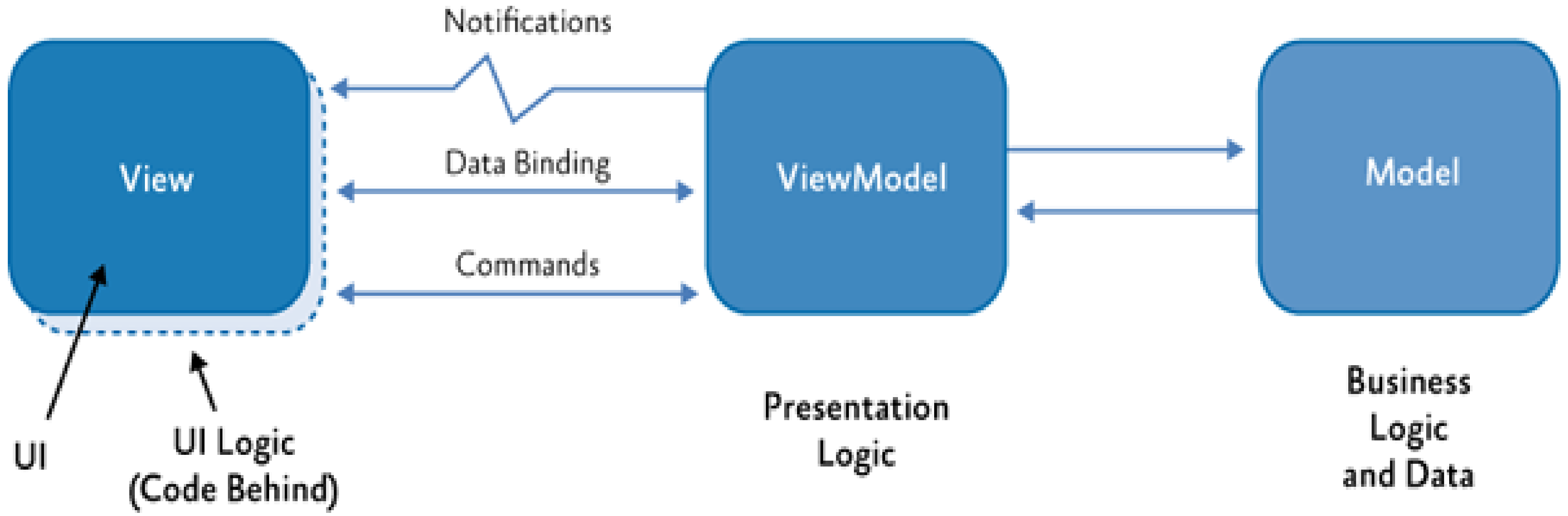
MVVM

MVVM Grundkonzept



View

- **The view is a visual element**, such as a window, page, user control, or data template.
- The view **references the view** model through its **DataContext** property.
- The controls in the view **are data bound to the properties and commands** exposed by the view model.
- **The view's code-behind only define UI logic** to implement visual behavior.

ViewModel

- The view model is a **non-visual class.**
- It **encapsulates the presentation logic.**
- The view model is **testable independently of the view** and the model.
- The view model typically does **not directly reference the view**.
- It **implements properties and commands** to which the view can data bind.
- It notifies the view of any state changes via change notification events via the **INotifyPropertyChanged** and **INotifyCollectionChanged** interfaces.
- The view model **coordinates the view's interaction** with the model.

Model

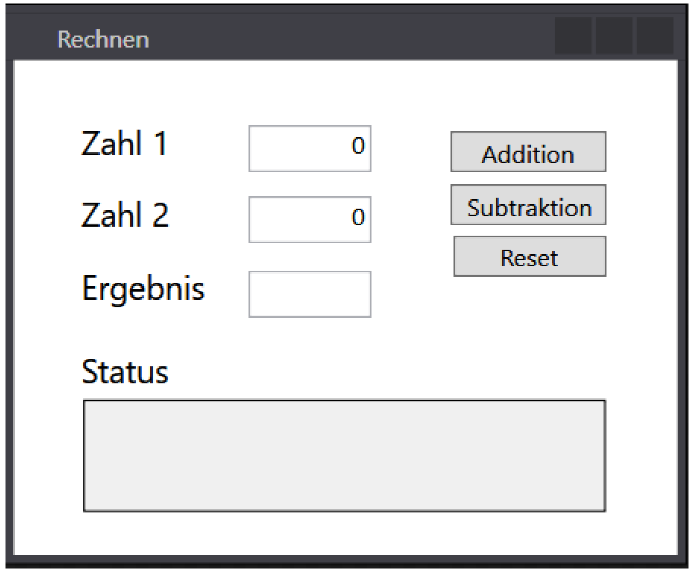
- Model classes are **non-visual classes that encapsulate the application's data and business logic.**
- They are responsible for **managing the application's data** and for ensuring its consistency and validity by encapsulating the required business rules and data validation logic.
- The model classes do **not directly reference the view or view model** classes and have no dependency on how they are implemented.
- The model classes are typically used in conjunction with a service or repository that encapsulates data access and caching. (SQL, XML, …)

Discussion (MSDN Microsoft)

- (!) The model classes can also provide property and collection change notification events through the **INotifyPropertyChanged** and **INotifyCollectionChanged** interfaces.
- (!) Model classes that represent collections of objects typically derive from the **ObservableCollection<T>** class.

