Entwicklung interaktiver Systeme

M. Sc. Tobias Puderer – WS 22/23

Übung 3: Grundlegender Aufbau eines GUI mit XAML

Hinweis: Es ist möglich, viele der geforderten Formatänderungen auch über die Benutzeroberfläche von Visual Studio zu erreichen. Ziel der Übung ist allerdings das Kennenlernen von XAML und C#. Sehen Sie daher bitte von der „einfachen Lösung“ ab ☺

1. Beim Erstellen eines GUI in WPF bedient man sich der Markupsprache XAML. Da diese an XML angelehnt ist, erfolgt der Aufbau des GUI hierarchisch. Als Basis dient genau ein Wurzelobjekt (z.B. `Window` oder `UserControl`), unter welchem alle weiteren Bildschirmobjekte hängen.

```
<Window x:Class="WpfApplication1.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>
       … weitere Inhalte
    </Grid>
</Window>
```

   Alle Bildschirmobjekte werden durch XAML Tags repräsentiert, ihre Eigenschaften entweder durch Attribute an den Tags oder weitere verschachtelte Tags.

```
<OberflaechenObjekt Attribut1="Ein Wert">
  <OberflaechenObjekt.Attribut2>
       Noch ein Wert
  </OberflaechenObjekt.Attribut2>
</OberflaechenObjekt >
```

   a) Definieren Sie einen `Button` mit mehreren Eigenschaften ihrer Wahl in XAML, indem sie teilweise Eigenschaften per Attribut und teilweise als verschachtelte Tags einfügen. Nutzen Sie als Container das `Grid`-Element, welches in der Beispieldatei „MainWindow.xaml“ automatisch eingefügt wurde. Mit welcher Eigenschaft kann der `Button` beschriftet werden?

   b) Erstellen Sie einen zweiten `Button` im `Grid`. Was fällt ihnen auf? Was müssen Sie ändern, um die Objekte z.B. nebeneinander zu platzieren, **ohne** dabei einen weiteren Container zu nutzen oder das Objekt im Editor mit der Maus zu verschieben (Auch wenn es verlockend ist)? Recherchieren Sie hierzu die Nutzung von `Grid`. Was versteht man in dem Zusammenhang unter einer „angehängten Eigenschaft“ (attached property)?

c) Offensichtlich handelt es sich bei einem `Grid` um eine Tabelle. Pro Zeile und Spalte eines `Grid`s können unterschiedliche Breiten bzw. Höhen vergeben werden. Neben der statischen Angabe von fixen Zahlwerten (in px) können auch relative Werte verwendet werden. Das erlaubt eine dynamischere Definition der Größenverhältnisse im Anwendungsfenster.

Bezeichnung	Beschreibung	Beispiel
Zahlenwert in Pixel	Definiert eine feste Höhe bzw. Breite für eine Reihe bzw. Spalte Im Beispiel sieht man eine Reihe mit einer festen Höhe von 100 Pixel.	`<RowDefinition Height="100"/>`

*****	Definiert die Höhe bzw. Breite der Zeile bzw. Spalte anteilig zu der zur Verfügung stehenden Gesamtgröße des `Grids`.
	
	Im ersten Beispiel gibt es zwei Spalten, deren jeweilige Breite sich zu gleichen Teilen auf die Gesamtbreite des Grids verteilt.
	
	Im zweiten Beispiel sind zwei Reihen mit einem Höhenverhältnis von 3:1 zu sehen, d.h. Reihe 1 erhält drei Anteile der Gesamthöhe und Reihe 2 nur einen.
		`<!–Beispiel 1 -->`
		`<ColumnDefinition Width="*"/>`
		`<ColumnDefinition Width="*"/>`
		
		`<!–Beispiel 2 -->`
		`<RowDefinition Height="3*">`
		`<RowDefinition Height="*">`

Auto	Definiert, dass die Höhe bzw. Breite abhängig vom Inhalt der Zeile bzw. Spalte sein soll. Bei mehreren, unterschiedlich hohen Elementen in einer Zeile, kommt für die Gesamthöhe der Zeile der größte `Height`-Wert zum Einsatz. Bei mehreren, unterschiedlich breiten Elementen innerhalb einer Spalte entspricht die Gesamtbreite der Spalte dem größten `Width`-Wert.	`<RowDefinition Width="Auto"/>` `<ColumnDefinition Height="Auto"/>`

Aufgabenstellung: Definieren Sie nun, dass die Spalte, die den zweiten Button enthält, stets doppelt so breit ist wie die erste Spalte. Vergrößern und verkleinern Sie das Applikationsfenster, um den Effekt zu testen.

