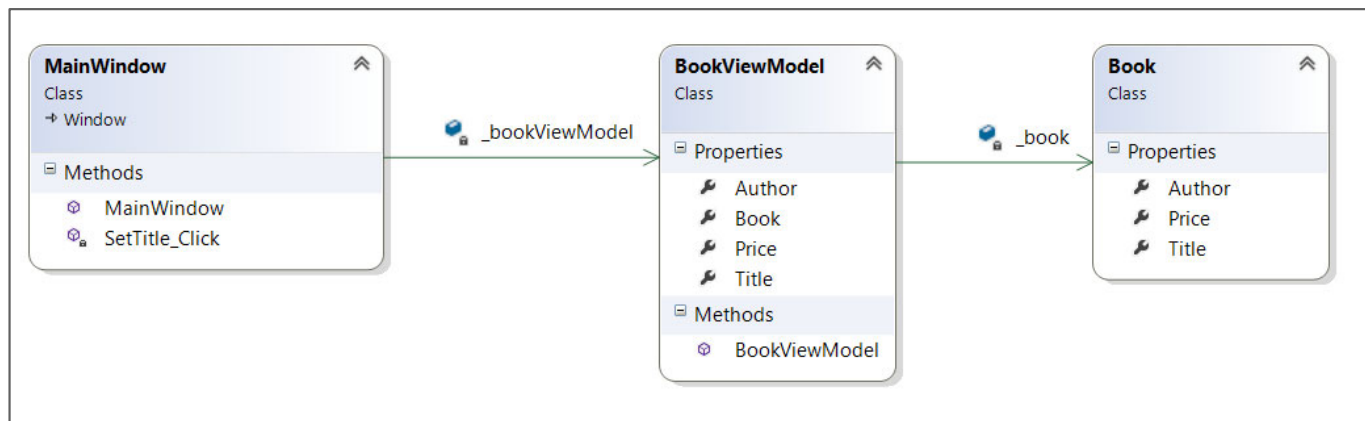
1. Data-Binding
  - a) ViewModel instanziiieren
  - b) DataContext setzen
  - c) Binding auf Properties
2. Model aktualisieren

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 1

WRONG  
WAY

### Klassendiagramm

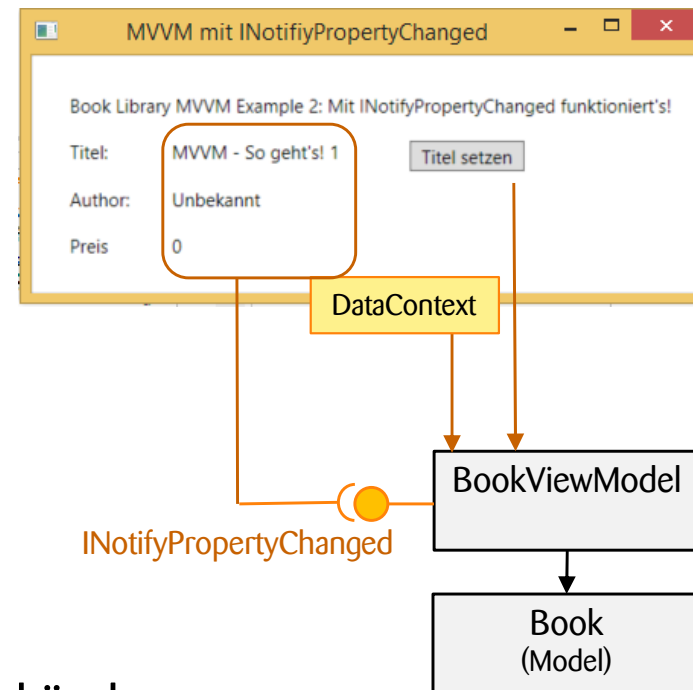
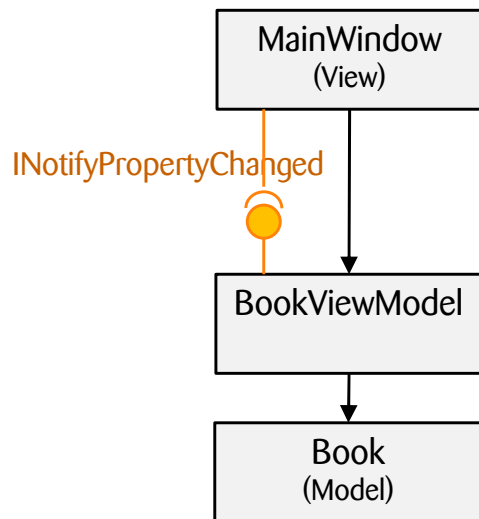


# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 2

### INotifyPropertyChanged hinzufügen

#### ■ Architektur



#### ■ Eigenschaften

- Funktionalität: Ein Buch anzeigen und Titel ändern
- ViewModel im XAML instanziiert, **INotifyPropertyChanged implementiert**
- DataContext im Code-Behind und Data-Binding im XAML



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 2

### ViewModel mit INotifyPropertyChanged

```
public class BookViewModel : INotifyPropertyChanged
{
    private Book _book;
    public BookViewModel()
    {
        _book = new Book() { Author = "Unbekannt", Title = "Unbekannt", Price = 0 };
    }

    #region Properties

    public Book Book
    {
        get { return _book; }
        set { _book = value; }
    }

    public string Title
    {
        get { return _book.Title; }
        set
        {
            _book.Title = value;
            RaisePropertyChanged("Title");
        }
    }

    public string Author
    {
        get { return _book.Author; }
        set
        {
            _book.Author = value;
            RaisePropertyChanged("Author");
        }
    }
}
```

1. INotifyPropertyChanged implementieren
2. PropertyChanged-Event deklarieren
3. Beim Setzen der Property-Werte den Event auslösen (Argument = Property-Name)
4. Methode RaisePropertyChanged implementieren