MVVM Beispiel

View Rechnen



DataContext setzen in der View

```
<Window.DataContext>
    <ViewModel:RechnenViewModel/>
</Window.DataContext>
```

Bindings ⇔

Commands ⇔

ViewModel Rechnen

```
Referenz auf Model
 Zahlen r = new Zahlen()

Properties für
 -Zahl1
 -Zahl2
 -Ergebnis
 -Fehler (Status)

Commands für
 -Addition
 -Subtraktion
 -Reset
```

⇔

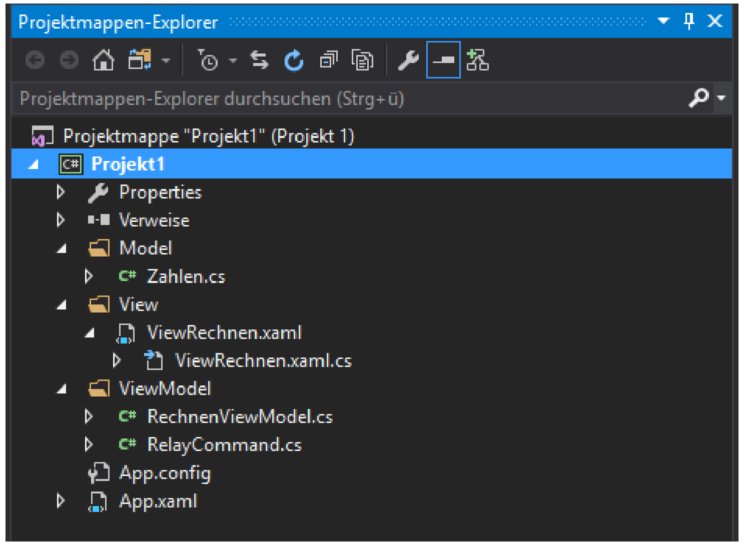
Interface für Properties: INotifyPropertyChanged
Interface für Commands: ICommand

Hilfsklasse RelayCommand verwenden
2 Methoden für jedes Command: Execute und CanExecute

Model Zahlen

```
public class Zahlen
    {
        public decimal? Zahl1 { get; set; }
        public decimal? Zahl2 { get; set; }
        public decimal? ZahlErg { get; set; }
        public void Sub()
        {
            this.ZahlErg = Zahl1 - Zahl2;
        }

        public void Add()
        {
            this.ZahlErg = Zahl1 + Zahl2;
        }
    }
```

MVVM Beispiel Ordnerstruktur



```
Projektmappen-Explorer
Projektmappen-Explorer durchsuchen (Strg+ü)
Projektmappe "Projekt1" (Projekt 1)
  Projekt1
    Properties
    Verweise
    Model
      Zahlen.cs
    View
      ViewRechnen.xaml
        ViewRechnen.xaml.cs
    ViewModel
      RechnenViewModel.cs
      RelayCommand.cs
    App.config
    App.xaml
```

3 Order erstellen:
- View
- ViewModel
- Model

App.xaml umschreiben

```
<Application x:Class="Projekt1.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:Projekt1"
             StartupUri="View\ViewRechnen.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

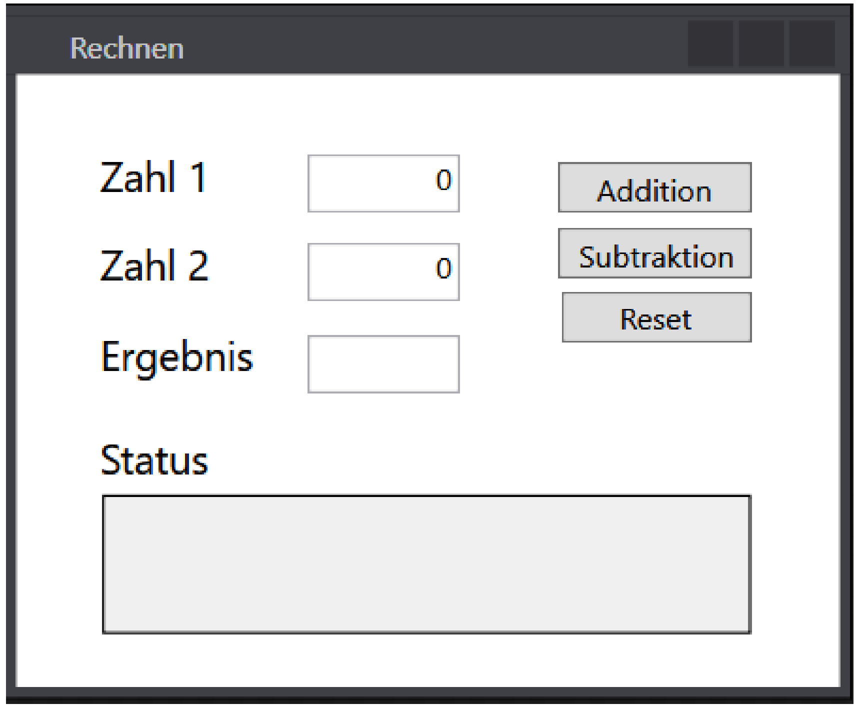
```
namespace Projekt1.View
{
    public partial class ViewRechnen : Window
    {
        public ViewRechnen() {
            InitializeComponent();
        }  //gesamter Code in View
    }
}
```

```
...
using Projekt1.Model;
namespace Projekt1.ViewModel
{
    class RechnenViewModel : INotifyPropertyChanged
    {
        ....         //Properties and Commands
    }
}
```

```
namespace Projekt1.Model
{
    public class Zahlen
    {
        ....  //Model
    }
}
```

MVVM Beispiel Rechnen

View Rechnen



Properties

Zahl1
```
Text="{Binding Zahl1, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

