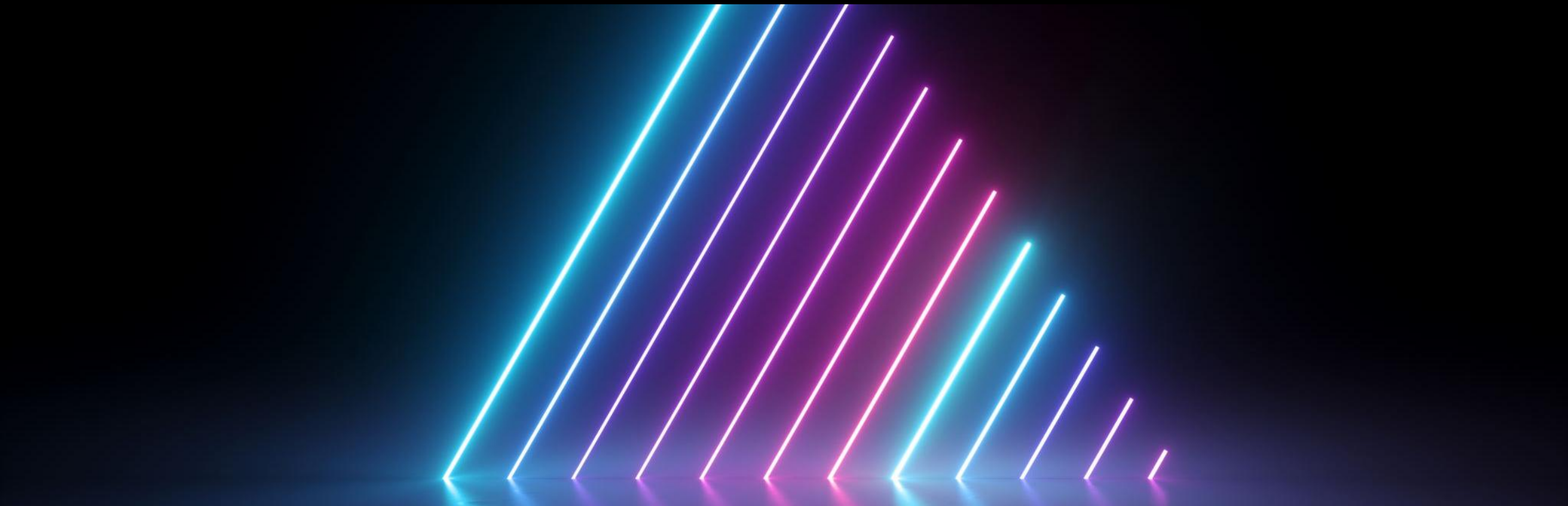
CSharp

07.04.2025



Plan für die Woche

SMART INDUSTRY CAMPUS

Montag

- Wiederholung Java vs. C#
- Übungen

Dienstag

- GUI-Programmierung in C# mit Hilfe der Toolbox in VS
- Implementierung der GUI um Bücher hinzuzufügen

Mittwoch

- Datenbankanbindung mit C#
- Implementierung der DAO-Schicht in C#

Donnerstag

- Spiele-Projekt

Freitag

- Spiele-Projekt

Plan für heute

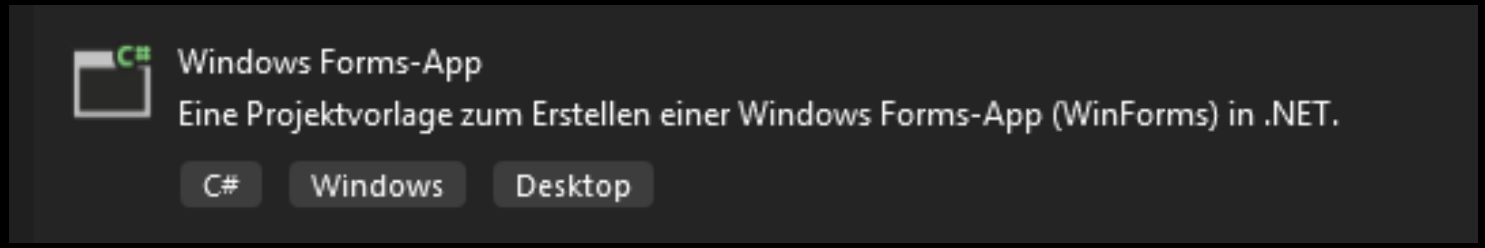
- ✓ Erzeugen einer Graphischen Oberfläche
- ✓ Nutzen mehrerer Fenster