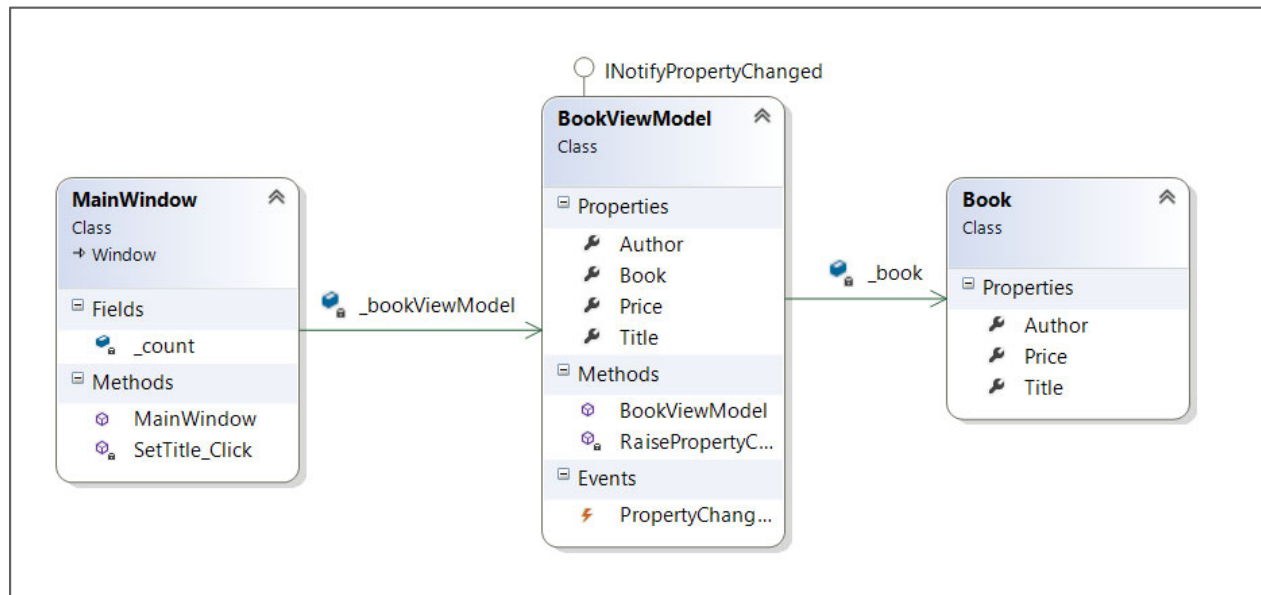
```
#region INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;
#endregion

#region Methods
private void RaisePropertyChanged(string propertyName)
{
    // Referenz auf Property Changed Handler
    PropertyChangedEventHandler handler = PropertyChanged;
    if (handler != null)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
#endregion
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Example 2

### Klassendiagramm

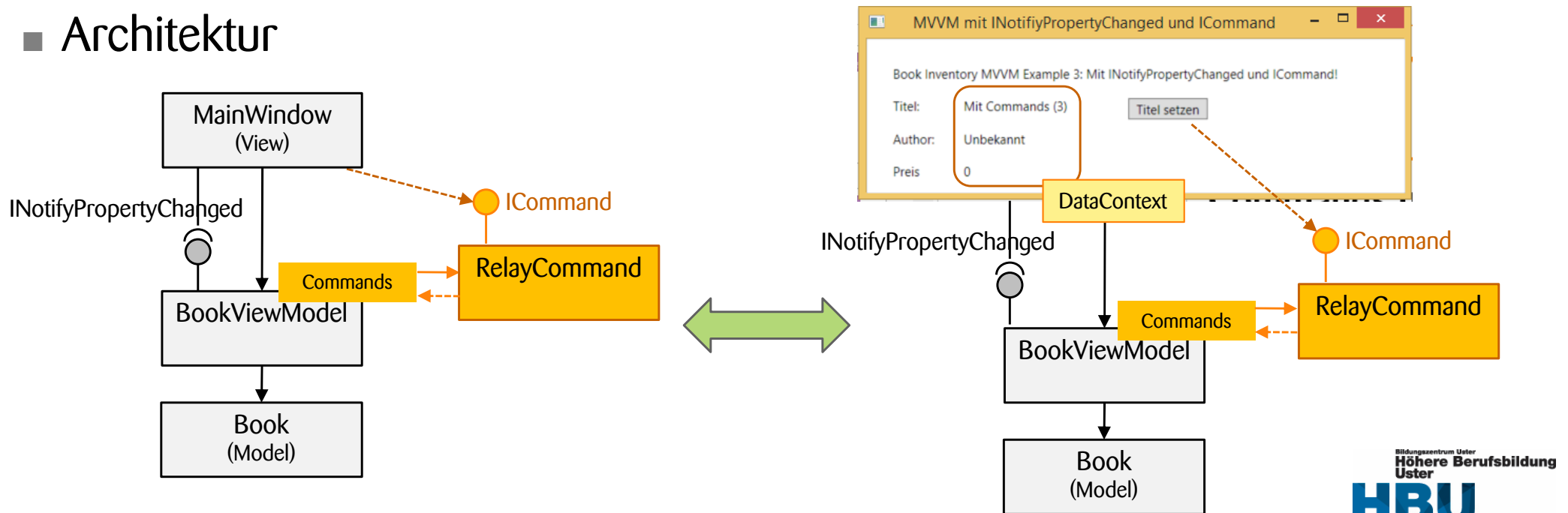


# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 3

### Commands und RelayCommand

#### ■ Architektur



#### ■ Eigenschaften

- Funktionalität: Ein Buch anzeigen und Titel ändern
- ViewModel im XAML instanziiert, **INotifyPropertyChanged** implementiert, Commands über **RelayCommand** implementiert
- **DataContext** und Data Binding im XAML

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 3

### View mit Command-Binding

XAML

```
<Canvas HorizontalAlignment="Left" Height="134" Margin="10,10,0,0" VerticalAlignment="Top" Width="498">
  <Label Content="Book Inventory MVVM Example 3: Mit INotifyPropertyChanged und ICommand!" Canvas.Left="10" Canvas.Top="10" Width="441"/>
  <Button Name="SetTitle" Content="Titel setzen" Canvas.Left="238" Canvas.Top="46" Width="75" Command="{Binding UpdateTitle}" />
  <Label Content="Titel:" Canvas.Left="10" Canvas.Top="40" Width="79"/>
  <Label Content="Author:" Canvas.Left="10" Canvas.Top="71"/>
  <Label Content="Preis" Canvas.Left="10" Canvas.Top="102"/>
  <Label Name="LabelText" Content="{Binding Title}" Canvas.Left="77" Canvas.Top="40" Width="138"/>
  <Label Name="LabelAuthor" Content="{Binding Author}" Canvas.Left="77" Canvas.Top="71" Width="138"/>
  <Label Name="LabelPrice" Content="{Binding Price}" Canvas.Left="77" Canvas.Top="102" Width="45"/>
</Canvas>
```

Code Behind

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

Click-EventHandler entfernt

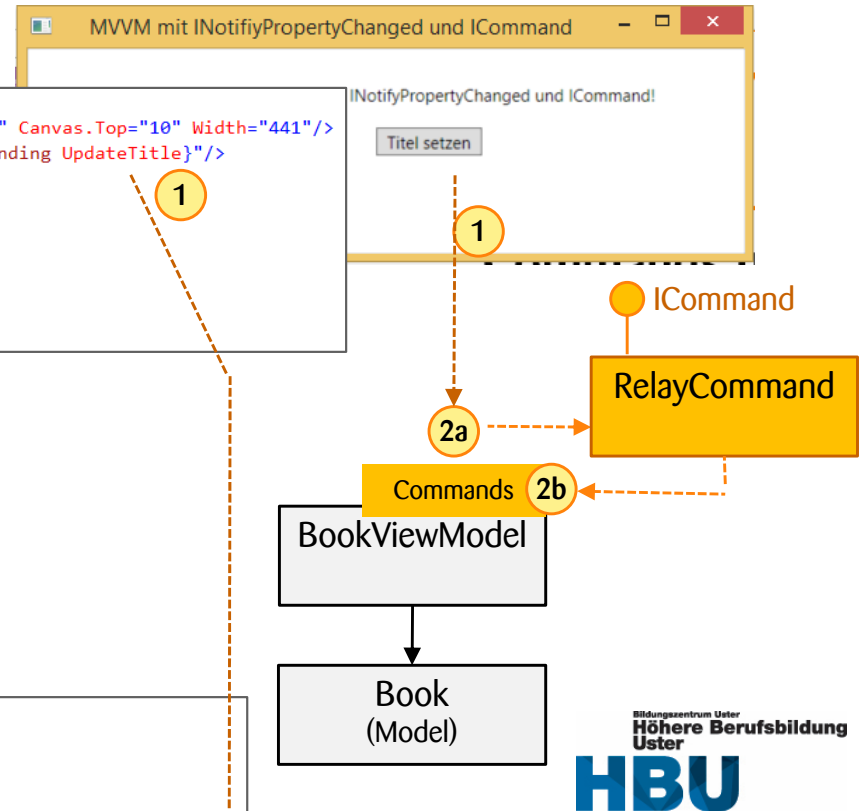
### ViewModel mit Commands

BookViewModel