Zahl2
```
Text="{Binding Zahl2, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

Ergebnis
```
Text="{Binding ZahlErgebnis, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Foreground="{Binding FarbeVordergrund}"
```

Commands
```
Command="{Binding SubtraktionButtonCommand}"
Command="{Binding AdditionButtonCommand}"
Command="{Binding ResetButtonCommand}"
```

Status
```
Text="{Binding Status, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
Background="{DynamicResource {x:Static SystemColors.ControlBrushKey}}"
```

MVVM Beispiel

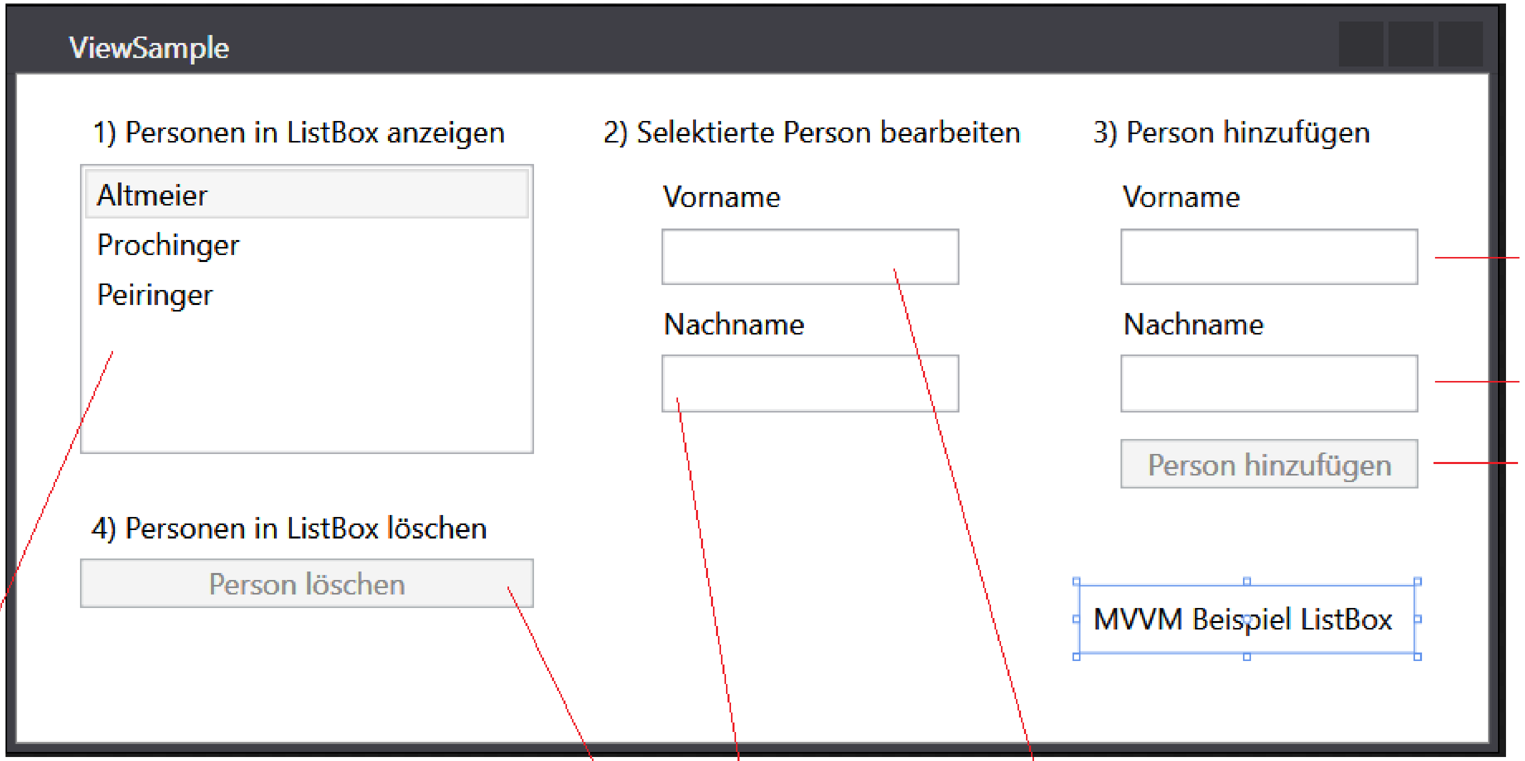
RechnenViewModel

```
using System.ComponentModel;
using Projekt1.Model;
...
namespace Projekt1.ViewModel
{
    class RechnenViewModel : INotifyPropertyChanged
    {
        #region Property ChangedEvent
        public event PropertyChangedEventHandler PropertyChanged;

        protected internal void OnPropertyChanged(string propertyname)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new
                PropertyChangedEventArgs(propertyname));
        }                                           //Code immer gleich für
        #endregion                                  //INotifyPropertyChanged

        #region Property Zahl1
        public string Zahl1
        {
            get { return z.Zahl1.ToString(); }
            set
            {
                decimal parsed;
                if (Decimal.TryParse(value, out parsed))
                {
                    z.Zahl1 = parsed;
                }
                OnPropertyChanged("Zahl1");
            }
        }
        #endregion
```