Wie erzeugt man ein Forms Projekt?

Neues Projekt anlegen

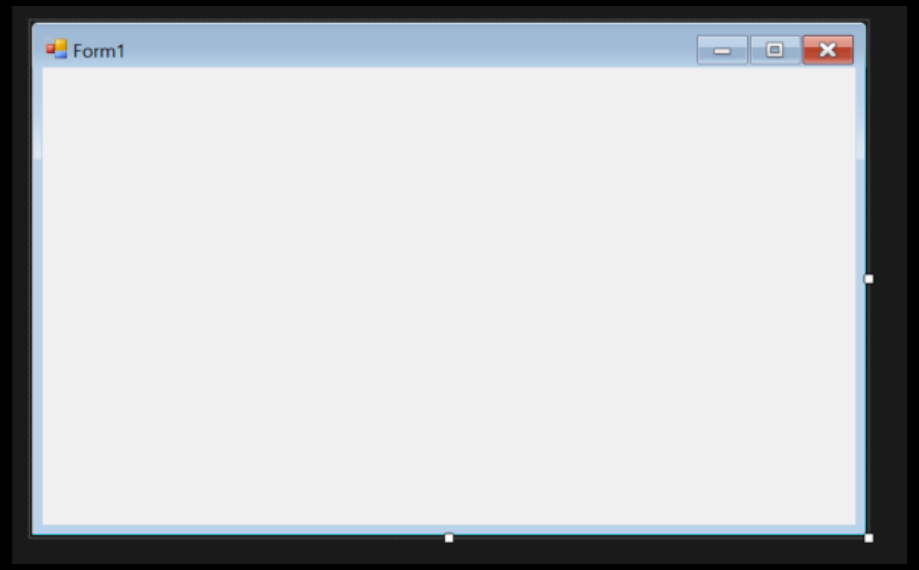
→ In Visual Studio ein neues Projekt anlegen (Datei – Neu – Projekt)

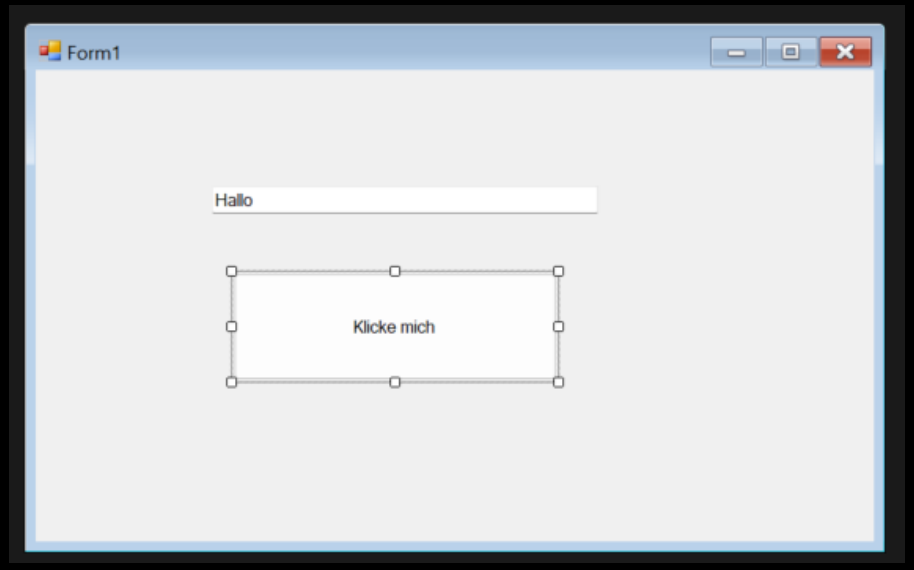
→ Windows Forms-App auswählen



Neues Projekt anlegen

- → Es öffnet sich ein Fenster
- → Mittels Str+Alt+X wird die ToolBox geöffnet
- → Aus der ToolBox kann man verschiedene Komponenten in das Hauptfenster hineinziehen