```
#region Commands
void UpdateTitleExecute()
{
    _count++;
    Title = string.Format("Mit Commands ({0})", _count);
}

bool CanUpdateTitleExecute()
{
    return true;
}

public ICommand UpdateTitle { get { return new RelayCommand(UpdateTitleExecute, CanUpdateTitleExecute); } }
#endregion
```



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 3

---

### Commands

- **Attribute Command**
  - Einige WPF-Controls haben ein `Command`-Attribut
  - An dieses lassen sich Properties vom Typ `ICommand` binden
- **Interface ICommand**
  - Methode `bool CanExecute()` gibt an, ob das Command ausgeführt werden kann (wird verwendet, um z.B. den Button zu aktivieren/deaktivieren)
  - Methode `void Execute()` führt das Command aus, falls `CanExecute()` `true` zurück liefert
- **RelayCommand**
  - In dieser Klasse wird der gemeinsame Code von allen Commands implementiert

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 3

### RelayCommand

#### Properties & Constructors

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

    public RelayCommand(Action execute)
        : this(execute, null)
    {
    }

    public RelayCommand(Action execute, Func<Boolean> canExecute)
    {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
}
```

#### ICommand Members

```
#region ICommand Members
public event EventHandler CanExecuteChanged
{
    add
    {
        if (_canExecute != null)
            CommandManager.RequerySuggested += value;
    }
    remove
    {
        if (_canExecute != null)
            CommandManager.RequerySuggested -= value;
    }
}

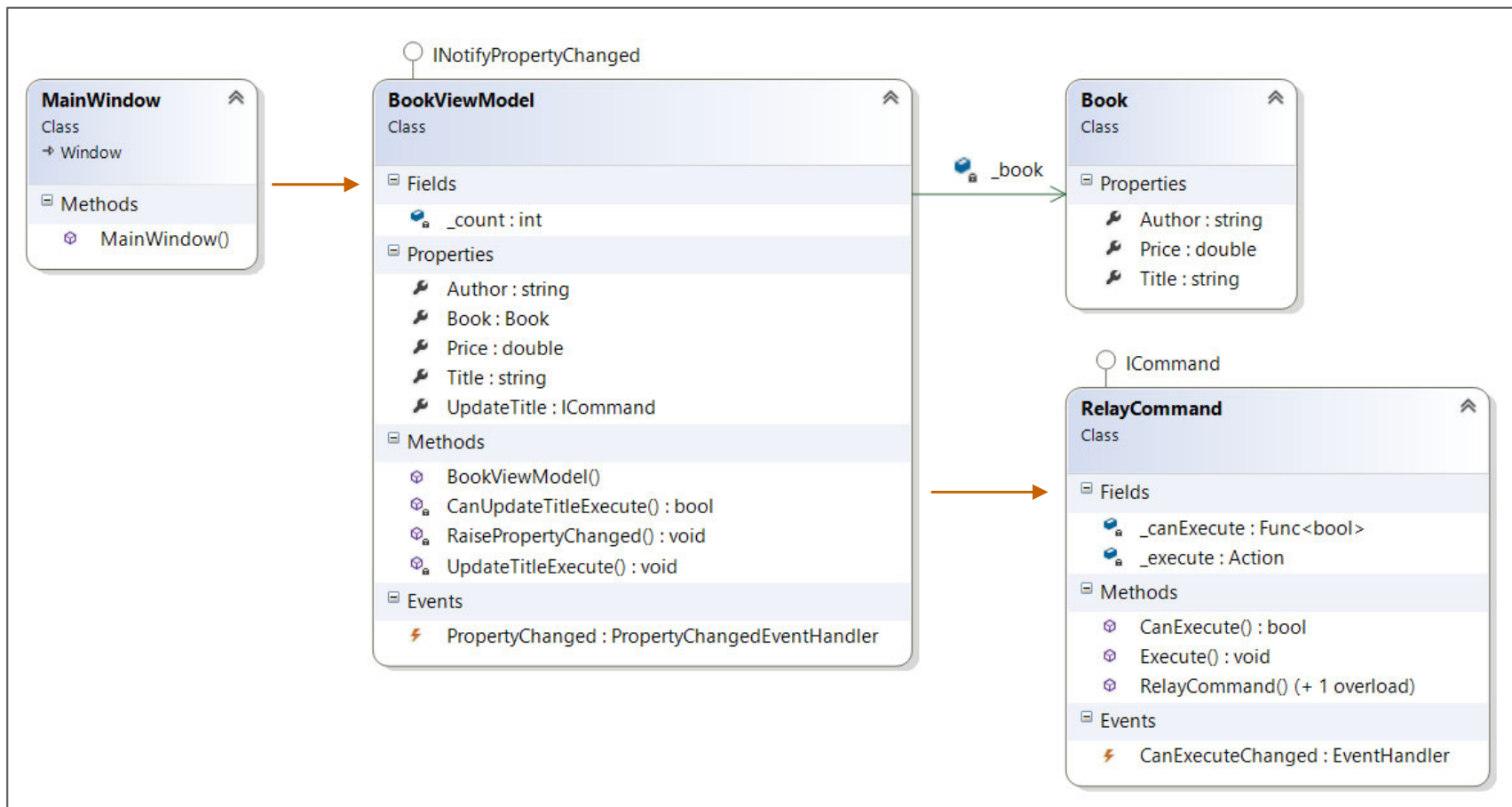
public Boolean CanExecute(Object parameter)
{
    return _canExecute == null || _canExecute();
}

public void Execute(Object parameter)
{
    _execute();
}
#endregion
}
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 3

### Klassendiagramm

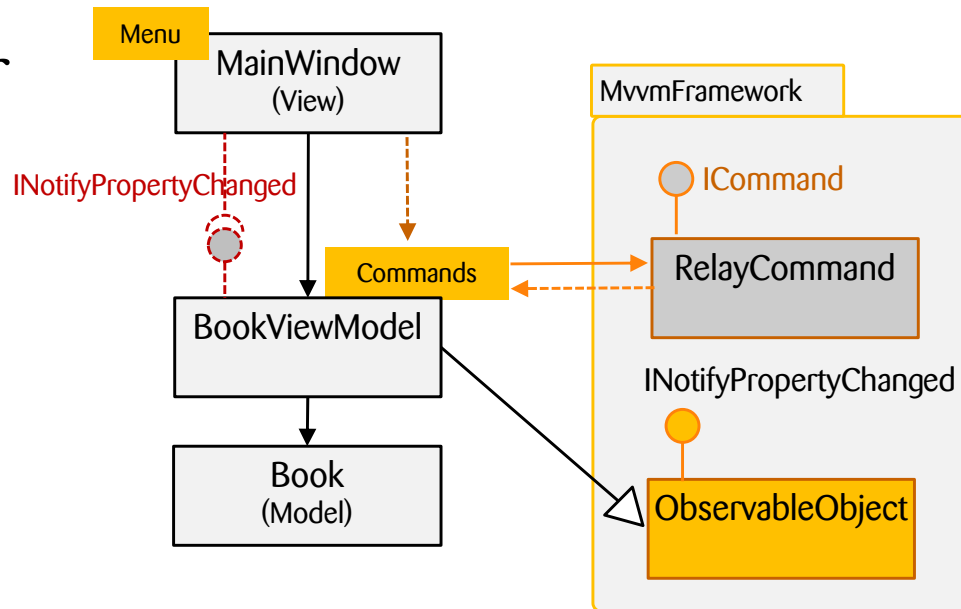


# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

### MVVM-Framework und ObservableObject

#### ■ Architektur



#### ■ Eigenschaften

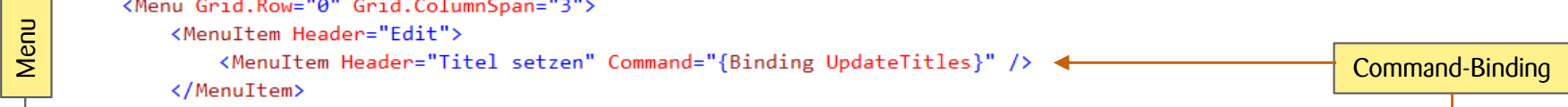
- Funktionalität: Buch anzeigen und Titel ändern **im Menu**
- ViewModel im XAML instanziiert, **erbt INotifyPropertyChanged von ObservableObject**, Commands via RelayCommand
- **Eigenes Projekt** für RelayCommand und ObservableObject
- DataContext und Data-Binding im XAML



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

### View