```
public RechnenViewModel() { //Konstruktor
    _z = new Zahlen(); //Model anlegen
}


public ICommand AdditionButtonCommand {
     get { return new RelayCommand(Addition, CanExecuteAddition); } }
                                     //definiert in ICommand
                                     //RelayCommand immer gleich

     private void Addition()
            {
                try
                {
                    Status = ""; // zurücksetzen
                    z.Add();     //Zahlen addieren
                    OnPropertyChanged("ZahlErgebnis");   //was passiert
                    Status = "Berechnung ok";            //bei diesen
                    Farbe();                             //Zeilen
                }
                catch (Exception ex)
                {
                    Status = "Fehler bei der Addition:" +
                             Environment.NewLine + ex.Message;
                    ZahlErgebnis = "Fehler!";
                }
            }

      private bool CanExecuteAddition()
      {
        return true;
      }
        ...
    }
}
```

MVVM ListBox Sample - View

```
View Person
```



```
Text="{Binding NewPerson.Vorname}"
Text="{Binding NewPerson.Nachname}"
Command="{Binding AddPersonCommand, Mode=OneWay}"
```

```
SelectedItem="{Binding SelectedPerson}"
SelectedIndex="{Binding SelectedIndexPerson}"
ItemsSource="{Binding PersonCollection}"
DisplayMemberPath="Nachname"
```

```
Text="{Binding SelectedPerson.Vorname}"
Text="{Binding SelectedPerson.Nachname}"
Command="{Binding DelPersonCommand, Mode=OneWay}"
```

```
DataContext für das ViewModel
<Window.DataContext>
    <ViewModel:PersonViewModel/>
</Window.DataContext>
```

MVVM ListBox Sample – ViewModel

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using MVVM_Sample.Model;
using System.ComponentModel;          //für INotifyPropertyChanged
using System.Collections.ObjectModel; //für ObservableCollection
using System.Windows.Input;           //für ICommand

namespace MVVM_Sample.ViewModel
{
    class PersonViewModel : INotifyPropertyChanged
    {

        #region Event PropertyChanged
        public event PropertyChangedEventHandler PropertyChanged;
        protected internal void OnPropertyChanged(string propertyname)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new
                PropertyChangedEventArgs(propertyname));
        }
        #endregion
```

dieser Code ist immer gleich

MVVM ListBox Sample – ViewModel

```
#region Konstruktor
public PersonViewModel()
{
    personCollection = new ObservableCollection<Person>();
    personCollection.Add(new Person { Vorname = "Jürgen", Nachname = "Altmeier" });
    personCollection.Add(new Person { Vorname = "Andreas", Nachname = "Prochinger" });
    personCollection.Add(new Person { Vorname = "Jürgen", Nachname = "Peiringer" });
}
#endregion
```

ObservableCollection implementiert automatisch das Interface INotifyPropertyChanged

einfach Initialisierung der Collection

```
#region PersonCollection
private ObservableCollection<Person> personCollection;
public ObservableCollection<Person> PersonCollection
    {
    get { return personCollection; }
    set
    {
        if (value != personCollection)
            personCollection = value;
        this.OnPropertyChanged("PersonCollection");
    }
}
#endregion
```

Property

View benachrichtigen

MVVM ListBox Sample – ViewModel

```
#region Properties SelectedPerson
private Person selectedPerson;
public Person SelectedPerson
{
    get { return selectedPerson; }
    set
    {
        selectedPerson = value;
        this.OnPropertyChanged("SelectedPerson");
    }
}
#endregion
```

private Speichervariable und
öffentliches Property

```
#region SelectedIndexPerson
private int selectedIndexPerson;
public int SelectedIndexPerson
{
    get { return selectedIndexPerson; }
    set
    {
            selectedIndexPerson = value;
        this.OnPropertyChanged("SelectedIndexPerson");
    }
}
#endregion
```

private Speichervariable und
öffentliches Property

MVVM ListBox Sample – ViewModel

```
#region Properties NewPerson
private Person newPerson = new Person();
public Person NewPerson
{
    get { return newPerson; }
    set
    {
        newPerson = value;
        OnPropertyChanged("NewPerson");
    }
}
#endregion
```

MVVM ListBox Sample – ViewModel

```
#region Command AddPersonCommand
public ICommand AddPersonCommand { get { return new RelayCommand(Add, CanExecuteAdd); } }

private void Add()
{
    personCollection.Add(new Person(newPerson.Vorname, newPerson.Nachname));
    NewPerson.Vorname = null;
    NewPerson.Nachname = null;
    OnPropertyChanged("NewPerson");  //Benachrichtigung Person hat sich geändert
}

private bool CanExecuteAdd()
{
    return NewPerson.Vorname != null && NewPerson.Nachname != null;
}
#endregion
```