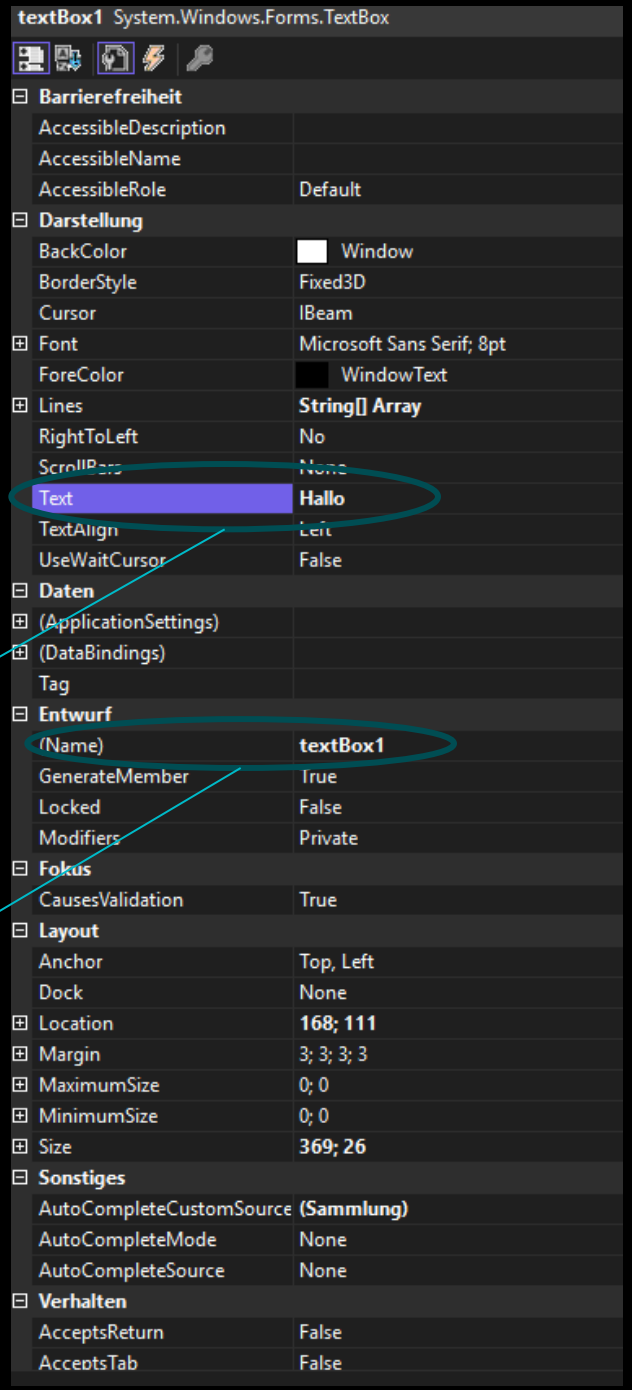


Fenster bearbeiten

Komponenten bearbeiten

→ Mit einmaligem Klick kann man die Komponenten bearbeiten (oder Rechtsklick – Eigenschaften)

→ Größe, Name, Inhalt, …



Was steht in dem Feld?

Wie heißt die Variable im Code?