2. Neben dem **Grid**, welches einer Tabelle entspricht, gibt es auch andere Wege, Elemente auf dem Bildschirm auszurichten. Zwei weitere Möglichkeiten sind **StackPanel**, welches Elemente standardmäßig untereinander anordnet und **WrapPanel**, das die enthaltenen Objekte entsprechend der zur Verfügung stehenden Breite des Panels ggf. wie Fließtext umbricht.

   a) Erstellen Sie ein Beispiel, in welchem Sie in einem **WrapPanel**-Layout fünf verschiedene **TextBox** –Elemente anordnen. Verändern Sie sie Breite des Panels und beobachten Sie das Ergebnis.

   b) Neben der rein grafischen Darstellung existiert für jede XAML-Datei eine Code-Behind-Schicht, welche in C# entwickelt wird. Hier werden die vom Benutzer ausgelösten Ereignisse abgehandelt. Aber auch Oberflächenobjekte können hier erzeugt werden. (TIPP: Nutzen Sie den Abschnitt nach der InitializeComponent() Methode im Konstruktor). Zur Bekanntmachung der im XAML definierten Objekte wird die *„Name“* Eigenschaft verwendet. Mit diesem Namen lassen sich die Objekte im C# Code ansprechen.

```
<WrapPanel Name="MeinContainer">
   …
</WrapPanel>
```

   Das Setzen der Eigenschaften erfolgt per Wertzuweisung unter Benutzung des Namens der Eigenschaft.

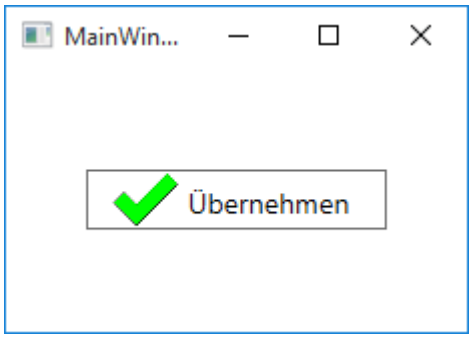
```
public MainWindow()
{
  InitializeComponent();
  TextBox tb = new TextBox();
  tb.NameDerEigenschaft = Wert;  // z.B. tb.Text = „Hallo Welt“;

  // Auch möglich:
  // tb.SetValue(TextBox.TextProperty,„Hallo Welt“);

  …

  // Textbox zu Container hinzufügen
  MeinContainer.Children.Add(tb);
}
```

   Realisieren Sie Aufgabenteil a) erneut in C#. Jedes der **TextBox** Elemente soll den Namen und die Beschriftung „TB_B_[*Nummer der Textbox*]“ erhalten also z.B. TB_B_1, TB_B_2, …

c) Erhöhen Sie die Anzahl der Textboxen auf 50. Dies können Sie entweder mit Copy & Paste erreichen (nicht empfohlen) oder mit Hilfe eines Programmiersprachenkonstrukts.

d) Die Eigenschaft eines Buttons zum Hinzufügen von Text kann auch genutzt werden, um beliebige weitere Oberflächenelemente in den Button einzufügen. Die Eigenschaft des Buttons akzeptiert zwar nur ein Objekt, aber da es unerheblich ist, welches Objekt dies ist, kann es sich auch um einen Container mit mehreren, sich darin befindlichen Oberflächenelementen handeln.

Fügen Sie dem Button aus Teilaufgabe 1 zusätzlich zu seiner Beschriftung ein kleines Bild hinzu. Das Bild soll sich links neben dem Beschriftungstext befinden. Hierfür wird ein Container mit je einem Element zur Darstellung des Bildes und zur Darstellung des Beschriftungstextes benötigt. Eine Möglichkeit, um dies zu erreichen ist die Nutzung eines Grids, wie wir es bereits kennengelernt haben. Recherchieren Sie, welcher der bereits bekannten Container ebenfalls dafür in Frage kommt und welche Eigenschaft Sie an diesem setzen müssen, um ihr Ziel zu erreichen.



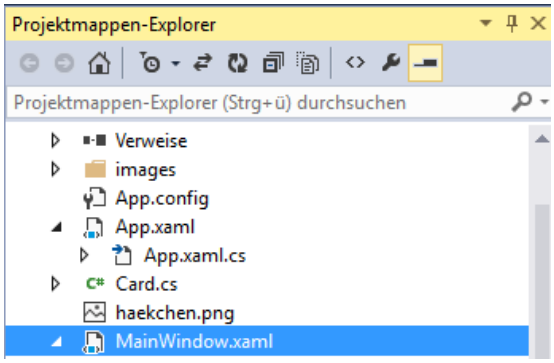
Ressourcen wie Bilder, Textdateien u.ä. können direkt im Projektverzeichnis abgelegt werden (Copy-Paste). Das **Image**-Element besitzt eine Eigenschaft „Source“ mit welchem der relative Pfad zur Bilddatei angegeben werden kann.

