Einfacher Taschenrechner mit WPF

Objektbasiert programmieren nach Vorgabe

Inhaltsverzeichnis

1 Ziel

Der Lernende macht die ersten Erfahrungen wie eine Benutzerschnittstelle für einen einfachen Taschenrechner erstellt wird. Ziel dieser Aufgabe ist es, dem Lernenden einfache Steuerelemente zu zeigen und die Verwendung dieser im Programm umzusetzen. Dieses Dokument hat aber nicht den Anspruch der Vollständigkeit und der perfekten Umsetzung des Problems. Es soll nur der Einstieg in die GUI-Programmierung erleichtert werden. In den nachfolgenden Dokumenten werden zu einem späteren Zeitpunkt weitere Steuerelemente und Programmiertechniken erklärt und angewendet.

2 Ausgangslage

In dieser Aufgabe wollen wir einen einfachen Taschenrechner nach folgendem Muster erstellen:

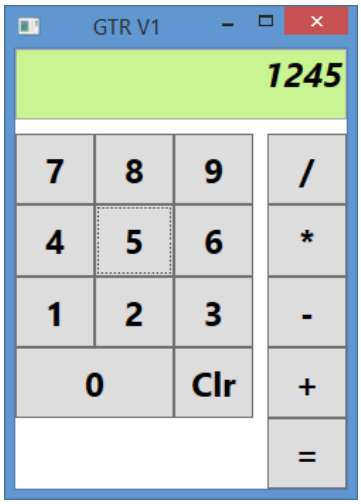

Abbildung 1: GTR V1

Wie Sie aus nebenstehendem Bild erkennen können, handelt es sich um einen einfachen „Ganzzahl"-Taschenrechner. Das heisst, wir können selber keinen Dezimalpunkt setzen und die Resultatausgabe erfolgt immer als abgerundete Ganzzahl.

Folgende Tasten stehen beim Taschenrechner zur Verfügung:

Tasten:	**0 – 9**	Ziffern für Zahleingabe
Tasten:	**/**, *****, **-**, **+**	mögliche Rechenoperationen
Taste	**=**	erzeugt die Resultatausgabe
Taste	**Clr**	löscht die Anzeige

Damit eine Division durch 0 (/ 0) nicht durchgeführt werden kann, muss dies überprüft werden. Sollte dieser Fall eintreten, so muss der Benutzer in geeigneter Weise informiert werden.

Der Taschenrechner selbst soll in der Darstellung frei verändert werden können. Sie untenstehende Beispiele:

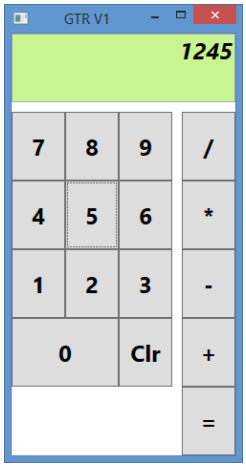

Abbildung 2: GTR V1 schmal

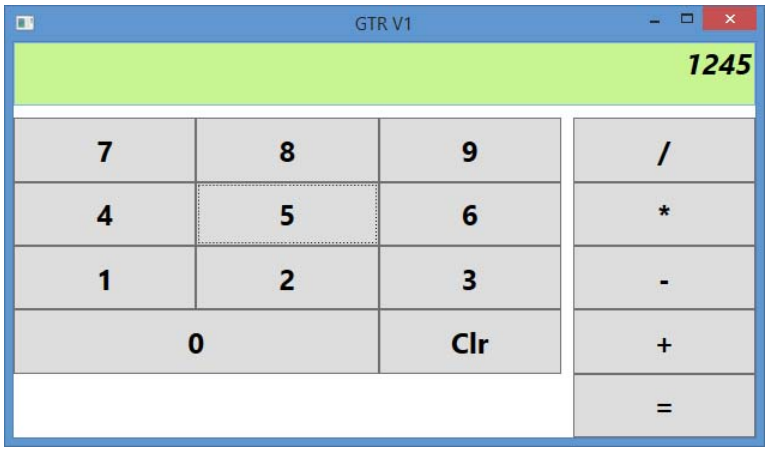

Abbildung 3: GTR V1 lang gezogen

3 Aufgabenstellungen

Wir werden nun obige Aufgabenstellung Schritt für Schritt umsetzen und zu den einzelnen Bereichen weitere Informationen erhalten.

3.1 Erstellung des Projektes für eine WPF-Anwendung

Über: `DATEI -> NEU -> Projekte…` gelangen Sie zum Projekterstellungsfenster.