```
<Window.DataContext>
    <!-- Wir instanzieren deklarativ ein BookViewModel und setzen den DataContext -->
    <local:BookInventoryViewModel />
</Window.DataContext>
<Grid>
    <Menu Grid.Row="0" Grid.ColumnSpan="3">
        <MenuItem Header="Edit">
            <MenuItem Header="Titel setzen" Command="{Binding UpdateTitles}" />
        </MenuItem>
    </Menu>
    <Canvas HorizontalAlignment="Left" Height="301" Margin="10,10,0,0" VerticalAlignment="Top" Width="498">
        <Label Content="Book Inventory MVVM Example 5: Mit MvvmFramework - Title update funktioniert nicht!" Canvas.Left="10" Canvas.Top=
        <Button Name="SetTitle" Content="Titel setzen" Canvas.Left="238" Canvas.Top="46" Width="76" Command="{Binding UpdateTitles}"/>
        <Label Content="Titel:" Canvas.Left="10" Canvas.Top="40" Width="79"/>
    </Canvas>
</Grid>
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

### ObservableObject (1/2)

```
namespace MvvmCommon
{
    [Serializable]
    public abstract class ObservableObject : INotifyPropertyChanged
    {
        [field: NonSerialized]
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(PropertyChangedEventArgs e)
        {
            var handler = this.PropertyChanged;
            if (handler != null)
            {
                handler(this, e);
            }
        }

        protected void RaisePropertyChanged<T>(Expression<Func<T>> propertyExpression)
        {
            var propertyName = PropertyHelper.ExtractPropertyName(propertyExpression);
            this.RaisePropertyChanged(propertyName);
        }

        protected void RaisePropertyChanged(String propertyName)
        {
            VerifyPropertyName(propertyName);
            OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Generische Implementierung  
benutzt PropertyHelper

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

---

### ObservableObject (2/2)

Fürs Debugging

```
/// <summary>
/// Warnt den Compiler, falls es zum spezifizierten Namen kein Property im Objekt existiert.
/// Diese Methode ist nur für Debug Build.
/// </summary>
[Conditional("DEBUG")]
[DebuggerStepThrough]
public void VerifyPropertyName(String propertyName)
{
    // Sicherstellen, dass der Property Name auf dem Objekt existiert
    if (PropertyDescriptor.GetProperties(this)[propertyName] == null)
    {
        Debug.Fail("Invalid property name: " + propertyName);
    }
}
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