ob Button
klickbar

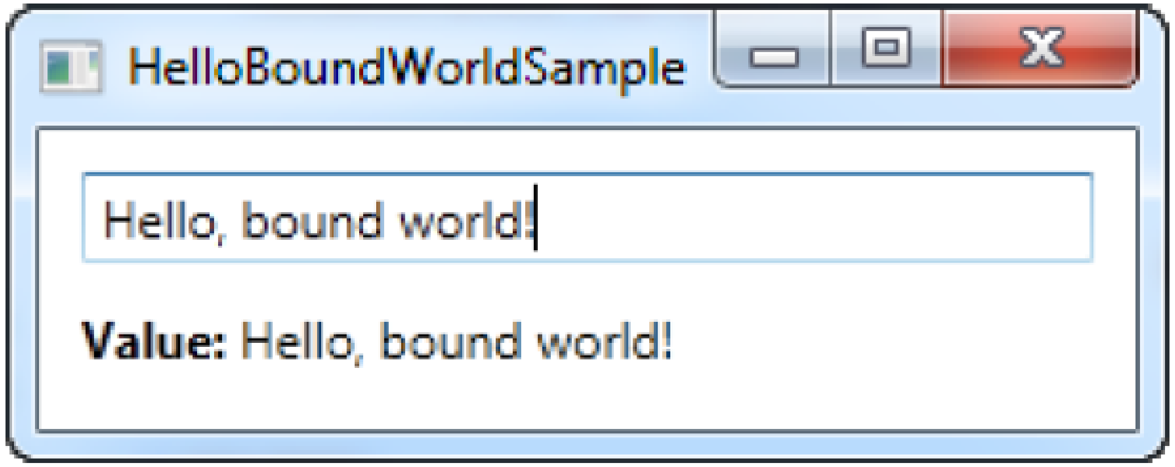
MVVM ListBox Sample – ViewModel

```
        #region Command DelPersonCommand
        private RelayCommand delPersonCommand;
        public ICommand DelPersonCommand
        {
           get {return delPersonCommand ?? (delPersonCommand = new RelayCommand(DeletePerson, CanExecuteDel)); }
        }
        private bool CanExecuteDel()
        {
            return SelectedPerson != null;
        }

        private void DeletePerson()
        {
            PersonCollection.Remove(SelectedPerson);
        }
        #endregion
    }
}
```

MVVM ListBox Sample – Model

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVVM_Sample.Model
{
    class Person
    {
        public String Vorname { set; get; }
        public String Nachname { set; get; }

        public Person() { }
        public Person(String vorname, String nachname)
        {
            Vorname = vorname;
            Nachname = nachname;
        }
    }
}
```

Binding 1

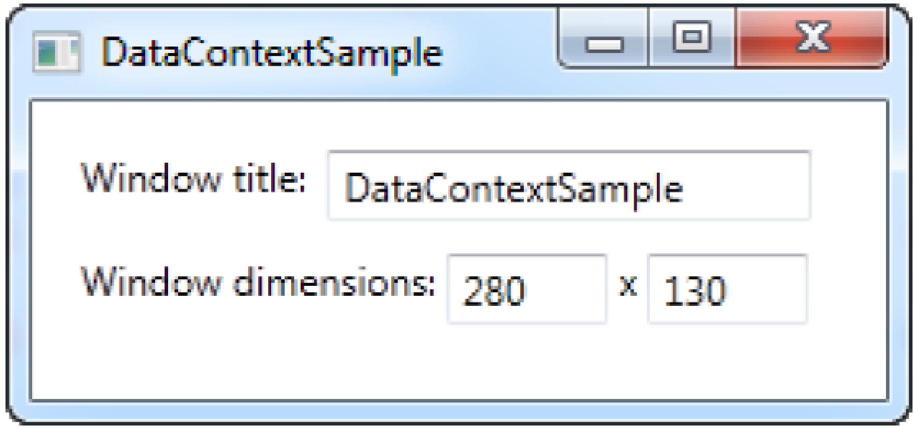


Hello, bound world!

Value: Hello, bound world!

```
<Window x:Class="WpfTutorialSamples.DataBinding.HelloBoundWorldSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="HelloBoundWorldSample" Height="110" Width="280">
    <StackPanel Margin="10">
                <TextBox Name="txtValue" />
                <WrapPanel Margin="0,10">
                        <TextBlock Text="Value: " FontWeight="Bold" />
                        <TextBlock Text="{Binding Path=Text, ElementName=txtValue}" />
                </WrapPanel>
        </StackPanel>
</Window>
```

Binding 2



Window title: DataContextSample

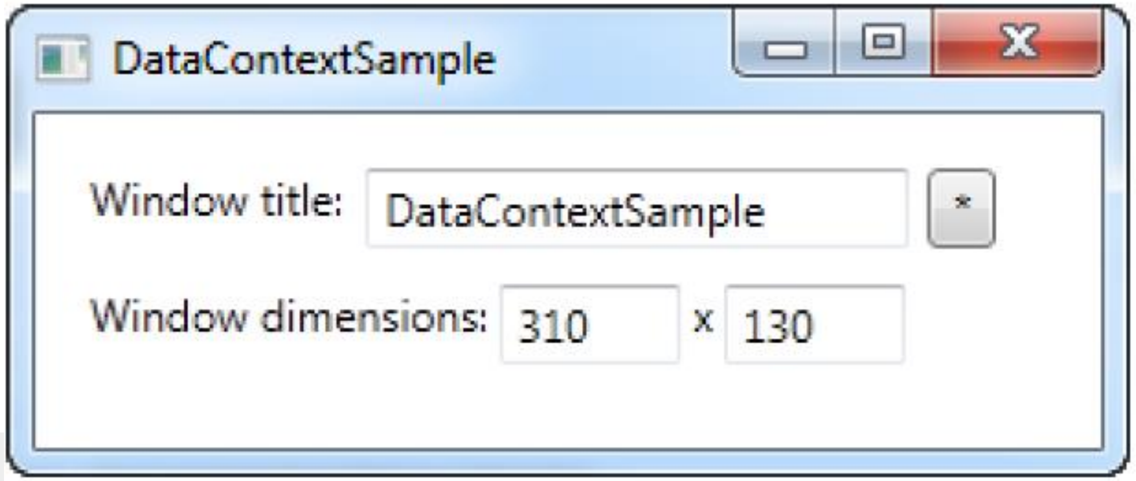
Window dimensions: 280 x 130