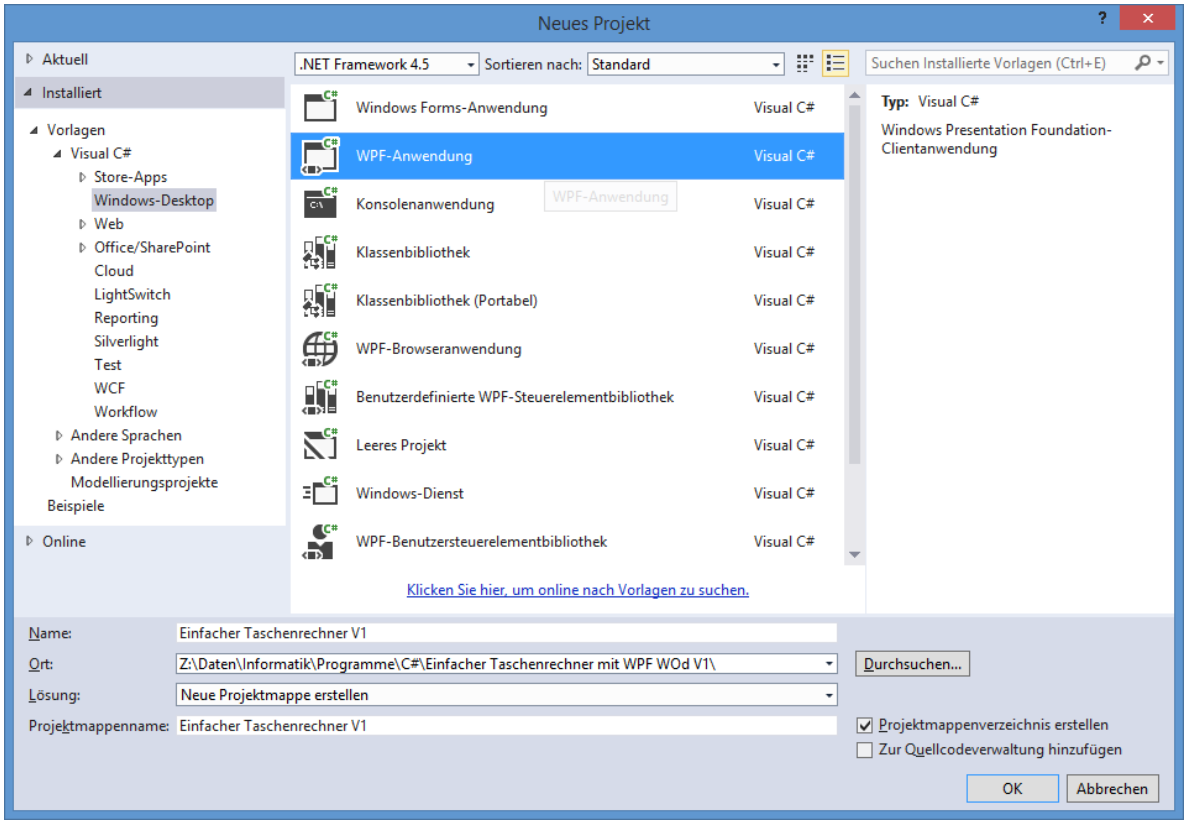

Abbildung 4: Einstellungen im Projekterstellungsfenster

Nachdem Sie alle obigen Einstellungen so eingegeben haben und dies mit der Taste OK bestätigt haben, öffnet sich das Programmentwicklungsfenster.

In diesem Fenster sehen Sie 2 Files welche automatisch für sie erzeugt wurden. Es sind dies:

- `MainWindow.xaml` und
- `MainWindow.xaml.cs`

Beide Files werden benötigt um unser gefordertes Programm zu erstellen. Diese stehen sehr eng miteinander in Verbindung und dürfen auf keinen Fall gelöscht werden. In beiden Files werden wir jetzt Programmcode hinzufügen um unseren Taschenrechner zu erstellen.