---

### PropertyHelper – verwendet Reflection

```
using System;
using System.Linq.Expressions;
using System.Reflection;

namespace MvvmCommon
{
    public static class PropertyHelper
    {
        public static string ExtractPropertyName<T>(Expression<Func<T>> propertyExpression)
        {
            if (propertyExpression == null)
            {
                throw new ArgumentNullException("propertyExpression");
            }

            var memberExpression = propertyExpression.Body as MemberExpression;
            if (memberExpression == null)
            {
                throw new ArgumentException("The expression is not a member access expression.", "propertyExpression");
            }

            var property = memberExpression.Member as PropertyInfo;
            if (property == null)
            {
                throw new ArgumentException("The member access expression does not access a property.", "propertyExpression");
            }

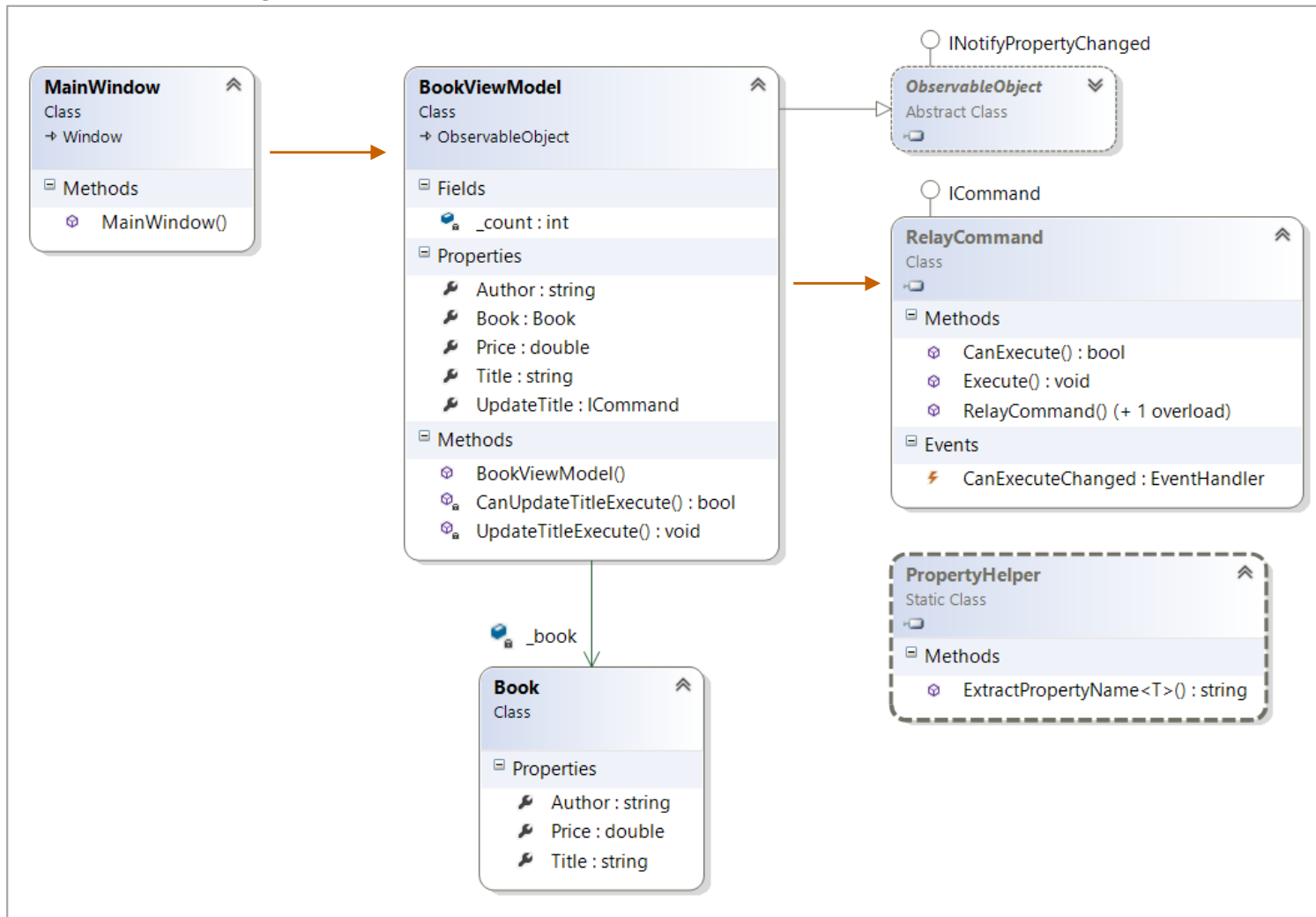
            var getMethod = property.GetGetMethod(true);
            if (getMethod.IsStatic)
            {
                throw new ArgumentException("The referenced property is a static property.", "propertyExpression");
            }

            return memberExpression.Member.Name;
        }
    }
}
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 4

### Klassendiagramm



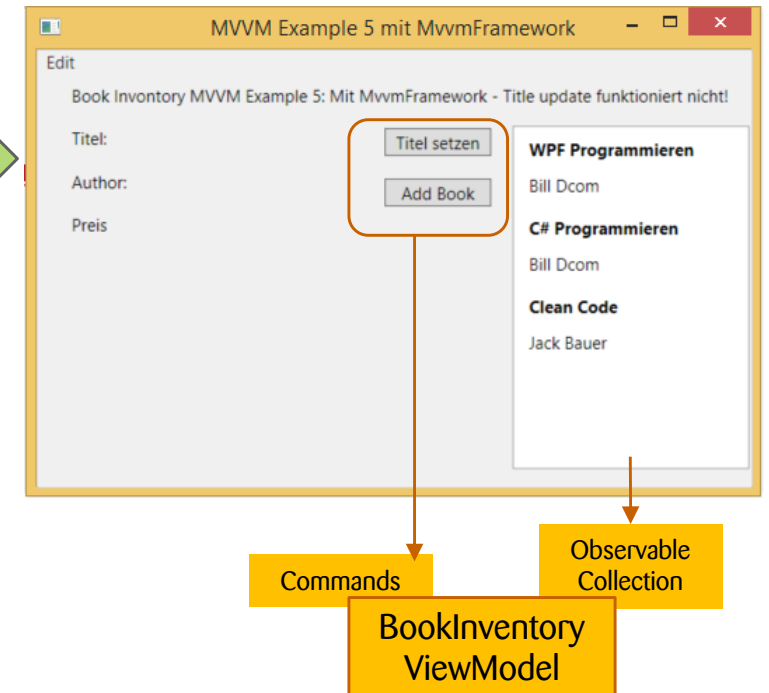
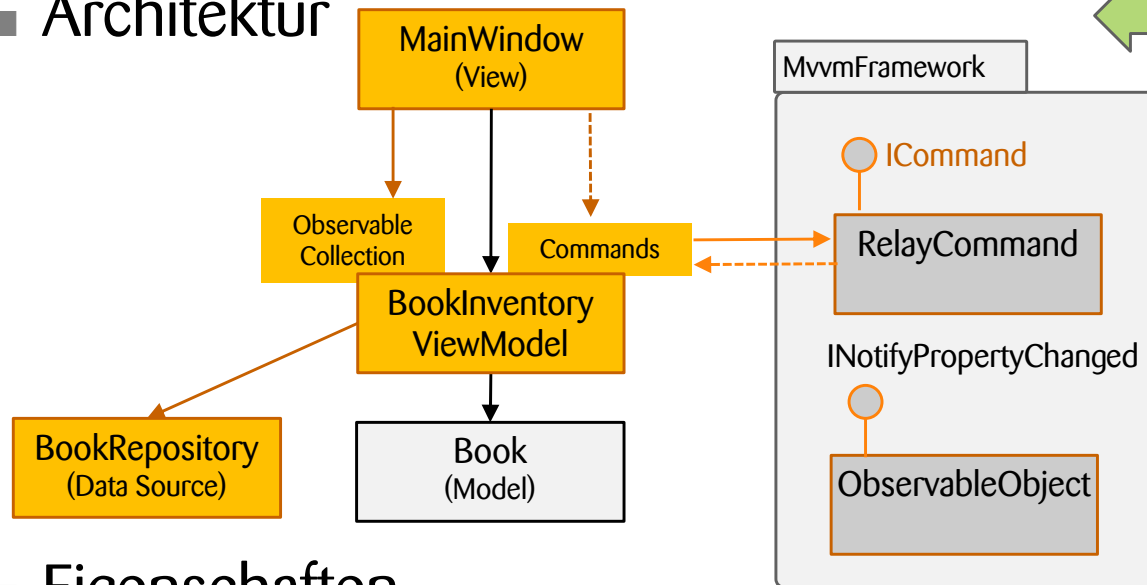
# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### Bücherliste mit ObservableCollection

#### ■ Architektur



#### ■ Eigenschaften

- Funktionalität: Bücherinventar als Liste anzeigen, Buch hinzufügen, Titel ändern
- Titel Änderung wird auf View nicht automatisch aktualisiert
- View: ListView und Button «Add Book» hinzugefügt
- BookRepository als Datenquelle hinzugefügt

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### BookRepository

