



WS 2018/19

Skript - Kap. 5

Entwicklungsumgebung Visual Studio .NET und GUI (2. Hörsaalübung)

Prof. Dr.-Ing. Uwe Rüppel
Meiling Shi, M.Sc.

Achtung!

Geänderte Sprechstunde:

- statt 30.11.18 Fr. 12:00 -17:00 Uhr
- 29.11.18 Do. 12:00-17:00 Uhr

Kolloquium: KW 50 und KW 51

VisualStudio: .Net Framework ≥ 4.7

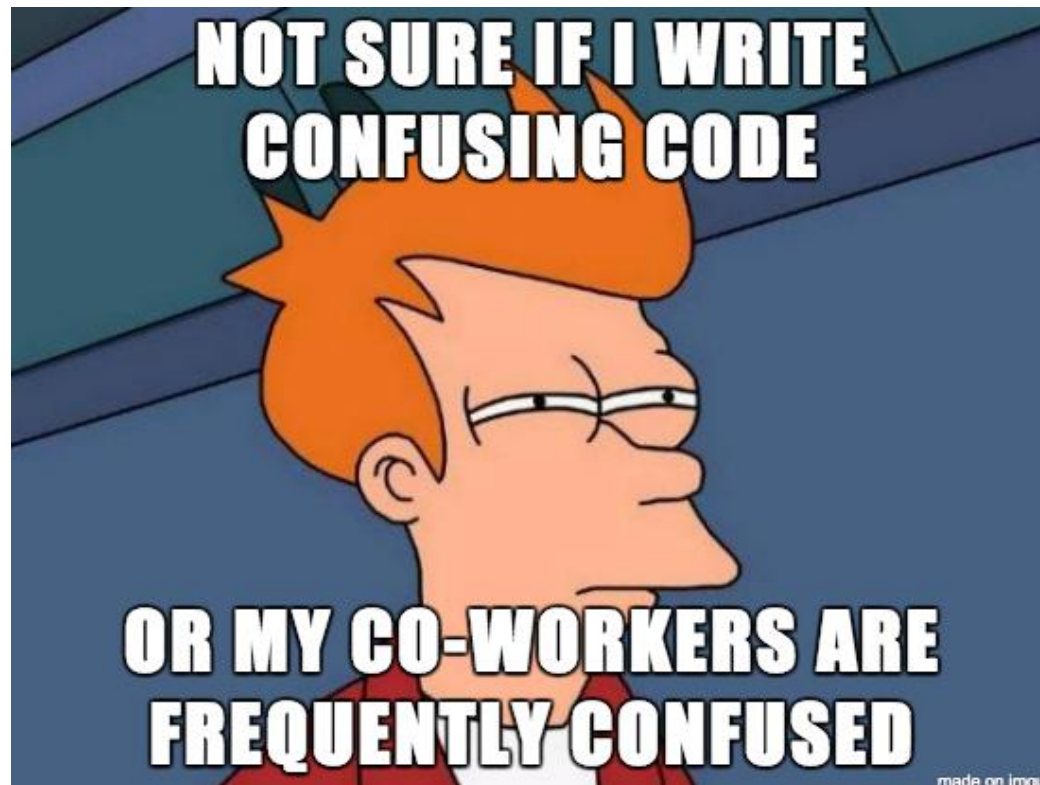
1. HowTo: Clean Code
2. Importieren von Projekten
3. Grafische Benutzeroberflächen mit MS Visual Studio
4. Serialisieren
5. Debugging
6. Demo

1. How To: Clean Code



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Quellcodedokumentation & Programmierprinzipien