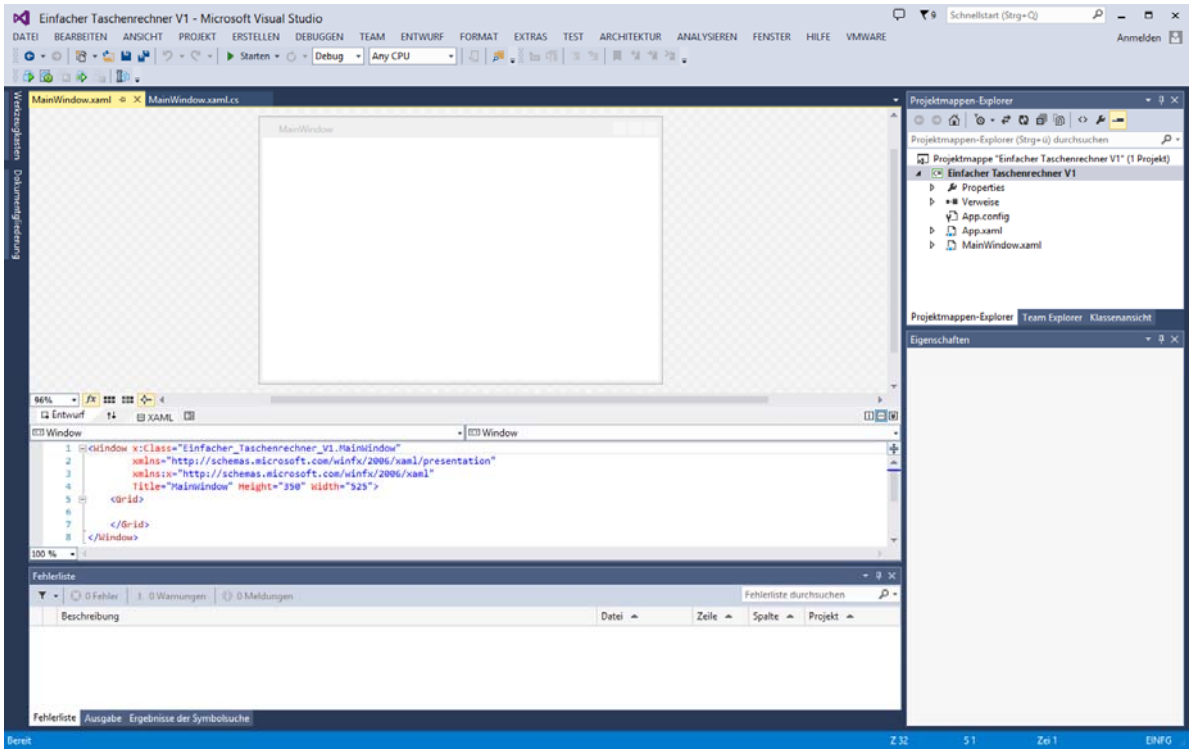

Abbildung 5: Programmentwicklungsfenster

3.2 Erstellung des GUI

Um dies zu tun, wird das File: `MainWindow.xaml` Inhaltlich nach unseren Vorstellungen abgeändert und erweitert.

3.2.1 Änderung der Startgrösse unseres Fensters

Wenn wir unser „MainWindow“ ansehen, so stellen wir fest, dass im File: `MainWindow.xaml` gewisse Standardwerte bereits eingetragen sind. Im Originalzustand sieht unser File wie folgt aus:

```
<Window x:Class="Einfacher_Taschenrechner_V1.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>

    </Grid>
</Window>
```

Abbildung 6: File: MainWindow.xaml (Originalzustand)

Abbildung 7: Original: MainWindow

Damit unser dargestelltes Fenster die richtig Startgrösse erhält, muss als erstes der Parameter: `Width` auf den Wert: `250` angepasst werden.

```
Width="250"
```

```
<Window x:Class="Einfacher_Taschenrechner_V1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="250">
    <Grid>

    </Grid>
</Window>
```

Abbildung 8: File: MainWindow.xaml (Fenstergrösse verändert)

3.2.2 Erstellung eines 5 x 7 – Rasters im MainWindow, welches frei skalierbar ist

In der Aufgabenstellung wurde gefordert, dass wir den Taschenrechner in der Grösser verändert können. Um dies zu ermöglichen, verwenden wir einen sogenannten: „Grid-Container".

Im File: `MainWindow.xaml` werden wir nun als erstes unser gefordertes Raster aufbauen.