```
namespace BookInventoryMvvmExample5
{
    public class BookRepository
    {
        #region Members
        readonly Random _random = new Random();
        readonly string[] _titles = { "C# Programmieren", "Clean Code", "Design Patterns", ".NET Framework", "WPF Programmieren" };
        readonly string[] _authors = { "Jack Bauer", "Steven King", "Roy Hackman", "Bill Dcom", "Donald Duck" };
        readonly double[] _prices = { 12.95, 39.90, 24.90, 29.50 };
        #endregion

        #region Properties
        public string GetRandomBookTitles
        {
            get { return _titles[_random.Next(_titles.Length)]; }
        }

        public string GetRandomAuthors
        {
            get { return _authors[_random.Next(_authors.Length)]; }
        }

        public double GetRandomPrices
        {
            get { return _prices[_random.Next(_prices.Length)]; }
        }
        #endregion
    }
}
```

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### BookInventoryViewModel (1/2)

```
namespace BookInventoryMvvmExample5
{
    public class BookInventoryViewModel
    {
        #region Members
        private readonly BookRepository _bookRepository = new BookRepository();
        ObservableCollection<Book> _books = new ObservableCollection<Book>();
        #endregion

        #region Properties
        public ObservableCollection<Book> Books
        {
            get { return _books; }
            set { _books = value; }
        }
        #endregion

        #region Construction
        public BookInventoryViewModel()
        {
            for (int i = 0; i < 3; i++)
            {
                _books.Add(new Book
                {
                    Title = _bookRepository.GetRandomBookTitles,
                    Author = _bookRepository.GetRandomAuthors,
                    Price = _bookRepository.GetRandomPrices
                });
            }
        }
        #endregion
    }
}
```

ObservableCollection<Book>

Bücher aus dem  
Repository lesen



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### BookInventoryViewModel (2/2)

```
#region Commands
void UpdateTitleExecute()
{
    if (_books == null)
        return;

    foreach (var book in _books)
    {
        book.Title = _bookRepository.GetRandomBookTitles;
    }
}

bool CanUpdateTitleExecute()
{
    return true;
}

public ICommand UpdateTitles { get { return new RelayCommand(UpdateTitleExecute, CanUpdateTitleExecute); } }

void AddBookExecute()
{
    if (_books == null)
        return;

    _books.Add(new Book
    {
        Title = _bookRepository.GetRandomBookTitles,
        Author = _bookRepository.GetRandomAuthors,
        Price = _bookRepository.GetRandomPrices
    });
}

bool CanAddBookExecute()
{
    return true;
}

public ICommand AddBook { get { return new RelayCommand(AddBookExecute, CanAddBookExecute); } }
#endregion
}
```

UpdateTitle-Command

AddBook-Command

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### View

ListView zur Darstellung  
der Bücherliste

```
<Label Name="LabelPrice" Content="{Binding ElementName=ListViewBooks, Path=SelectedItem.Price}" Canvas.Left="77" Canvas.Top="102" Width="45"/>

