# Aufgabe

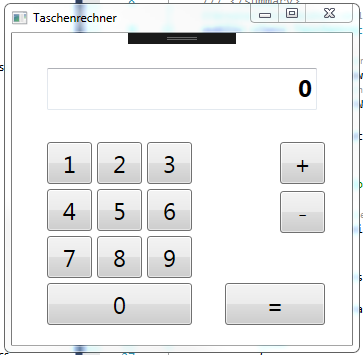
Programmierung eines Taschenrechners in verschiedenen Varianten. Als Kommandozeilen-Programm, mit WPF-GUI, mit TDD (Test driven design), getrennter UI- und Geschäftslogik.

# User Story 1

Aufgabe ist es, ein Kommandozeilenprogramm zu schreiben, dass Additionen durchführen kann. Das Programm soll zwei Parameter entgegennehmen und das Ergebnis (die Summe) ausgeben.

# User Interface

Das User Interface soll in etwa so aussehen:



# Architektur

Das Programm soll in 4 C#-Projekten realisiert werden:

* UI  
  Hauptprogramm, enthält die Controls. Ruft die Methoden in CalculatorUI auf.
* UI Interface Logik  
  Diese Klassenbibliothek soll eine Klasse CalculatorUI mit einer public-Methode „Process\_key\_pressure\_and\_return\_new\_display\_text“ enthalten.  
  Ruft die Methoden in Klasse Calculator auf.
* Business Logik  
  Diese Klassenbibliothek soll eine Klasse Rechner Calculator mit zwei public-Methoden „Add“ und „Subtract“ enthalten. Gerechnet werden soll mit dem Datentyp int.
* Unit tests  
  In dieser Klassenbibliothek sollen alle Unit Tests implementiert werden.

# Vorgehensweise

TDD (test driven design)

* Zunächst sollen alle Unit Tests für die Methoden aus Klasse Calculator erstellt werden.
* Dann die Unit Tests für die Methode aus Klasse CalculatorUI.
* Dann soll der Rest implementiert werden.

Die Klasse CalculatorUI hat folgende public-Methode:   
public string Process\_key\_pressure\_and\_return\_new\_display\_text(string taste)

# Erweiterungen

* Das UI erstellt eine Instanz des Calculators und macht Dependency Injection dieser Instanz in die Klasse CalculatorUI. Hierzu wird für die Klasse Calculator ein Interface ICalculator erstellt.

1. Implementation einer neuen Klasse DecimalCalculator mit Datentyp decimal statt int.
2. Implementation eines Console Application Projekts, dass die beiden Klassen Calculator und CalculatorUI nutzt.

# Data Binding

## Xaml-Code

<TextBox Name="Ergebnis" Text="{Binding Anzeigewert, Mode=TwoWay}"/>

## UI Klasse

## public string Anzeigewert { get; set; }

## UI CodeBehind file must implement INotifyPropertyChanged

public partial class MainWindow : Window, INotifyPropertyChanged

private PropertyChangedEventHandler propertyChanged;

event PropertyChangedEventHandler INotifyPropertyChanged.PropertyChanged

{

add { propertyChanged += value; }

remove { propertyChanged -= value; }

}

public void NotifyPropertyChanged(String info)

{

if (propertyChanged != null)

propertyChanged(this, new PropertyChangedEventArgs(info));

}

## UI Notification

private void Button\_Click(object sender, RoutedEventArgs e)

{

Anzeigewert = . . . ;

NotifyPropertyChanged("Anzeigewert");

}