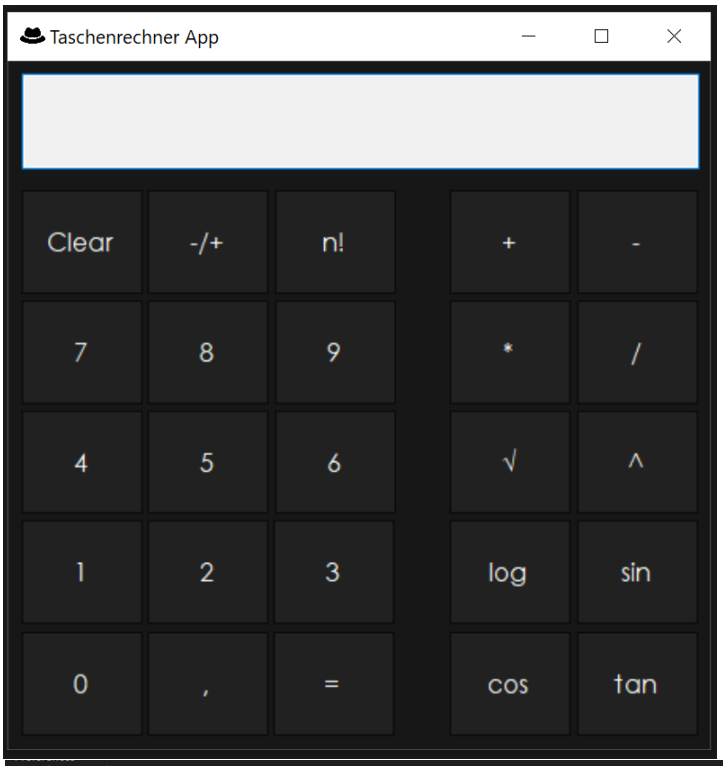
1. Taschenrechner App




```
public partial class Form1 : Form
{
    private Taschenrechner rechner;
    private string eingabe = "";

    1 reference
    public Form1()
    {
        InitializeComponent();
        rechner = new Taschenrechner();
    }

    10 references
    private void buttonZahl_Click(object sender, EventArgs e) // Event-Handler für Zahlen-Buttons
    {
        Button btn = sender as Button;
```

```
public class Taschenrechner
{
    0 references
    public double Addition(double x, double y) => x + y;
    0 references
    public double Subtraktion(double x, double y) => x - y;
    0 references
    public double Multiplikation(double x, double y) => x * y;
    0 references
    public double Division(double x, double y) => x / y;

    1 reference
    public double Sinus(double x) => Math.Sin(x);
    1 reference
    public double Cosinus(double x) => Math.Cos(x);
    1 reference
    public double Tangens(double x) => Math.Tan(x);

    0 references
    public double Potenzieren(double x, double y) => Math.Pow(x, y);
    1 reference
    public double WurzelZiehen(double x) => Math.Sqrt(x);
    1 reference
    public double Logarithmus(double x) => Math.Log(x);
```

Akhmed Osmayev, 1WA

Projektziel

Ziel war es einen Taschenrechner als Windows Forms Applikation zu entwickeln. Er sollte Grundrechenarten und erweiterte mathematische Funktionen beinhalten (sin, cos, tan, log, fakultät).

Vorgehensweise

1. **Anforderungen:**
    - Welche Funktionen der Taschenrechner haben soll.
    - Welche GUI-Elemente benötigt werden (Buttons, TextBoxen).
2. **Grundaufbau:**
    - Neues Windows Forms-Projekt in Visual Studio erstellen mit .NET-Framework.
    - Layout für die Benutzeroberfläche erstellen, also die Größe/Form anpassen.
    - Event-Handler für Zahlen- und Operations-Buttons hinzufügen.

      (Event-Handler sind Methoden, die auf bestimmte Ereignisse reagieren, wie das Klicken eines Buttons oder auswählen von verschiedenen Antworten usw.)
    - Grundrechenarten (Addition, Subtraktion, etc.) programmieren.
3. **Erweiterte Funktionen hinzufügen:**
    - Buttons für spezielle Funktionen wie Sinus, Cosinus, etc. hinzufügen.
    - Logik für diese Funktionen implementieren.
4. **Fehlerbehandlung:**
    - Error Checks einbauen, um ungültige Eingaben und Berechnungsfehler zu behandeln.