3.2.2.1 Erzeugung frei skalierbarer Spalten

Um ein 5x7-Raster zu erstellen, benötigen wird die folgenden Befehle, welche wir im `MainWindow.xaml` – Fenster einfügen:

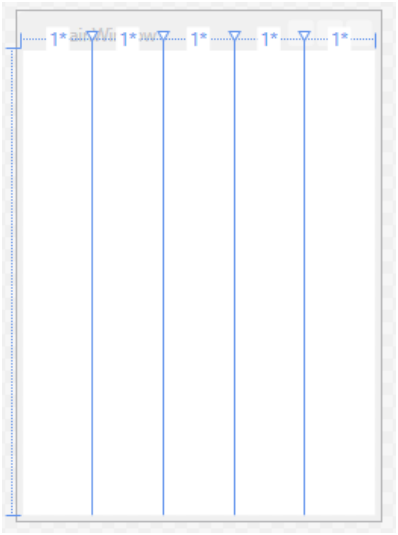

Abbildung 9: MainWindow mit 5 identischen Spalten

Als erstes wollen wir 5-Spalten im MainWindow einfügen. Dies wird mit der folgenden Befehlsabfolge verwirklicht, welche zwischen `<Grid>` und `</Grid>` eingefügt werden:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

Jedes Mal, wenn der Befehl:

```
<ColumnDefinition Width="*" />
```

verwendet wird, erzeugen wir eine weitere Spalte. Mit dem `"*"` Symbol wird angegeben, dass die Gesamtbreite gleichmässig aufgeteilt wird auf die einzelnen Spalten.

```
<Window x:Class="Einfacher_Taschenrechner_V1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="250">
    <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    </Grid>
</Window>
```

Abbildung 10: File: MainWindow.xaml (Fenster mit 5 gleichen Spalten)

3.2.2.2 Erzeugung frei skalierbarer Zeilen

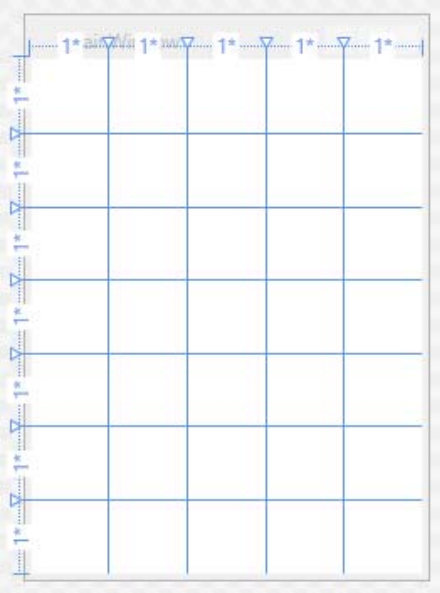

Abbildung 11: MainWindow mit einer identischen 5x7 Rasteraufteilung

Im nächsten Schritt fügen wir 7 Zeilen ein, welche alle die gleiche Höhe haben sollen. Um dies zu tun, fügen wir direkt nach `</Grid.ColumnDefinitions>` die folgenden Befehle ein:

```
<Grid.RowDefinitions>
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
  <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

Jedes Mal, wenn der Befehl:

```
<RowDefinition Height="*" />
```

verwendet wird, erzeugen wir eine weitere Zeile. Mit dem `"*"` Symbol wird angegeben, dass die Gesamthöhe gleichmässig aufgeteilt wird auf die einzelnen Zeilen.

```
<Window x:Class="Einfacher_Taschenrechner_V1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="250">
    <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    </Grid>
</Window>
```

Abbildung 12: File: MainWindow.xaml (Fenster mit einer identischen 5x7 Rasteraufteilung)

3.2.2.3 Anpassung des Rasters (Spaltenbreite, Zeilenhöhe)

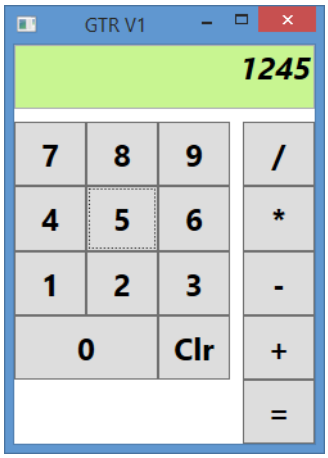

Abbildung 13: GTR V1

Wenn Sie das vorgegeben GUI betrachten, fällt Ihnen sicherlich auf, dass nicht alle Spalten und Zeilen identisch gross sind. Um dies zu verwirklichen, wird an der betreffenden Stelle das `"*"` Symbol in den Befehlen: `<ColumnDefinition Width="*" />` und `<RowDefinition Height="*" />` mit einem Integer Wert ersetzt. Der Wert den Sie hier angeben hat die Einheit Pixel.

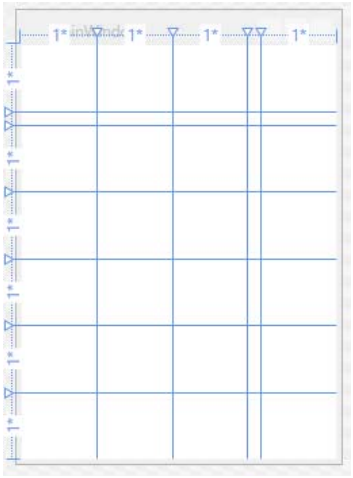

Abbildung 14: MainWindow mit angepassten Spalten und Zeilen

Da dies bei bei der 4. Spalte und der 2. Zeile der Fall ist, müssen diese Codelinien dementsprechend bearbeitet werden. Wir ersetzen diese Codezeilen wie folgt:

```
<RowDefinition Height="10" />
<ColumnDefinition Width="10" />
```

Das Resultat sehen Sie beim nebenstehenden Bild.

```
<Window x:Class="Einfacher_Taschenrechner_V1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="250">
    <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="10" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="10" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    </Grid>
</Window>
```

Abbildung 15: MainWindow.xaml (Fenster mit angepassten Spalten und Zeilen)

3.2.3 Hinzufügen der Steuerelemente in das Raster

3.2.3.1 Hinzufügen der Schaltflächen (Buttons)

Als nächsten Schritt wollen wir die sogenannten „Buttons“ dem Raster zuordnen. Dies geschieht wie folgt:

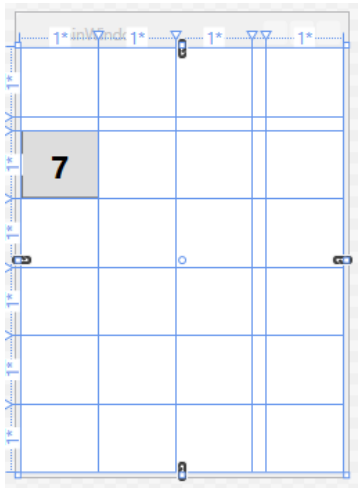

Abbildung 16: Taste 7 dem Raster zuweisen

Um z.B. die Taste: 7, wie im nebenstehenden Bild gezeigt, dem Raster zuzuordnen, muss der folgende Befehl verwendet werden:

```
<Button x:Name="cmd7" Grid.Column="0" Grid.Row="2"
          Content="7" FontSize="24" FontWeight="Bold">
</Button>
```

Diese Befehlszeile muss direkt nach `</Grid.RowDefinitions>` hinzugefügt werden.

Dabei haben die Parameter die folgende Bedeutung:

`x:Name`	Gibt den Namen der Taste an, mit der er im Programmcode angesprochen werden kann.
`Grid.Column`	Gibt die Spalte an, wo diese Taste platziert werden soll.
`Grid.Row`	Gibt die Zeile an, wo diese Taste platziert werden soll.
`Content`	Gibt die Bezeichnung an, welche auf der Taste erscheinen soll.
`FontSize`	Gibt die Schriftgrösse der Bezeichnung an.
`FontWeight`	Gibt die Auszeichnung der Bezeichnung an. (z.B. **fett**)

Wenn wir nun alle Schalter unseres Taschenrechners hinzufügen, erhalten wir den folgenden Code:

```
<Button x:Name="cmd7" Grid.Column="0" Grid.Row="2" Content="7" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd8" Grid.Column="1" Grid.Row="2" Content="8" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd9" Grid.Column="2" Grid.Row="2" Content="9" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd4" Grid.Column="0" Grid.Row="3" Content="4" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd5" Grid.Column="1" Grid.Row="3" Content="5" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd6" Grid.Column="2" Grid.Row="3" Content="6" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd1" Grid.Column="0" Grid.Row="4" Content="1" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd2" Grid.Column="1" Grid.Row="4" Content="2" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd3" Grid.Column="2" Grid.Row="4" Content="3" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmd0" Grid.Column="0" Grid.Row="5" Content="0" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmdClr" Grid.Column="2" Grid.Row="5" Content="Clr" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmdDiv" Grid.Column="4" Grid.Row="2" Content="/" FontSize="24"
        FontWeight="Bold"/>
```