On-Click-Funktion hinzufügen

→ Mit Doppelklick kann man definieren, was beim Klicken passieren soll

```
namespace DatenbankProjekt
{
    3 Verweise
    public partial class Form1 : Form
    {
        1 Verweis
        public Form1()
        {
            InitializeComponent();

        }

        1 Verweis
        private void click_me_button_Click(object sender, EventArgs e)
        {
            DBCreator db = new DBCreator();
            db.CreateDatabase();
        }

    }
}
```

Was passiert beim Klick?

Aufgabe  *ca. 30 Minuten*

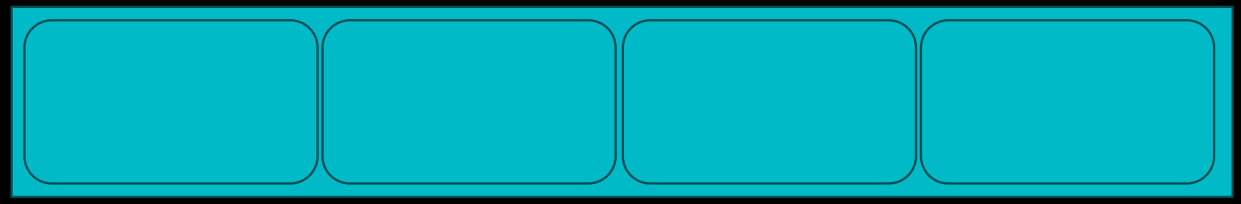
- → Spielt mit VS und Forms herum
- → Grundvoraussetzung:
  - → Erstellt ein Fenster mit einem Button und einem Textfeld
    - → Button hat den Text „Klicke mich“, die Variable „clickMeButton“
    - → Textfeld soll als initialen Text „Hallo“ haben
    - → Beim Klicken des Buttons soll der Text von „Hallo“ zu „Ich wurde geklickt“ werden

