Entwicklung interaktiver Systeme

M. Sc. Tobias Puderer – WS 22/23

Übung 4: Events

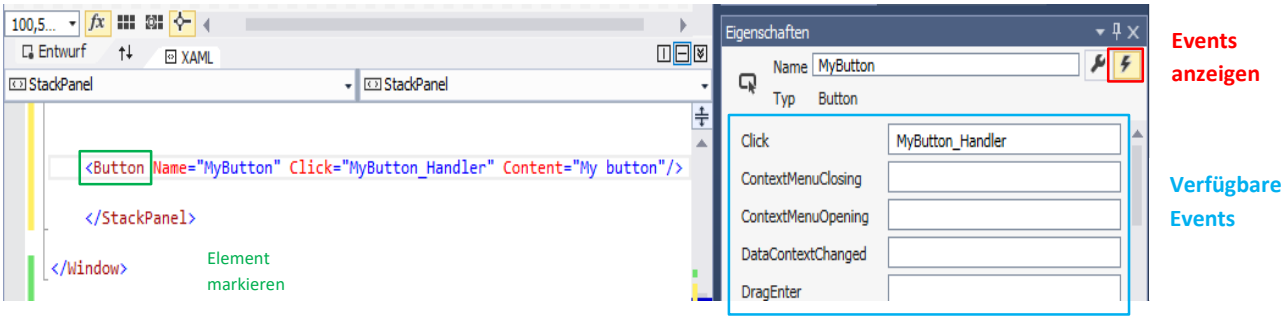
1. In einem GUI Framework muss es möglich sein, auf Benutzereingaben wie z.B. Mausklicks oder Auswahlereignisse in Listen zu reagieren. Wie eventuell bereits aus anderen GUI Frameworks bekannt, existieren hierfür vorgefertigte Events, die direkt genutzt werden können.

```
<Grid MouseLeftButtonDown="DoOnClick" Background="Red"/>
```

```
private void DoOnClick (object sender, RoutedEventArgs e)
{
        Grid b = sender as Grid; //Expliziter Cast
        //...
        MessageBox.Show("Klick Event für ein Grid");
}
```

Zu einem Event (z.B. *Click*) wird immer eine passende EventHandler-Methode (z.B. DoOnClick(…)) benötigt. Deren Signatur besteht immer aus einer Referenz auf das Objekt, welches das Event gerade behandelt (sender) sowie Informationen über das Event selbst wie z.B. der ursprüngliche Auslöser (In „e“ enthalten).

TIPP: Wenn Sie in Visual Studio innerhalb eines XAML Dokuments ein Element anklicken, so können Sie sich im „Eigenschaften“-Fenster alle für dieses Element verfügbaren Events ansehen.



a) Erstellen Sie einen Button, der bei Klick (Event *Click*) mit Hilfe einer `MessageBox` den String „Hallo Welt“ anzeigt. Die Zuweisung des Handlers soll über XAML erfolgen.

b) Wiederholen Sie Aufgabenteil a) bzw. die Zuordnung des Handlers in C#. Hierzu muss man wissen, dass dem jeweiligen Event durch Verwendung des += Operators eine oder mehrere behandelnde Methoden hinzugefügt werden kann. Bei der Zuweisung wird der Name der aufzurufenden Methode ohne die Angabe der Parameter angegeben. Orientieren Sie sich an nachfolgendem Beispiel.

```
public Uebung4() {

     InitializeComponent();

     MyButton.Click += MyButton_Handler;
     //NICHT: MyButton.Click += MyButton_Handler(object sender, RoutedEventArgs e);
}

private void MyButton_Handler(object sender, RoutedEventArgs e) {
       …
}
```

c) Fügen Sie einen zweiten Click Handler (dieser muss einen anderen Namen haben) hinzu, welcher die Hintergrundfarbe des Buttons bei jedem Klick auf eine zufällige Farbe setzt. Beide Handler sollen beim Klick auf den Button ausgeführt werden. Mit Hilfe der Methode Next() der Klasse **Random** können Zufallszahlen ermittelt werden.

```
Random rand = new Random();
rand.Next(256); // Positive Zufallszahl bis 255
```

Verschiedene Farben können Sie z.B. mit Hilfe der „FromRGB()“ Methode der Klasse **Color** erhalten. (RGB: Rot-Grün-Blau; Wertbereich von 0..255)

```
// Beispiel: Rot
Color.FromRgb(255, 0, 0);
```