      (z.B.: „if“ und „throw“ verwenden für eine Fehler Eingabe)

Fazit

Durch schrittweises Vorgehen und Debugging entstand ein funktionsfähiger Taschenrechner. Klare Anforderungen und genaue Recherche haben dabei geholfen, Probleme effektiv zu lösen.

PS.: form1.designer.cs ist eine automatisch generierte Datei, die form1.cs entspricht. Es hält alle „WinForms“-bezogenen Initialisierungselemente getrennt vom "sauberen" Code form1.cs

Akhmed Osmayev, 1WA

2. Bruchrechner App


```
private void BerechneErgebnis()
{
    try
    {
        if (bruch1 == null || bruch2 == null)
            throw new InvalidOperationException("Beide Brüche müssen definiert sein.");

        Bruch ergebnis;
        switch (operation)
        {
            case "+":
                ergebnis = bruch1 + bruch2;
                break;
            case "-":
                ergebnis = bruch1 - bruch2;
                break;
            case "*":
                ergebnis = bruch1 * bruch2;
                break;
            case "/":
                ergebnis = bruch1 / bruch2;
                break;
            case "^":
                int exponent = int.Parse(textBoxZaehler2.Text); // Exponent aus Zähler
                ergebnis = bruch1.Potenzieren(exponent);
                break;
            default:
                throw new InvalidOperationException("Ungültige Operation.");
        }

        textBoxErgebnisZaehler.Text = ergebnis.Zaehler.ToString();
        textBoxErgebnisNenner.Text = ergebnis.Nenner.ToString();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fehler: " + ex.Message);
    }
}
```

Akhmed Osmayev, 1WA

Projektziel

Hier mussten wir einen Bruchrechner als Windows Forms-Applikation entwickeln. Dieser sollte die Grundrechenarten, Potenzieren und Wurzelziehen mit Brüchen unterstützen.

Vorgehensweise

1. **Anforderungen:**
    - Überlegen, welche mathematischen Rechnungsarten der Rechner unterstützen soll.
    - Bestimmen, wie die GUI aussehen soll, welche Elemente (Buttons, Textboxes etc.)
2. **Grundaufbau:**
    - Ein neues Windows Forms-Projekt in Visual Studio anlegen.
    - Das Layout der Benutzeroberfläche gestalten.
3. **Implementierung der Bruch-Klasse:**
    - Eine separate Klasse Bruch wurde erstellt, um Brüche darzustellen und sie miteinander zu rechnen.
    - Diese Klasse beinhaltet Methoden zur Addition, Subtraktion, Multiplikation, Division, Potenzieren und Wurzelziehen von Brüchen.
4. **Fehlerbehandlung:**
    - Error Checks einbauen, um ungültige Eingaben und Berechnungsfehler zu behandeln.
    - Event-Handler dürfen nicht fehlen, damit die Elemente problemlos funktionieren.

Debugging und Fehlerbehebung

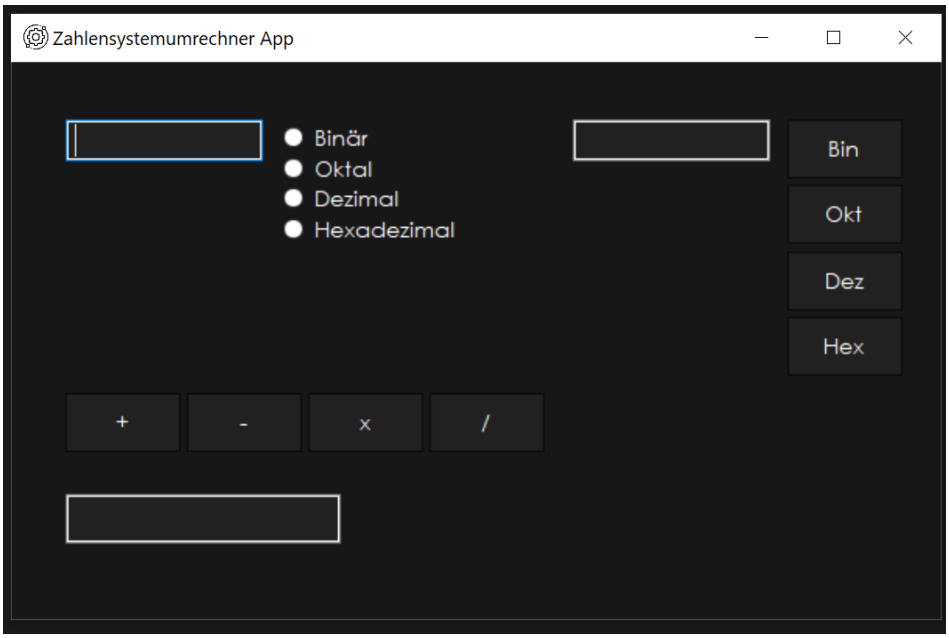
1. **Berechnungsfehler:**
    - Es gab Probleme bei der Potenzierung und dem Wurzelziehen von Brüchen.
    - Lösung: Berechnungen überprüft und die entsprechenden Methoden in der Bruch-Klasse verbessert.
2. **Anzeigeprobleme:**
    - Ergebnisse wurden nicht korrekt in den TextBoxen angezeigt.
    - Lösung: TextBoxen nach jeder Berechnung aktualisieren lassen.