Layouts

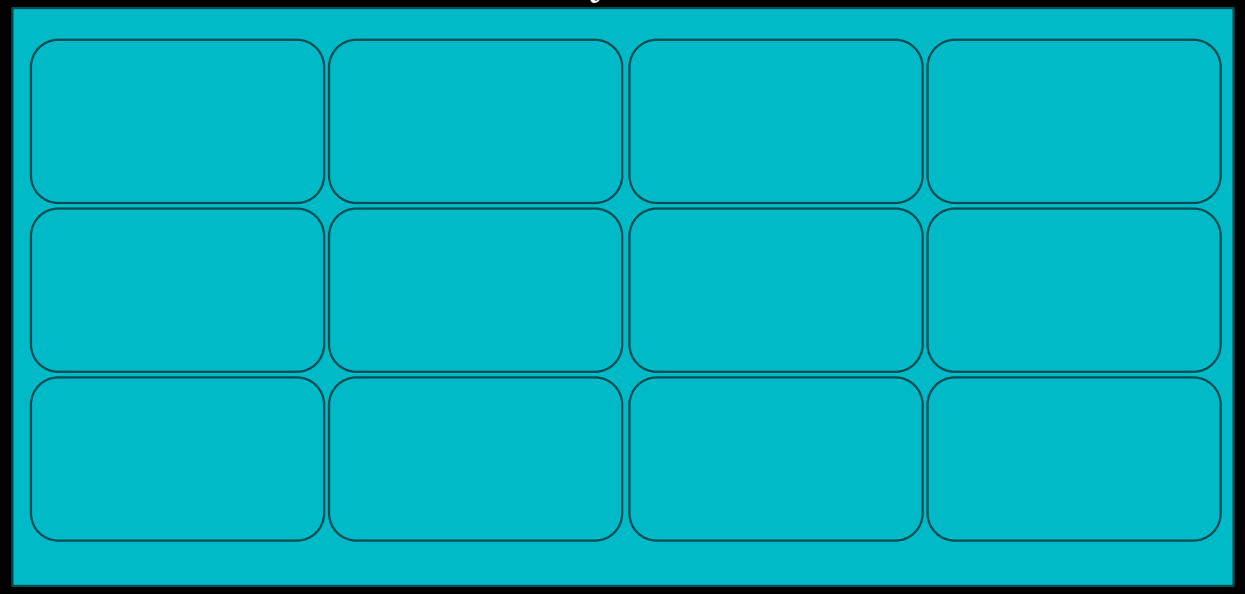
Komponenten bearbeiten

- → Zwei wichtige Layouts
  - → TableLayoutPanel
  - → FlowLayoutPanel

FlowLayoutPanel



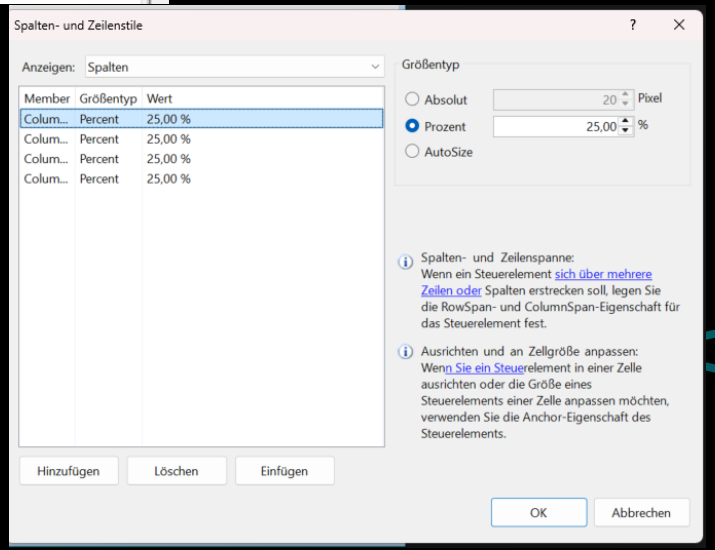
TableLayoutPanel



TableLayoutPanel

→ Man kann die Anzahl der Spalten und Zeilen genau definieren

→ Auch den prozentualen Anteil

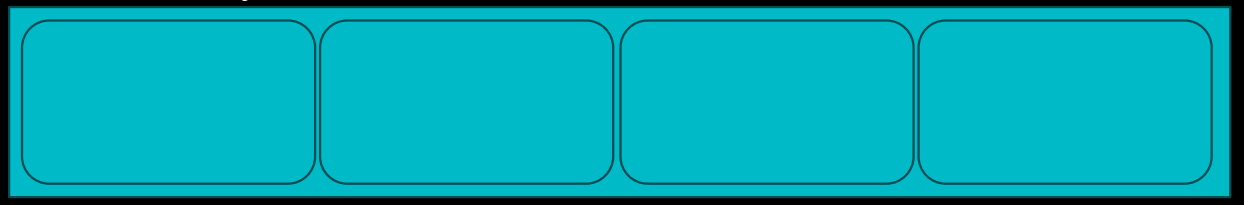


(Name)	**tableLayoutPanel1**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoScroll	False
AutoScrollMargin	0; 0
AutoScrollMinSize	0; 0
AutoSize	False
AutoSizeMode	GrowOnly
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
CausesValidation	True
CellBorderStyle	None
ColumnCount	**4**
Columns	**(Collection)**
ContextMenuStrip	(none)
Cursor	Default
Dock	None
Enabled	True
Font	Segoe UI; 9pt
ForeColor	ControlText
GenerateMember	True
GrowStyle	AddRows
ImeMode	NoControl
Location	**10; 300**
Locked	False
Margin	3; 3; 3; 3
MaximumSize	0; 0
MinimumSize	0; 0
Modifiers	Private
Padding	0; 0; 0; 0
RightToLeft	No
RowCount	**5**
Rows	**(Collection)**
Size	**690; 588**
TabIndex	**0**
TabStop	False
Tag	