```
<StackPanel Margin="15">
        <WrapPanel>
                <TextBlock Text="Window title:  " />
                <TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}"
                               Width="150" />
        </WrapPanel>
        <WrapPanel Margin="0,10,0,0">
                <TextBlock Text="Window dimensions: " />
                <TextBox Text="{Binding Width}" Width="50" />
                <TextBlock Text=" x " />
                <TextBox Text="{Binding Height}" Width="50" />
        </WrapPanel>
</StackPanel>
```

```
public partial class DataContextSample : Window
{
        public DataContextSample()
        {
                InitializeComponent();
                this.DataContext = this;
        }
}
```

- XAML und CodeBehind

Binding 3

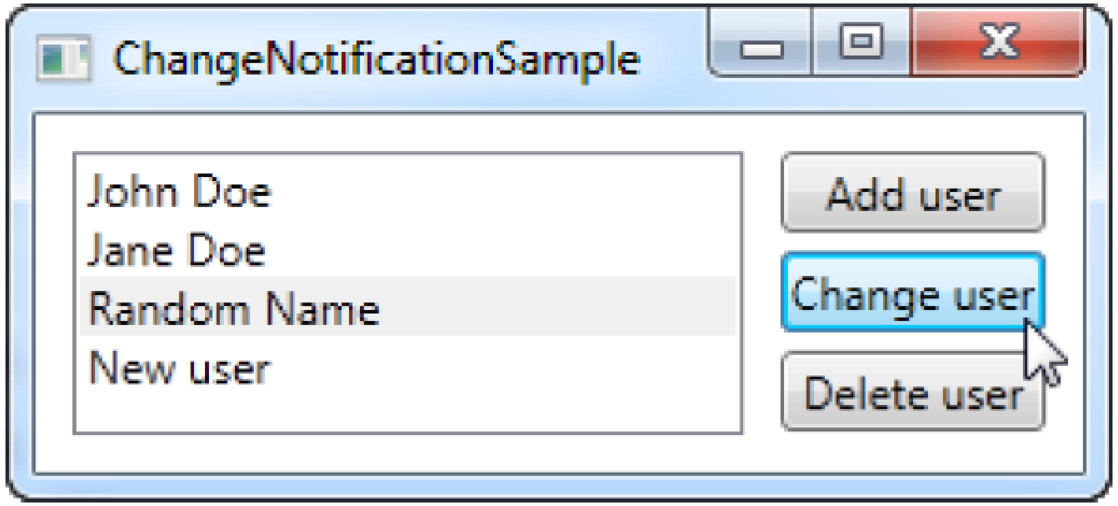


Window title: DataContextSample *

Window dimensions: 310 x 130

```
<WrapPanel>
        <TextBlock Text="Window title:  " />
        <TextBox Name="txtWindowTitle" Text="{Binding Title,
                      UpdateSourceTrigger=Explicit}" Width="150" />
        <Button Name="btnUpdateSource" Click="btnUpdateSource_Click"
                Margin="5,0" Padding="5,0">*</Button>
</WrapPanel>
<WrapPanel Margin="0,10,0,0">
        <TextBlock Text="Window dimensions: " />
        <TextBox Text="{Binding Width, UpdateSourceTrigger=LostFocus}"
                 Width="50" />
        <TextBlock Text=" x " />
        <TextBox Text="{Binding Height,
                 UpdateSourceTrigger=PropertyChanged}" Width="50" />
</WrapPanel>
```

Binding 4 – Teil 1

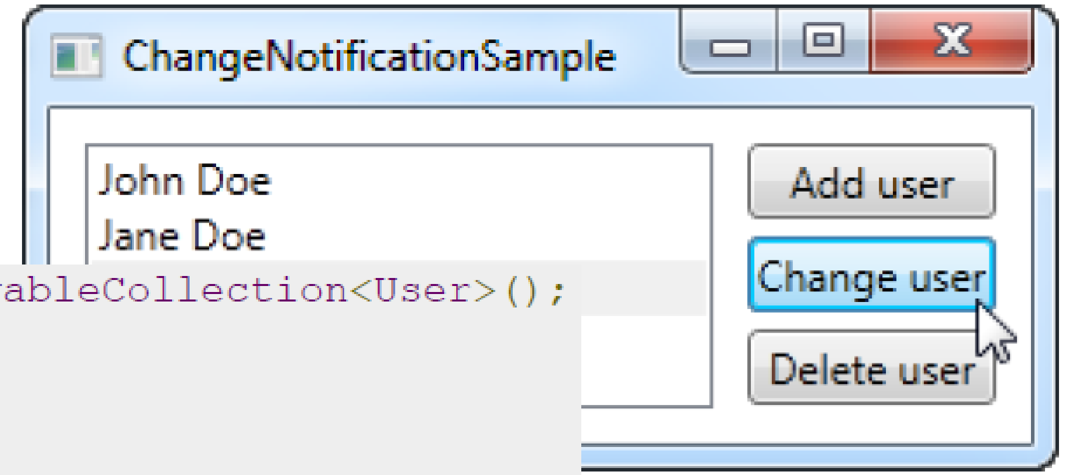

ChangeNotificationSample

John Doe
Jane Doe
Random Name
New user

Add user
Change user
Delete user