```
<Button x:Name="cmdMul" Grid.Column="4" Grid.Row="3" Content="*" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmdSub" Grid.Column="4" Grid.Row="4" Content="-" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmdAdd" Grid.Column="4" Grid.Row="5" Content="+" FontSize="24"
        FontWeight="Bold"/>
<Button x:Name="cmdRes" Grid.Column="4" Grid.Row="6" Content="=" FontSize="24"
        FontWeight="Bold"/>
```

Das Fenster unseres Taschenrechners sieht jetzt wie folgt aus (linkes Bild):

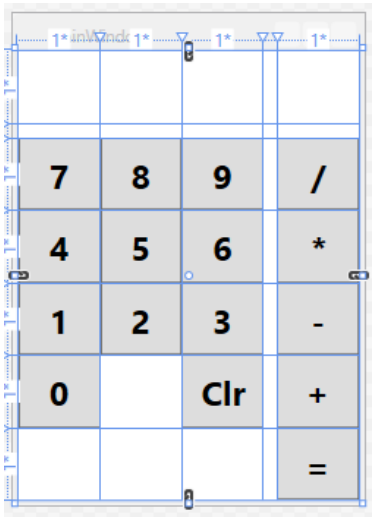

Abbildung 17: Zuweisung aller Tasten zum Raster

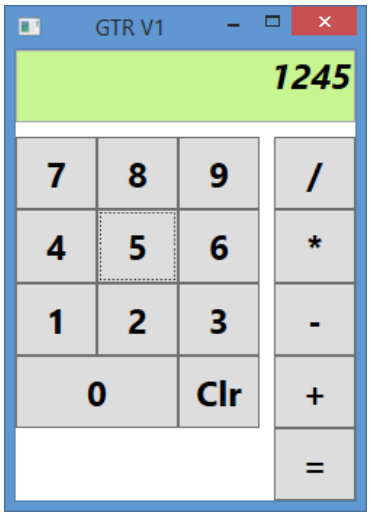

Abbildung 18: Gefordertes GUI

Im linken Fenster stellen wir fest, dass die Taste: 0 nicht wie gefordert über 2 Felder gelegt wurde. Dieser Bereich muss nun mittels dem Parameter:

```
Grid.ColumnSpan="2"
```

bei der Taste: 0 angepasst werden

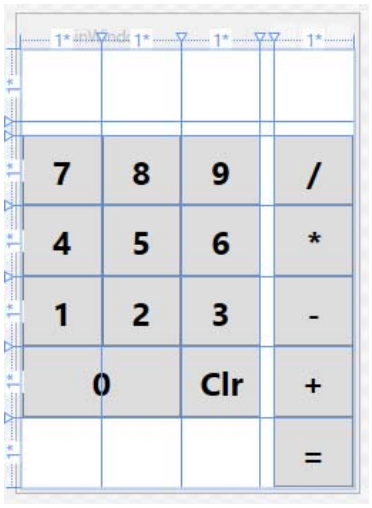

Abbildung 19: Korrekte Darstellung der Tasten

Somit lautet der vollständige Code für die Taste: 0 wie folgt:

```
<Button x:Name="cmd0" Grid.Column="0"
        Grid.ColumnSpan="2" Grid.Row="5" Content="0"
        FontSize="24" FontWeight="Bold"/>
```

Hinweise:

- *Es wurden im obigen Beispiel nur einzelne Parameter der Taste angegeben. In der Praxis gibt es noch einige mehr die zur Anwendung kommen können.*
- *Mittels dem Eigenschaftsfenster der dazugehörenden Taste können auf einfache Weise weitere Parameter hinzugefügt oder bestehende Parameter im Wert verändert werden.*

3.2.4 Hinzufügen einer TextBox für die Ausgabe des Resultates

Bis zum jetzigen Zeitpunkt haben wir alle Tasten in der gewünschten Form auf dem Fenster angeordnet. Was uns aber noch fehlt ist ein Ausgabefeld für die einzugebende Zahl und das Resultat. Dies wollen wir jetzt ändern, indem wir ein eine TextBox, wie im geforderten GUI angegeben, platzieren.

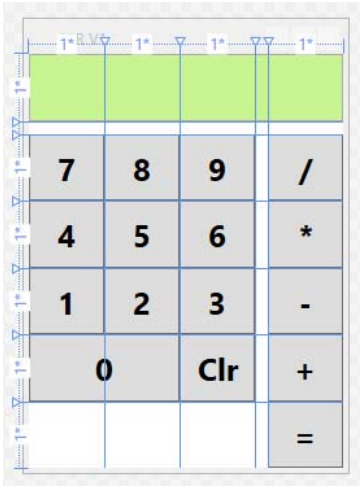

Abbildung 20: Vollständig erstelltes GUI

Der Befehl hierzu lautet:

```
<TextBox x:Name="txtAnzeige" Grid.Column="0"
         Grid.Row="0" Grid.ColumnSpan="5"
         FontWeight="Bold" FontSize="24"
         FontStyle="Italic" TextAlignment="Right"
         Background="#FFC8F591">
</TextBox>
```

Neben den bereits bekannten Parametern finden wir hier noch drei weitere Parameter. Es sind dies:

`FontStyle`	Mit diesem Befehl kann der auszugebende Text in der Darstellung angepasst werden. (z.B. *Italic*)
`TextAlignment`	Gibt an, ob der Text rechts oder linksbündig ausgegeben werden soll.
`Background`	Farbwert des Hintergrundes. Am einfachsten durch das Eigenschaftsfeld einstellbar.

3.3 Ausführbare Methoden beim Drücken einer Taste

Bis jetzt haben wir nur die Oberfläche des GUI erstellt, ohne dass dabei der Benutzer die Möglichkeit hat etwas einzugeben, indem er die entsprechende Taste anklickt. Damit wir unseren Taschenrechner auch benützen können, müssen wir diese Funktionalität implementieren. Die Implementation diese Codes wird im Fiel: MainWindow.xaml.cs eingefügt.

Bevor wird dies jedoch machen, wollen wir das automatisch erzeugte Originalfile einmal genauer anschauen.

3.3.1 Originalfile: MainWindow.xaml.cs, welches automatisch erstellt wurde

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Einfacher_Taschenrechner_V1
{
  /// <summary>
```

```
  /// Interaktionslogik für MainWindow.xaml
  /// </summary>
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
    }
  }
}
```

Abbildung 21: Originalfile: MainWindow.xaml.cs

An diesem File können wir erkennen, dass beim Start die Komponenten automatisch initialisiert werden (`InitializeComponent()`). Weitere Methoden sind bis jetzt aber nicht vorhanden.

3.3.2 Hinzufügen von Ereignismethoden

Generell muss gesagt werden, dass beim Drücken einer Taste eine Ereignisbehandlung stattfinden muss. Wie dies ganz genau funktioniert, wird in einem späteren Kapitel erklärt werden. Zum jetzigen Zeitpunkt müssen wir nur wissen, dass durch einen Doppelklick auf eine Taste unseres GUIs automatisch eine Methode im File: MainWindow.xaml.cs erstellt wird, welche einen leeren Rumpf beinhaltet.

Wird zum Beispiel ein Doppelklick auf die Taste: 0 gemacht, so wird der bestehende Code im File: `MainWindow.xaml.cs` wie folgt erweitert:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Einfacher_Taschenrechner_V1
{
  /// <summary>
  /// Interaktionslogik für MainWindow.xaml
  /// </summary>
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
    }

    private void cmd0_Click(object sender, RoutedEventArgs e)
    {

    }
  }
}
```

Abbildung 22: Erstellung des leeren Rumpfes der Ereignismethode für die Taste: 0

Ein weitere Veränderung hat sich durch diesen Doppelklick auf die Taste: 0 im File: `MainWindow.xaml` ergeben. Hier wurde der Code für die Taste: 0 wie folgt erweitert:

```
<Button x:Name="cmd0" Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="5" Content="0"
        FontSize="24" FontWeight="Bold" Click="cmd0_Click"/>
```

Diese Erweiterung der Zeile gibt nun an, dass beim Klick auf dies Taste: 0 die automatisch mit leerem Rumpf erzeugte Methode: `cmd0_Click()` im File: `MainWindow.xaml.cs` aufgerufen werden soll. Diese ist aber bis zum jetzigen Zeitpunkt leer und muss dementsprechend mit Code gefüllt werden.

3.3.3 Code der Ereignismethoden

Nachfolgend finden Sie den ganzen Code für die einzelnen Tasten, welche beim Anklicken eine Ereignismethode aufrufen werden. Neben diesen Ereignismethoden wurde auch noch eine weitere Methode: `SetAnzeige()` eingefügt. Studieren Sie zuerst diesen Code und fügen Sie diesen dann in Ihre Applikation ein.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Einfacher_Taschenrechner_mit_WPF_V1._0
{
  /// <summary>
  /// Interaktionslogik für MainWindow.xaml
  /// </summary>
  public partial class MainWindow : Window
  {
    //lokale Variablen
    private long zwischenwert = 0;
    private long resultat = 0;
    private char operation = '+';


    public MainWindow()
    {
      InitializeComponent();
    }

    private void cmd0_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10;
      SetAnzeige();
    }

    private void cmd1_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 1;
```

```
      SetAnzeige();
    }

    private void cmd2_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 2;
      SetAnzeige();
    }

    private void cmd3_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 3;
      SetAnzeige();
    }

    private void cmd4_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 4;
      SetAnzeige();
    }

    private void cmd5_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 5;
      SetAnzeige();
    }

    private void cmd6_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 6;
      SetAnzeige();
    }

    private void cmd7_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 7;
      SetAnzeige();
    }

    private void cmd8_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 8;
      SetAnzeige();
    }

    private void cmd9_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = zwischenwert * 10 + 9;
      SetAnzeige();
    }

    private void cmdClr_Click(object sender, RoutedEventArgs e)
    {
      zwischenwert = 0;
      SetAnzeige();
    }

    private void cmdAdd_Click(object sender, RoutedEventArgs e)
    {
      resultat = zwischenwert;
      operation = '+';
      zwischenwert = 0;
      SetAnzeige();
```

```
    }

    private void cmdSub_Click(object sender, RoutedEventArgs e)
    {
      resultat = zwischenwert;
      operation = '-';
      zwischenwert = 0;
      SetAnzeige();
    }

    private void cmdMul_Click(object sender, RoutedEventArgs e)
    {
      resultat = zwischenwert;
      operation = '*';
      zwischenwert = 0;
      SetAnzeige();
    }

    private void cmdDiv_Click(object sender, RoutedEventArgs e)
    {
      resultat = zwischenwert;
      operation = '/';
      zwischenwert = 0;
      SetAnzeige();
    }

    private void cmdRes_Click(object sender, RoutedEventArgs e)
    {
      switch (operation)
      {
        case '+':
          {
            resultat += zwischenwert;
            break;
          }
        case '-':
          {
            resultat -= zwischenwert;
            break;
          }
        case '*':
          {
            resultat *= zwischenwert;
            break;
          }
        case '/':
          {
            if (zwischenwert == 0)
            {
              MessageBox.Show("Achtung: Division duch 0 ist nicht erlaubt!");
            }
            else
            {
              resultat /= zwischenwert;
            }
            break;
          }
      }
      zwischenwert = resultat;
      SetAnzeige();
    }

    private void SetAnzeige()
```

```
    {
      txtAnzeige.Text = zwischenwert.ToString();
    }
  }
}
```

Abbildung 23: Kompletter Code der Ereignismethoden des Taschenrechners

4 Geforderte Lösungsbereiche

- Erzeugung des GUI-Oberfläche nach Vorgabe
- Einfügen der Ereignismethoden
- Erklärung der Funktionsweise des Codes
- Auflistung von Verbesserungsmöglichkeiten des GUI

5 Hilfsmittel

Visual Studio

6 Zeitbedarf

ca. 180 Minuten

7 Abbildungsverzeichnis

Historie

Vers.	Bemerkungen	Verantwortl.	Datum
1.0	- Komplette Aufgabe in C# erstellt	W. Odermatt	01.11.2015

Referenzunterlagen

LfNr.	Titel / Autor / File / Verlag / ISBN	Dokument-Typ	Ausgabe
1	„Programmiersprache C“, Implementierung C, Grundkurs / H.U. Steck	Theorieunterlagen	2003
2	„Programmieren in C“ / W. Sommergut / Beck EDV-Berater im dtv / 3-423-50158-8	Fachbuch	1997
3	„C Kompakt Referenz“ / H. Herold / Addison Wesley / 3-8273-1984-6	Fachbuch	2002
4	„C in 21 Tagen“ / P. Aitken, B. L. Jones / Markt + Technik / 3-8272-5727-1	Fachbuch	2000
5	„C Programmieren von Anfang an“ / H. Erlenkötter / rororo / 3-499-60074-9	Fachbuch	2001
6	„Programmieren in C“ / B.W. Kernighan, D.M. Ritchie / Hanser / 3-446-25497-3	Fachbuch	1990
7	„Einstieg in Visual C# 2013“ / Thomas Theis / Galileo Computing / 978-3-8362-2814-5	Fachbuch	2014
8	“Visual C# 2012” / Andreas Kühnel / Rheinwerk Computing / 978-3-8362-1997-6	Fachbuch	2013
9	“Objektorientiertes Programmieren in Visual C#” / Peter Loos / Microsoft Press / 3-86645-406-6	Fachbuch	2006