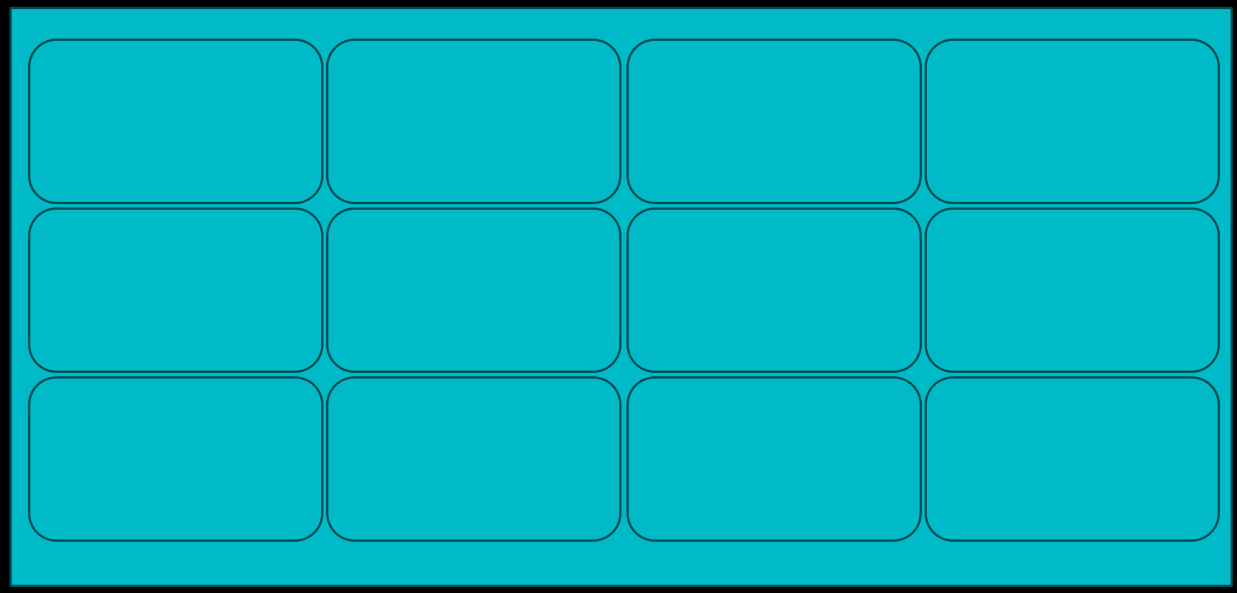
Komponenten in Layouts hinzufügen

- → In die Layouts können mehrere Komponenten hinzugefügt werden
- → Sie sind dann nach dem Layout sortiert
- → Später kann durch das Layout iteriert werden, und jede Komponente angesprochen werden