Die Hintergrundeigenschaft „Background“ erwartet einen Wert vom Typ **Brush**, daher muss die Farbe vor der Zuweisung in einem solchen gekapselt werden. Da es sich um einen festen Farbwert handelt, kommt hier der Typ **SolidColorBrush** zum Einsatz.

```
byte r = (byte)rand.Next(256);
byte g = …
…
Ihr_Button.Background = new SolidColorBrush(Color.FromRgb(r, g, b));
```

Den Button können Sie entweder über seinen Namen (In XAML definiert) ansprechen, oder explizit das „sender“ Argument der Handler Methode per Cast in den Typ **Button** umwandeln. Die Syntax der Typumwandlung (Cast) entspricht der in Java.

d) Erstellen Sie eine `TextBox`, die beim Überfahren mit der Maus die Hintergrundfarbe und Textfarbe ändert. Sobald der Bereich der `TextBox` mit der Maus verlassen wird, soll die alte Konfiguration wieder hergestellt werden.

Setzen Sie dies mit Hilfe einer Kombination von XAML und Code-Behind um, indem Sie den EventHandler im XAML registrieren und die Designänderungen im C# Code realisieren. Welches Event kommt hierfür in Frage? Ermitteln Sie aus der Liste der verfügbaren Events, welches für die gewünschte Funktionalität benötigt wird. Die Hintergrund-Eigenschaft der `TextBox` kann über das „sender“ Objekt modifiziert werden, indem dieses explizit auf `TextBox` gecastet wird (Vgl. Aufgabenbeschreibung).

2. Malprogramm

InkCanvas ist ein sehr mächtiges Oberflächenelement, das es (unter anderem) erlaubt, Zeichnungen auf dem Bildschirm anzufertigen.

Platzieren Sie ein **InkCanvas** Element und einige Steuerelemente, die die Eigenschaften des Pinsels verändern können. Ziehen sie zur Ausrichtung der Elemente den Einsatz eines **DockPanel** in Kombination mit **ToolBarTrays** in Betracht.

```
<DockPanel LastChildFill="true">
  <ToolBarTray DockPanel.Dock="Right" Orientation="Vertical">
     <ToolBar>
       <Button … />
     </ToolBar>
  </ToolBarTray>

  <InkCanvas Name="canvas" />

</DockPanel>
```

Manche Attribute des **InkCanvas** wie z.B. Farbe und Größe des Pinsels müssen über dessen Eigenschaft „DefaultDrawingAttributes“ gepflegt werden, wohingegen andere (Pinsel oder Radierermodus) direkt am Objekt gesetzt werden müssen.

```
canvas.DefaultDrawingAttributes.Color = …   // Pinselfarbe

canvas.EditingMode = …  // Stift oder Radierer?
```

- Es soll möglich sein, zwischen den Farben Rot, Grün und Blau zu wechseln. Realisieren Sie dies über **Buttons.**

- Es sollen 3 verschiedene Pinselgrößen angeboten werden. Realisieren Sie dies über eine **ListBox**.

- Es soll möglich sein zwischen Pinsel und Radiergummi zu wechseln. Realisieren Sie dies über **RadioButtons** oder eine **ComboBox**.

Um die Übung etwas interessanter zu machen, soll es möglich sein, das gezeichnete Bild abzuspeichern. Nutzen Sie dafür folgendes Code-Snippet.

```
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = "Image"; // Default Dateiname
dlg.DefaultExt = ".png"; // Default Dateierweiterung
dlg.Filter = "Image Files (.png)|*.png"; // Typfilter für Dateispeicherdialog

// Zeige den Dateispeicherdialog
Nullable<bool> result = dlg.ShowDialog();

// Verarbeite Ergebnis. Nur wenn “OK” geklickt wird, wird auch gespeichert!
if (result == true)
  {
    // Speichere die Bilddaten in die Datei
    string filename = dlg.FileName;

    MemoryStream ms = new MemoryStream();
    FileStream fs = new FileStream(filename, FileMode.Create);

    RenderTargetBitmap rtb = new
RenderTargetBitmap((int)InkCanvas.ActualWidth, (int)InkCanvas.ActualHeight,
96d, 96d, PixelFormats.Default);
    rtb.Render(InkCanvas);
    PngBitmapEncoder encoder = new PngBitmapEncoder();
    encoder.Frames.Add(BitmapFrame.Create(rtb));

    encoder.Save(fs);
    fs.Close();
  }
```

Abhängig davon, wie das InkCanvas Element im XAML benannt wurde