```
<Window x:Class="WpfTutorialSamples.DataBinding.ChangeNotificationSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ChangeNotificationSample" Height="135" Width="300">
        <DockPanel Margin="10">
                <StackPanel DockPanel.Dock="Right" Margin="10,0,0,0">
                        <Button Name="btnAddUser" Click="btnAddUser Click">Add user</Button>
                        <Button Name="btnChangeUser" Click="btnChangeUser Click"
Margin="0,5">Change user</Button>
                        <Button Name="btnDeleteUser" Click="btnDeleteUser Click">Delete
user</Button>
                </StackPanel>
                <ListBox Name="lbUsers" DisplayMemberPath="Name"></ListBox>
        </DockPanel>
</Window>
```

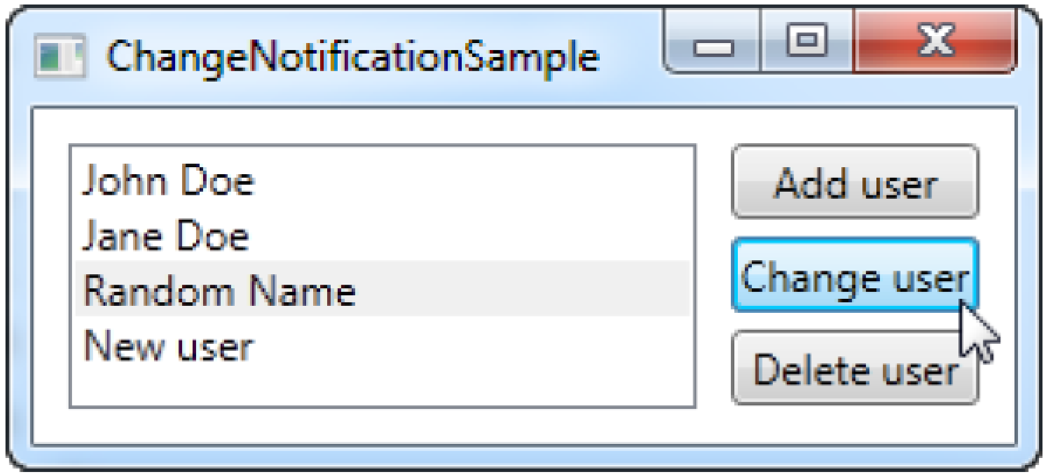
Binding 4 – Teil 2



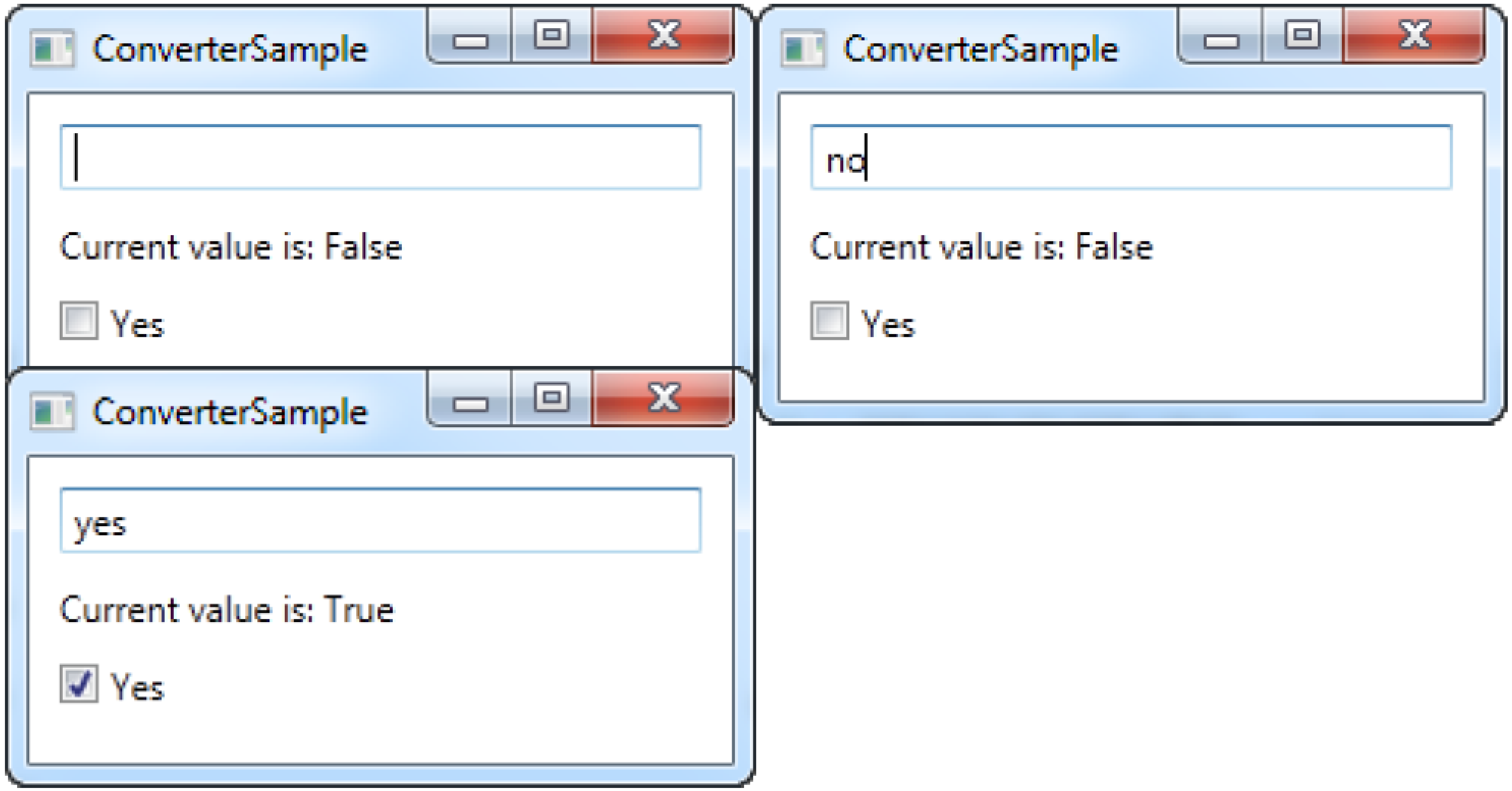
```
private ObservableCollection<User> users = new ObservableCollection<User>();