FlowLayoutPanel



TableLayoutPanel



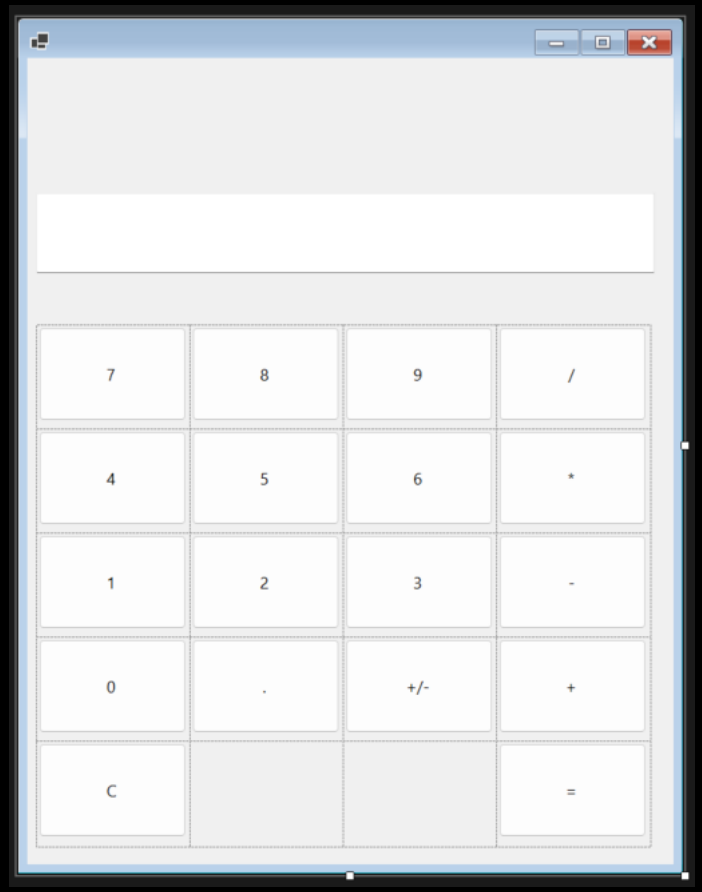
Buttons in Layouts hinzufügen

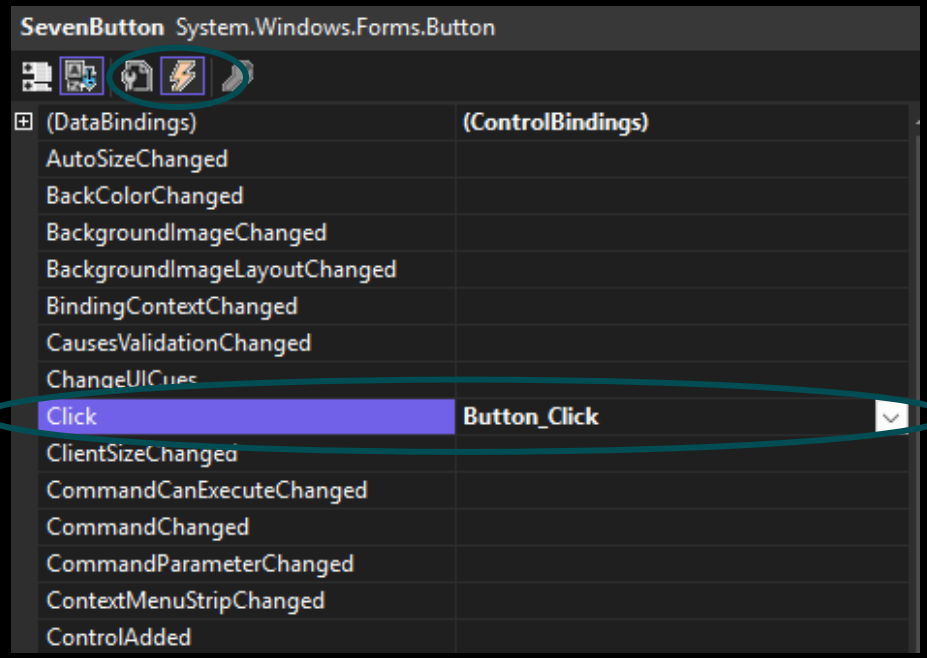
→ Z.B. beim Memory: wurden alle Buttons aufgedeckt?

```
1 Verweis
private bool AllPairsMatched()
{
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Button button = control as Button;
        if (button != null && button.ForeColor == button.BackColor)
            return false;
    }
    return true;
}
```

Mehrere Buttons – gleiche Funktion

→ Soll es nur eine Funktion geben, die alle Buttons/Komponenten erfüllen, kann man in den Eigenschaften die On-Click-Methoden einstellen





Button_Click überprüft den Text des Buttons und führt dementsprechend mathematische Operationen aus. Ist für alle Buttons gleich.

Aufgabe  *ca. 1 – 1,5 Stunden*

→ Programmiert ein Fenster, in dem ihr Buchdaten eingeben könnt (um in einer späteren Aufgabe ein Buch in die Library-Datenbank abzuspeichern)

→ Wie ihr es gestaltet, ist euch überlassen

Mehrere Forms/Panels

Weiteres Fenster hinzufügen

- → Rechtsklick auf Projekt – Hinzufügen – Formular
- → Man kann nun ein weiteres Fenster nach Belieben programmieren
- → Interaktion zwischen den beiden Fenstern?
  - → In der Klasse des Hauptfensters (Form1) ein Objekt des neuen Fensters (Forms 2) erstellen
  - → Mittels .Show() wird das neue Fenster direkt mit dem alten Fenster gezeigt
  - → Mittels .ShowDialog() wird zuerst das neue Fenster und beim Schließen das alte Fenster gezeigt
    - → Oft in Kombination mit DialogResult

