Handout
Thema: Datenbindung & MVVM

Architektur

1. Das MVVM-Entwurfsmuster (Pattern)

Wir trennen die Anwendung in drei Schichten, um den Code übersichtlich und testbar zu halten.

- **Model (Daten):** Enthält die reinen Daten (z. B. Klasse Mitarbeiter). Keine Logik für die Oberfläche!
- **View (XAML):** Die Benutzeroberfläche. Sie kümmert sich nur um die Darstellung, nicht um die Verarbeitung.
- **ViewModel (Vermittler):** Der "Klebstoff". Es holt Daten vom Model, bereitet sie auf und hält den Status für die View bereit.

2. Data Binding (Die Magie dazwischen)

"Jetzt die spannende Frage: Wie kommen die Daten vom ViewModel in die View, ohne dass wir alles händisch kopieren müssen?"

- **Die Antwort: Data Binding.**
    - "Statt txtBox.Text = "Wert" zu schreiben, sagen wir im XAML einfach: Text={Binding Name}. Die View 'abonniert' die Eigenschaft einfach."
- **Die Technik: Das Interface INotifyPropertyChanged.**
    - "Damit die View merkt, wenn sich im C#-Code etwas ändert, nutzen wir ein Interface. Man kann es sich wie ein **Alarm-System** vorstellen."
- **Der Ablauf:**
    - "Im ViewModel ändert sich ein Wert (z. B. das Ergebnis einer Rechnung)."
    - "Das ViewModel löst ein Event aus: *'Achtung, der Wert X hat sich geändert!'*"
    - "Die View hört diesen Alarm und aktualisiert das Textfeld sofort automatisch."

Der "Werkzeugkasten" (Helper Code)

Um Zeit zu sparen, kopiert bitte diese zwei Klassen in euer Projekt. Sie sind für die Aufgabe notwendig.

A. Die Basis-Klasse (ViewModelBase.cs)

Die Basisklasse muss das INotifyPropertyChanged interface implementieren.

```
// --- Implementierung von INotifyPropertyChanged ---

public event PropertyChangedEventHandler PropertyChanged;

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



// --- Eigenschaften (Properties) ---

private double _widerstand = 0.0;
public double Widerstand
{
    get => _widerstand;
    private set
    {
        if (_widerstand != value)
        {
            _widerstand = value;
            // Benachrichtigt die UI automatisch über die Änderung
            OnPropertyChanged();
        }
    }
}

private double _spannung = 0.0;
public double Spannung
{
    get => _spannung;
    set
    {
        if (_spannung != value)
        {
            _spannung = value;
            // [CallerMemberName] erkennt automatisch den Namen "Spannung"
            OnPropertyChanged();
        }
    }
}

private double _strom = 0.0;
public double Strom
{
    get => _strom;
    set
    {
        if (_strom != value)
        {
            _strom = value;
            // [CallerMemberName] erkennt automatisch den Namen "Strom"
            OnPropertyChanged();
```

```
        }
    }
}


// --- Logik ---

public void BtnBerechnen_Click(object sender, RoutedEventArgs e)
{
    // Sicherheitsprüfung: Division durch Null verhindern
    // Wir nutzen eine kleine Toleranz (Epsilon), da Gleitkommazahlen ungenau sein können
    if (Math.Abs(Strom) < 0.0001)
    {
        MessageBox.Show("Strom darf nicht 0 sein.", "Fehler", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    // Formel: R = U / I (Ohmsches Gesetz)
    Widerstand = Spannung / Strom;
}
```