<ListView Name="ListViewBooks" Height="245" Width="169" Canvas.Left="329" Canvas.Top="44" RenderTransformOrigin="0.5,0.5" ItemsSource="{Binding Books}">
    <ListView.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <Label Content="{Binding Title}" FontWeight="Bold" />
                <Label Content="{Binding Author}"/>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
<Button Content="Add Book" Canvas.Left="238" Canvas.Top="82" Width="76" Command="{Binding AddBook}"/>
</Canvas>
</Grid>
</Window>
```

Data-Binding

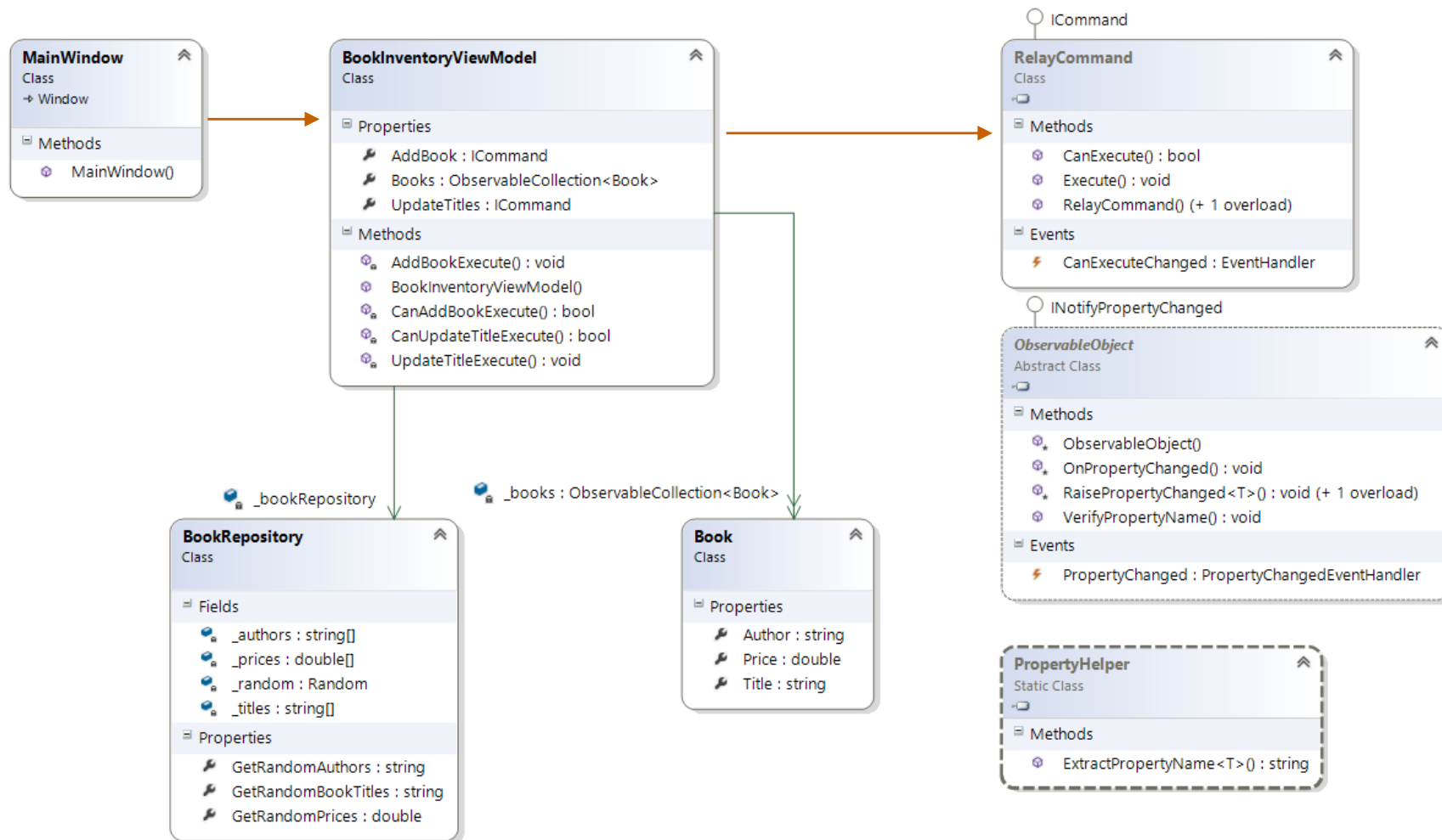
Command-Binding

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 5 Collections

WRONG  
WAY

### Klassendiagramm

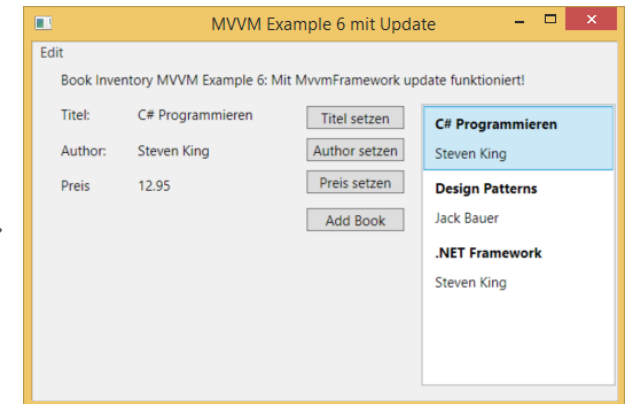
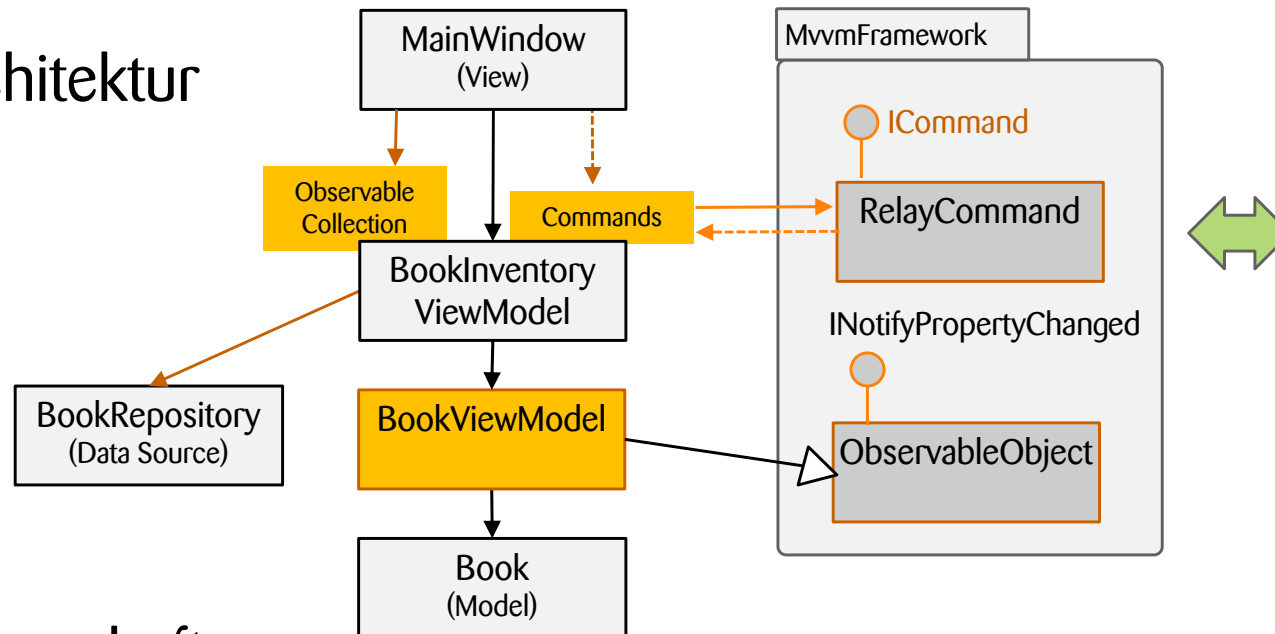


# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### Bücherliste Final

#### ■ Architektur



#### ■ Eigenschaften

- Funktionalität: Bücher Inventar als Liste anzeigen, Buch hinzufügen, Titel, **Author und Preis** ändern
- **Änderungen am Buch werden auf GUI automatisch aktualisiert**
- View: **Buttons «Author setzen» und «Preis setzen»** hinzugefügt
- **BookViewModel in ObservableCollection verwendet statt Book**

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### BookInventoryViewModel (1/3)

```
using System.Collections.ObjectModel;
using System.Windows.Input;
using MvvmCommon;

namespace BookInventoryMvvmExample6
{
    public class BookInventoryViewModel
    {
        #region Members
        private readonly BookRepository _bookRepository = new BookRepository();
        ObservableCollection<BookViewModel> _books = new ObservableCollection<BookViewModel>();
        #endregion

        #region Properties
        public ObservableCollection<BookViewModel> Books
        {
            get { return _books; }
            set { _books = value; }
        }
        #endregion

        #region Construction
        public BookInventoryViewModel()
        {
            for (int i = 0; i < 3; i++)
            {
                _books.Add(new BookViewModel()
                {
                    Title = _bookRepository.GetRandomBookTitles,
                    Author = _bookRepository.GetRandomAuthors,
                    Price = _bookRepository.GetRandomPrices
                });
            }
        }
    }
    #endregion
}
```

ObservableCollection<BookViewModel>

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### BookInventoryViewModel (2/3)

```
#region Commands
void UpdateTitlesExecute()
{
    if (_books == null)
        return;

    foreach (var book in _books)
    {
        book.Title = _bookRepository.GetRandomBookTitles;
    }
}

bool CanUpdateTitlesExecute()
{
    return true;
}

public ICommand UpdateTitles { get { return new RelayCommand(UpdateTitlesExecute, CanUpdateTitlesExecute); } }

void UpdateAuthorsExecute()
{
    if (_books == null)
        return;

    foreach (var book in _books)
    {
        book.Author = _bookRepository.GetRandomAuthors;
    }
}

bool CanUpdateAuthorsExecute()
{
    return true;
}

public ICommand UpdateAuthors { get { return new RelayCommand(UpdateAuthorsExecute, CanUpdateAuthorsExecute); } }
```

UpdateTitles-Command

UpdateAuthors-Command

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### BookInventoryViewModel (3/3)

```
void UpdatePricesExecute()
{
    if (_books == null)
        return;

    foreach (var book in _books)
    {
        book.Price = _bookRepository.GetRandomPrices;
    }
}

bool CanUpdatePricesExecute()
{
    return true;
}

public ICommand UpdatePrices { get { return new RelayCommand(UpdatePricesExecute, CanUpdatePricesExecute); } }

void AddBookExecute()
{
    if (_books == null)
        return;

    _books.Add(new BookViewModel()
    {
        Title = _bookRepository.GetRandomBookTitles,
        Author = _bookRepository.GetRandomAuthors,
        Price = _bookRepository.GetRandomPrices
    });
}

bool CanAddBookExecute()
{
    return true;
}

public ICommand AddBook { get { return new RelayCommand(AddBookExecute, CanAddBookExecute); } }
#endregion
}
```

UpdatePrices-Command

AddBook-Command

# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### View