public ChangeNotificationSample()
{
        InitializeComponent();

        users.Add(new User() { Name = "John Doe" });
        users.Add(new User() { Name = "Jane Doe" });

        lbUsers.ItemsSource = users;
}

private void btnAddUser_Click(object sender, RoutedEventArgs e)
{
        users.Add(new User() { Name = "New user" });
}

private void btnChangeUser_Click(object sender, RoutedEventArgs e)
{
        if(lbUsers.SelectedItem != null)
                (lbUsers.SelectedItem as User).Name = "Random Name";
}
...
```

Binding 4 – Teil 3

```
public class User : INotifyPropertyChanged
{
        private string name;
        public string Name {
                get { return this.name; }
                set
                {
                        if(this.name != value)
                        {
                                this.name = value;
                                this.NotifyPropertyChanged("Name");
                        }
                }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        public void NotifyPropertyChanged(string propName)
        {
                if(this.PropertyChanged != null)
                        this.PropertyChanged(this, new
                                 PropertyChangedEventArgs(propName));
        }
}
```



IValueConverter – Teil 1


ConverterSample

Current value is: False

☐ Yes

ConverterSample

no

Current value is: False

☐ Yes

ConverterSample

yes

Current value is: True

☑ Yes

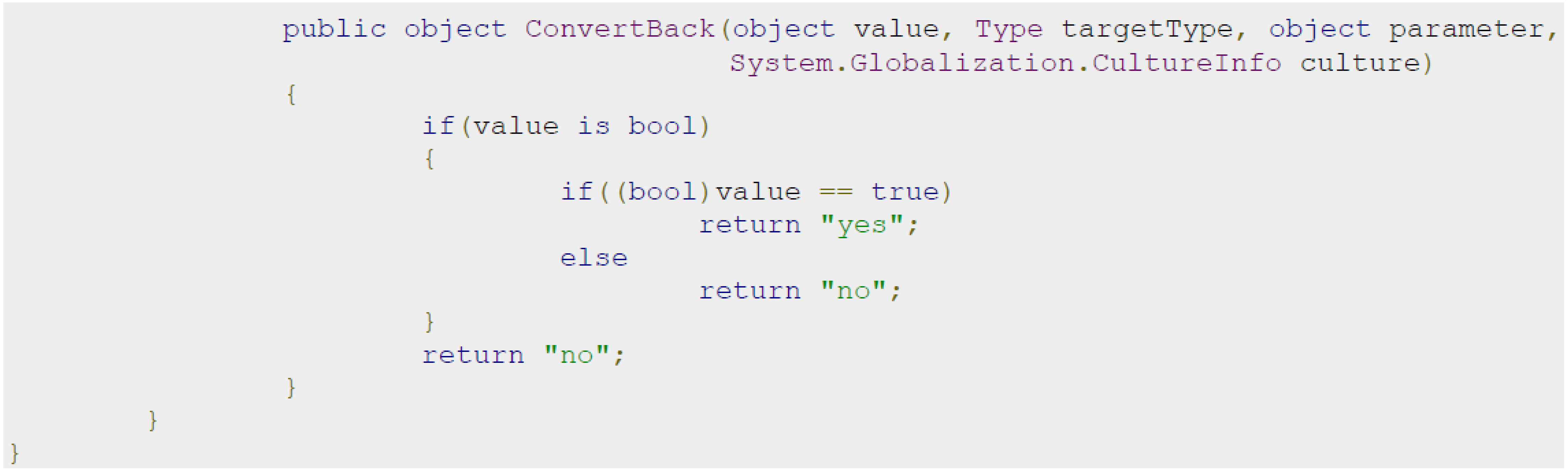
IValueConverter – Teil 2

```
<Window.Resources>
        <local:YesNoToBooleanConverter x:Key="YesNoToBooleanConverter" />
</Window.Resources>
<StackPanel Margin="10">
        <TextBox Name="txtValue" />
        <WrapPanel Margin="0,10">
                <TextBlock Text="Current value is: " />
                <TextBlock Text="{Binding ElementName=txtValue, Path=Text,
                  Converter={StaticResource YesNoToBooleanConverter}}"></TextBlock>
        </WrapPanel>
        <CheckBox IsChecked="{Binding ElementName=txtValue, Path=Text,
              Converter={StaticResource YesNoToBooleanConverter}}" Content="Yes" />
</StackPanel>
```

IValueConverter – Teil 3

```csharp
public class YesNoToBooleanConverter : IValueConverter
{
        public object Convert(object value, Type targetType, object parameter,
                              System.Globalization.CultureInfo culture)
        {
                switch(value.ToString().ToLower())
                {
                        case "yes":
                        case "oui":
                                return true;
                        case "no":
                        case "non":
                                return false;
                }
                return false;
        }
```

IValueConverter – Teil 4



```
                public object ConvertBack(object value, Type targetType, object parameter,
                                          System.Globalization.CultureInfo culture)
                {
                        if(value is bool)
                        {
                                if((bool)value == true)
                                        return "yes";
                                else
                                        return "no";
                        }
                        return "no";
                }
        }
}
```