Fazit

Durch das Recherchieren und gezieltes Debugging haben wir einen Bruchrechner erstellt. Die Logik mit der Bruch-Klasse und die Benutzeroberfläche erleichterte die Erstellung und die Fehlerbehebung.

Akhmed Osmayev, 1WA

3. Zahlensystemumrechner App




```
public partial class Form1 : Form
{
    1 reference
    public Form1()
    {
        InitializeComponent();
    }

    1 reference
    private void Form1_Load(object sender, EventArgs e)
    {
    }

    2 references
    private int GetSelectedBase()
    {
        if (radioButton1.Checked) return 2;
        if (radioButton2.Checked) return 8;
        if (radioButton3.Checked) return 10;
        if (radioButton4.Checked) return 16;
        throw new InvalidOperationException("Kein Zahlensystem ausgewählt!");
    }

    1 reference
    private void button1_Click(object sender, EventArgs e)
    {
        PerformOperation((num1, num2) => num1 + num2, "Addition");
    }

    1 reference
    private void button2_Click(object sender, EventArgs e)
    {
        PerformOperation((num1, num2) => num1 - num2, "Subtraktion");
    }

    1 reference
    private void button3_Click(object sender, EventArgs e)
    {
        PerformOperation((num1, num2) => num1 * num2, "Multiplikation");
    }

    1 reference
    private void button4_Click(object sender, EventArgs e)
    {
        PerformOperation((num1, num2) => num1 / num2, "Division");
    }
```

PS.: wenn du z.B. Binär mit Hexadezimal addieren willst, wandle beide zahlen zuerst in Dezimal um, und addiere sie so

Akhmed Osmayev, 1WA

Projektziel

Dieses Mal war das Ziel dieses Projekts die Entwicklung einer Anwendung, die Zahlen zwischen verschiedenen Zahlensystemen (Binär, Oktal, Dezimal, Hexadezimal) umrechnet und grundlegende Rechenoperationen mit diesen Zahlen durchführt.

Vorgehensweise

1. **Anforderungen:**
    - Festlegen, welche Zahlensysteme unterstützt werden (Binär, Oktal, Dezimal, Hexadezimal).
    - Bestimmen, welche mathematischen Rechenwege (Addition, Subtraktion, Multiplikation, Division) erstellt werden.
2. **Aufbau:**
    - Ein neues Windows Forms-Projekt in Visual Studio erstellen.
    - Die Benutzeroberfläche mit TextBoxen, RadioButtons und Buttons gestalten.
3. **NummerSystem-Klasse entwickeln:**
    - Eine separate Klasse NummerSystem wurde erstellt, um die Logik der Zahlensysteme zu implementieren.
    - Diese Klasse enthält Methoden zur Konvertierung zwischen Zahlensystemen und zur Durchführung verschiedener Operationen.
4. **Fehlerbehandlung:**
    - Error Checking von den Benutzereingaben, um sicherzustellen, dass nur gültige Werte ausgegeben werden.
    - Behandeln von Berechnungsfehlern und Anzeigen von Fehlermeldungen.

Debugging und Fehlerbehebung

1. **Berechnungsfehler:**
    - Es gab Schwierigkeiten bei der Umrechnung zwischen Zahlensystemen und bei Operationen.
    - Lösung: Berechnungen überprüft und Methoden in der NummerSystem-Klasse korrigiert.

Fazit

Durch eine klare Strukturierung und gezielte Fehlerbehebung konnten wir eine funktionale Anwendung entwickeln, die sowohl die Umrechnung als auch die Berechnung mit verschiedenen Zahlensystemen ermöglichte.

Neben der technischen Funktionalität wurde auch das Design der Anwendung angepasst, indem dunkle Farben verwendet wurden, um die Benutzeroberfläche ansprechender zu gestalten.

Akhmed Osmayev, 1WA