* C#
  + Allgemein
    - Objektorientierte Programmiersprache
    - Ursprünglich für Windows Systeme, heute aber auch plattformunabhängig (macOS, iOS, Android, Linux)
  + Unterschiede zu Java
    - Keine GETTER und SETTER Funktionen. Diese wurden in C# durch Properties ersetzt (Bild Vergleich)
    - Event Handler werden hier als Funktionen genutzt. Nicht wie in Java als Klassen, die ein Interface implementieren. (Bild Beispiel)
* WPF
  + Allgemein
    - WPF ist ein modernes Programmiermodell für die Entwicklung von Windows- und Webbrowser-Anwendungen.
    - WPF verwendet XAML als UI Gestaltungssprache und bietet sehr viele UI Elemente an (Fenster, 2D Grafiken, 3D Grafiken, Dokumente, Animationen, Videos)
    - Mit WPF sind auch viele Visualisierungsmöglichkeiten gegeben (Farbübergänge, Verformungen, Bewegungen, Schatten- und Spiegeleffekte, Bildveränderungen und Überblendeffekte).
    - Außerdem kann man mit WPF Elemente beliebig kombinieren. Man kann z.B. einen Button erstellen, welcher eine Liste enthält, auf welcher verschiedene Videos laufen.
    - WPF unterstützt außerdem wiederverwendbare Styles, welche auf verschiedene UI Elemente angewandt werden können.
  + XAML
    - Extensible Application Markup Language ist eine Beschreibungssprache zur Gestaltung von GUI und ist XML basiert.
    - Mit XAML kann man aber nicht nur eine GUI gestalten, sondern auch grafische Elemente, Verhaltensweisen, Transformationen, Animationen, Farbverläufe und vieles mehr definieren.
    - **ExampleWindow.xaml**
  + Ressourcen
    - Ressourcen können alles mögliche sein, wie z.B. Converter, Strings, Integer, Farben und Styles.
    - Ressourcen können in einer oder mehreren Dateien zusammengefasst werden
    - Ressourcen können über das Schlüsselwort StaticResource oder DynamicResource eingebunden werden. Unterschied.
    - **ExampleResources.xaml**
  + Binding
    - In WPF kann man Eigenschaften von Objekten an Eigenschaften von anderen Objekten binden. Beispiel: Der Titel eines Windows soll dem Wert einer TextBox entsprechen
    - Ein Binding kann über den Namen eines Objektes, den DataContext eines Objektes und ein bestimmtes nächst höheres Element referenziert werden.
    - Es können nur Eigenschaften vom Typ DependencyProperty gebunden werden
    - **ExampleBinding.xaml**
  + Converter
    - Manchmal sind Eigenschaften die man binden will nicht kompatibel. Z.B. wenn man einen String an einen Integer Wert binden will
    - Dazu gibt es Converter, welcher innerhalb eines Bindings einen Wert konvertieren können
    - Ein Converter ist eine Klasse, welche das Interface IValueConverter bzw. IMultiValueConverter implementiert.
    - **ExampleConverter.xaml**
  + Styles
    - Ein Style bestimmt das Aussehen sowie Teile des Verhaltens eines UI Controls. Man kann Standardwerte für Eigenschaften festlegen sowie auf Trigger reagieren.
    - Man kann selbst ein eigenes Template für ein Control erstellen
    - Ein Style kann auch von einem anderen Style erben, was komplexe Styles ermöglicht
    - **ExampleStyle.xaml**
  + DataTemplates
    - Da man Objekte (z.B. Models) an die UI binden kann, müssen diese auch gut dargestellt werden. Standardmäßig wird in WPF die ToString() Methode aufgerufen um ein Objekt visuell darzustellen.
    - Mit DataTemplates kann man Objekten jedoch ein umfangreiches Aussehen vergeben.
    - **ExampleDataTemplates.xaml**
* MVVM
  + Komponenten **ExampleMVVM.xaml**
    - Model
      * Enthält Informationen zu Business Objekten. Models können auch eine Property Validierung enthalten.
    - View
      * Repräsentiert die Benutzeroberfläche mit all ihrem UI-spezifischen Code. Die View zeigt die Daten an, welche sie von dem ViewModel erhält. Dies geschieht über Binding. Die View gibt außerdem über Binding Veränderungen der Daten an das ViewModel.
    - ViewModel
      * Das ViewModel stellt Kommandos und Daten bereit, welche von der View genutzt werden können. Außerdem enthält sie Daten, welche z.B. angeben, ob ein Kommando ausgeführt werden kann oder nicht. Die View ist also eine visuelle Darstellung des ViewModels. Das ViewModel hat allerdings keine Kentniss von der View.
  + INotifyPropertyChanged
    - INotifyPropertyChanged ist ein Interface, welches dafür sorgt, dass ein Binding informiert wird, dass sich eine Eigenschaften verändert hat. Somit kann das Binding sich automatisch auf den neuen Wert aktualisieren
  + DevExpress
    - BindableBase
      * Die BindableBase Klasse implementiert das INotifyPropertyChanged Interface und bietet eine einfache Implementierung von bindable Properties mithilfe der GetProperty() und SetProperty() Funktion.
      * **ExampleBindableBase.cs**
    - ViewModelBase
      * ViewModelBase erbt von BindableBase und hat deswegen auch die Funktionen der BasisKlasse. Zusätzlich bietet sie Funktionen wie eine getrennte Implementierung von Design und Runtime.
      * Sehr einfach Command Implementationen
      * Außerdem kann man Informationen zwischen verschiedenen ViewModels austauschen
      * **ExampleViewModelBase.cs**
    - Commands
      * Für Kommandos wird die DelegateCommand Klasse bereitgestellt, welche sehr einfach zu benutzen ist.
      * **ExampleCommands.cs**