Mehrere Panels

- → Man kann auch ein Panel als Komponente in das Hauptfenster ziehen
- → Setzt man es unsichtbar, und schiebt es mit Rechtsklick in den Hintergrund, dann kann ein neues Panel erstellt werden
- → Später im Code können durch PanelName.Visible = true / PanelName.Visible = false die Sichtbarkeit reguliert werden

Aufgabe  *ca. 45 Minuten*

→ Füge ein weiteres Fenster hinzu, das den Nutzer informiert, dass ein Buch erfolgreich hinzugefügt wurde

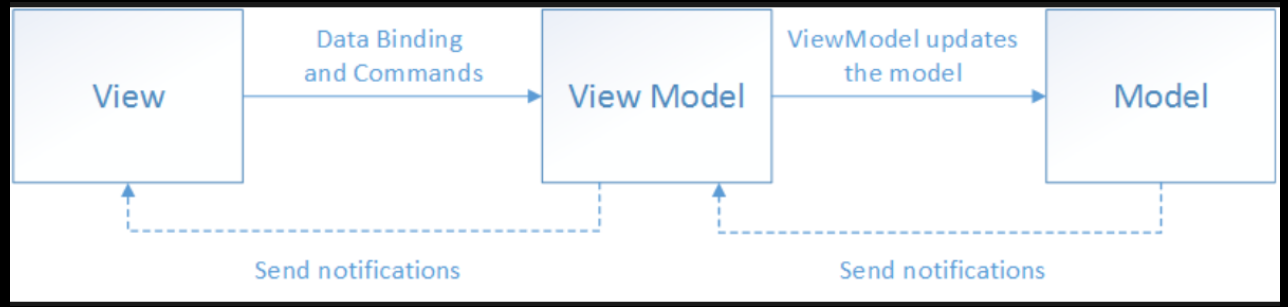
→ Verknüpfe beide Fenster

Ausblick: MVVM-Pattern

MVVM

DataBinding: Änderungen im Model werden automatisch in der UI widergespiegelt

- Ein Entwurfsmuster, das besonders in der Entwicklung mit C# und .NET verwendet wird
- Trennt die Software in drei Komponenten auf:
  - Model
    - Unabhängig von der Benutzeroberfläche
    - Kann bei kleinen Projekten wieder Datenbankabfragen beinhalten
  - View
    - Benutzeroberfläche
    - Bindet sich an das ViewModel, um Daten anzuzeigen
    - Enthält KEINE Logik
  - ViewModel
    - Vermittler zwischen Model und View
    - Enthält Logik für die Darstellung
    - Nutzt Data Bindung für eine lose Kopplung der View



https://learn.microsoft.com/de-de/dotnet/architecture/maui/mvvm

MVVM vs. MVC

- → In C# wird oft von MVVM gesprochen
  - → Beim Nutzen von WPF gibt es einige Mechanismen, die MVVM einfach umsetzen lassen
  - → WinForms ist nicht so mächtig und flexibel wie WPF, daher nicht ideal dafür
- → In Java Swing fehlt eine leistungsstarke Bindungsarchitektur – Änderungen werden manuell durch Listener verarbeitet
  - → Nutzen von MVC-Entwurfsmuster

MVVM vs. MVC

Feature	MVC	MVVM
Funktion	Controller verarbeitet Benutzeraktionen und aktualisiert das Model. D.h. die View fragt aktiv nach neuen Daten. Danach wird die UI manuell aktualisiert. (z.B. Reload notwendig)	View ist mit ViewModel über Data Binding verbunden. ViewModel hält die anzuzeigenden Daten. Das DataBinding sorgt für automatische UI-Updates. Wenn sich das Model ändert, wird die UI aktualisiert.