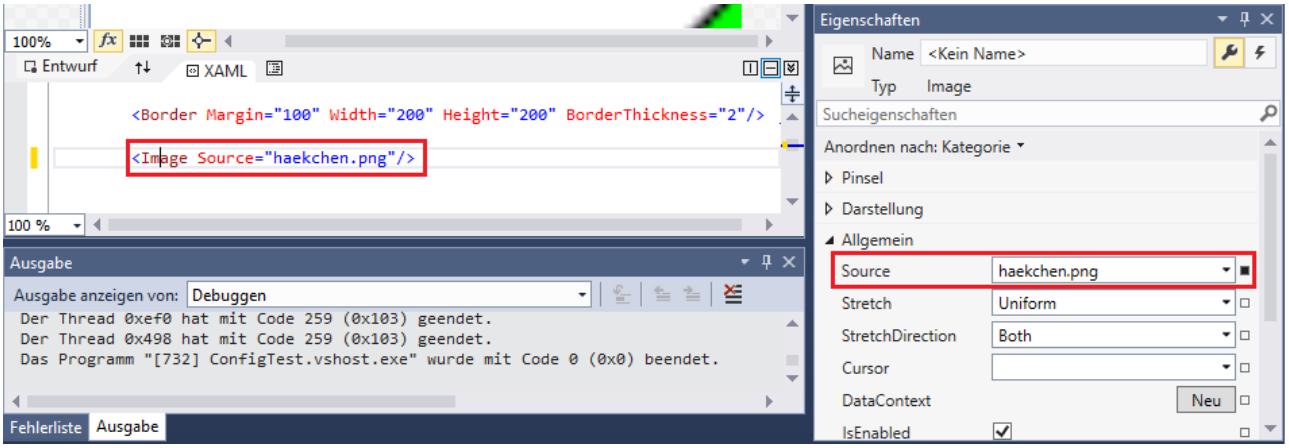
```
<Image Source="Image.png"/>
```

Hinweis: **Wenn das Bild im laufenden Programm nicht angezeigt wird, fügen Sie es direkt in Visual Studio (Copy-Paste) ein und wählen Sie die Datei explizit über den Eigenschaften Dialog des Image Objekts im Visual Studio Designer aus dem Drop Down Menü aus. Das Bild muss im Projektmappen explorer von Visual Studio sichtbar sein. Offenbar gibt es bei manchen Studierenden Probleme mit dem Zugriff auf Bilder über den Netzwerkpfad.**

Entwicklung interaktiver Systeme

M. Sc. Tobias Puderer – WS 22/23





3. Wie in Aufgabenteil 2 sichtbar wird, lässt sich alles was in XAML erzeugt werden kann, auch über C# erreichen. Diese Erkenntnis hilft dabei, den Aufbau des XAML Dokuments besser zu verstehen, da in XAML ebenfalls nur die Eigenschaften der zugrundeliegenden C#-Typen zugewiesen werden.

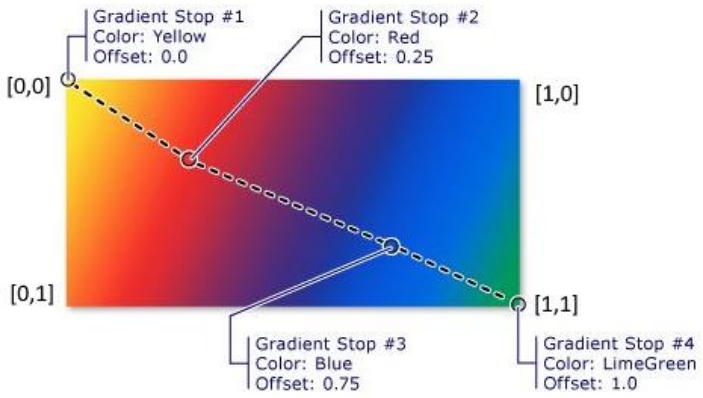
   Es kann zu Verwirrung führen, dass in XAML oft mit impliziten Typumwandlungen gearbeitet wird, von denen der Benutzer nichts bemerkt. Möchte man z.B. den Hintergrund eines `StackPanel` umfärben, kann dies – wie im nachfolgenden Beispiel gezeigt – entweder durch die direkte Zuweisung eines Farbwerts zur *Background*-Eigenschaft geschehen (implizite Typumwandlung) oder man betrachtet die *Background* Eigenschaft direkt als Objekt des C#-Typs `SolidColorBrush`. Diese beiden Varianten sind in Bezug auf ihre Bedeutung absolut gleich.

```
<StackPanel Background="Red oder #FF0000"/>
```

=

```
<StackPanel>
   <StackPanel.Background>
       <SolidColorBrush>
          <SolidColorBrush.Color>
              Red oder #FF0000
          </SolidColorBrush.Color>
       </SolidColorBrush>
    </StackPanel.Background>
</StackPanel>
```

   a) Definieren Sie nun einen beliebigen *Farbverlauf* für den Hintergrund des *Programmfensters*. Hierbei handelt es sich – wie bei `SolidColorBrush` - ebenfalls um eine Kindklasse des Typs `Brush`. Recherchieren Sie die dafür benötigten Klassen und Eigenschaften. Nutzen Sie hierzu das Internet oder das empfohlene Buch und nutzen Sie das unten gezeigte Bild als Referenz.



   b) Realisieren Sie Aufgabenteil a) auch im C# Quellcode.