```
<Grid>
  <Menu Grid.Row="0" Grid.ColumnSpan="3">
    <MenuItem Header="Edit">
      <MenuItem Header="Titel setzen" Command="{Binding UpdateTitles}" />
      <MenuItem Header="Autoren setzen" Command="{Binding UpdateAuthors}" />
      <MenuItem Header="Preise setzen" Command="{Binding UpdatePrices}" />
    </MenuItem>
  </Menu>
  <Canvas HorizontalAlignment="Left" Height="301" Margin="10,10,0,0" VerticalAlignment="Top" Width="498">
    <Label Content="Book Inventory MVVM Example 6: Mit MvvmFramework update funktioniert!" Canvas.Left="10" Canvas.Top="10" Width="478"/>
    <Label Content="Titel:" Canvas.Left="10" Canvas.Top="40" Width="79"/>
    <Label Content="Author:" Canvas.Left="10" Canvas.Top="71"/>
    <Label Content="Preis" Canvas.Left="10" Canvas.Top="102"/>
    <Label Name="LabelTitle" Content="{Binding ElementName=ListViewBooks, Path=SelectedItem.Title}" Canvas.Left="77" Canvas.Top="40" Width="138"/>
    <Label Name="LabelAuthor" Content="{Binding ElementName=ListViewBooks, Path=SelectedItem.Author}" Canvas.Left="77" Canvas.Top="71" Width="138"/>
    <Label Name="LabelPrice" Content="{Binding ElementName=ListViewBooks, Path=SelectedItem.Price}" Canvas.Left="77" Canvas.Top="102" Width="45"/>
    <Button Name="SetTitle" Content="Titel setzen" Canvas.Left="229" Canvas.Top="46" Width="85" Command="{Binding UpdateTitles}" />
    <Button Content="Author setzen" Canvas.Left="229" Canvas.Top="74" Width="85" Command="{Binding UpdateAuthors}" />
    <Button Content="Preis setzen" Canvas.Left="229" Canvas.Top="102" Width="85" Command="{Binding UpdatePrices}" />
    <Button Content="Add Book" Canvas.Left="229" Canvas.Top="135" Width="85" Command="{Binding AddBook}" />

    <ListView Name="ListViewBooks" Height="245" Width="169" Canvas.Left="329" Canvas.Top="44" RenderTransformOrigin="0.5,0.5" ItemsSource="{Binding Books}">
      <ListView.ItemTemplate>
        <DataTemplate>
```

Command-Binding

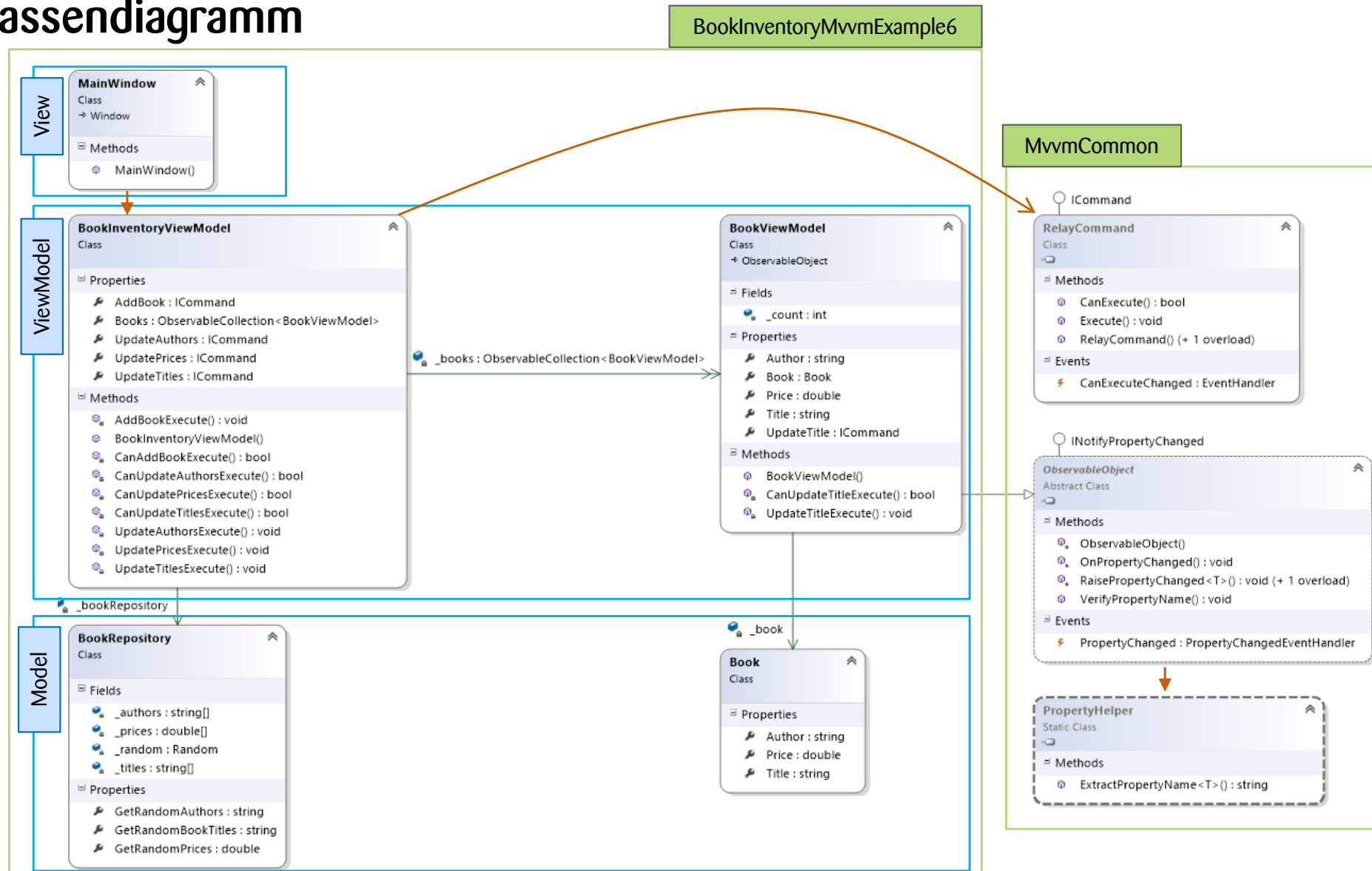
Command-Binding



# MVVM – Schritt für Schritt

## Bücherverwaltung – Beispiel 6 Collections

### Klassendiagramm



# Finden von DataBinding-Problemen

---

- Im Debug-Output werden Data-Binding-Fehler ausgegeben
- Durch den Einsatz eines leeren ValueConverters, in dem man einen Breakpoint setzen kann
- Durch das Binden auf ein einfacheres Objekt

## Snoop

- Snoop ist ein nützliches Tool zum Analysieren von Layoutproblemen in WPF
- Kostenloser Download: <https://github.com/cplotts/snoopwpf>