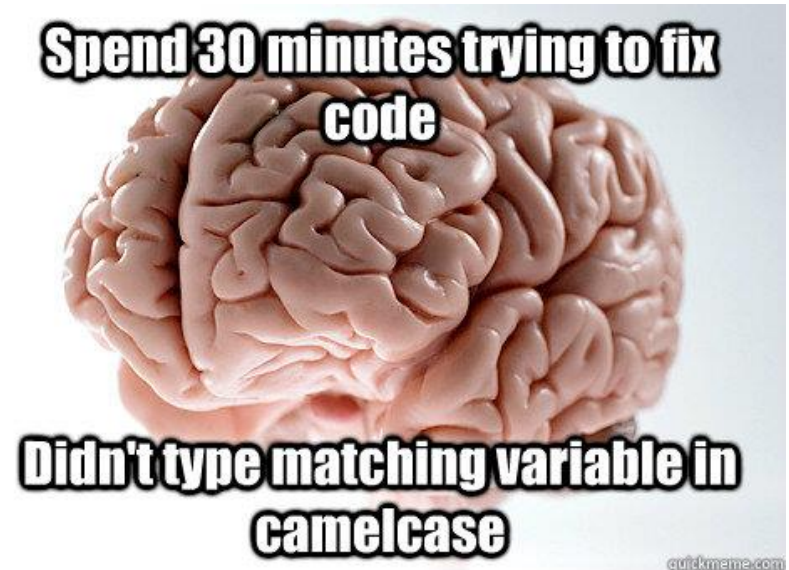
Programmierkonventionen – Notation (1)

- Aussagekräftige Namen für Klassen, Methoden, Attribute und GUI-Elemente
Vorschläge:
 - Formulare: *FormName* → z.B. *FormAnmeldung*
 - Steuerelemente: *SteuerelementnameName* → z.B. *textBoxMatrikelnr* oder *tbMatrikelnr*



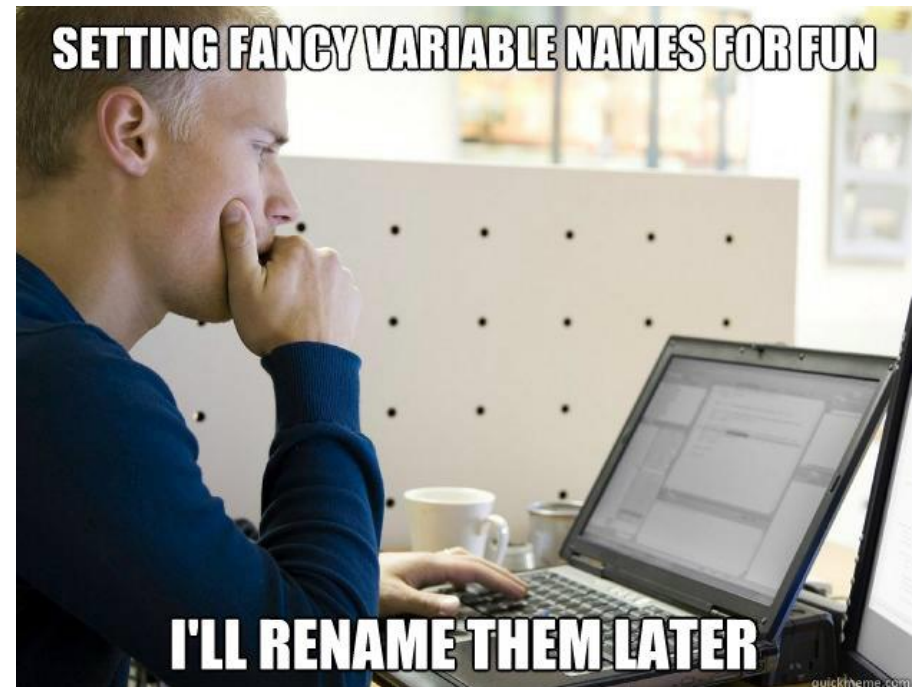
Programmierkonventionen – Notation (2)

- Klassenname: Anfangsbuchstabe groß
- Variablen, Attribute, Methoden:
Anfangsbuchstabe klein
- Zusammengesetzte Worte in CamelCase: `showDialog()`
- Hilfsvariablen: ein Buchstabe z.B. `i, j`
- Gleichmäßiger Aufbau zusammengesetzter Variablen, Attribute, Methoden
- Namen von Beginn an sinnvoll wählen !



Programmierkonventionen – Quellcode

- Eine Anweisung pro Zeile
- Automatische Einrückungen verwenden, in Visual Studio: Strg + K + D
- Max. Zeilenlänge: 80 Zeichen
- Nicht an Kommentaren sparen: z.B. vor jeder Methode ein Kommentar (Was macht sie?)



Kommentieren von Methoden

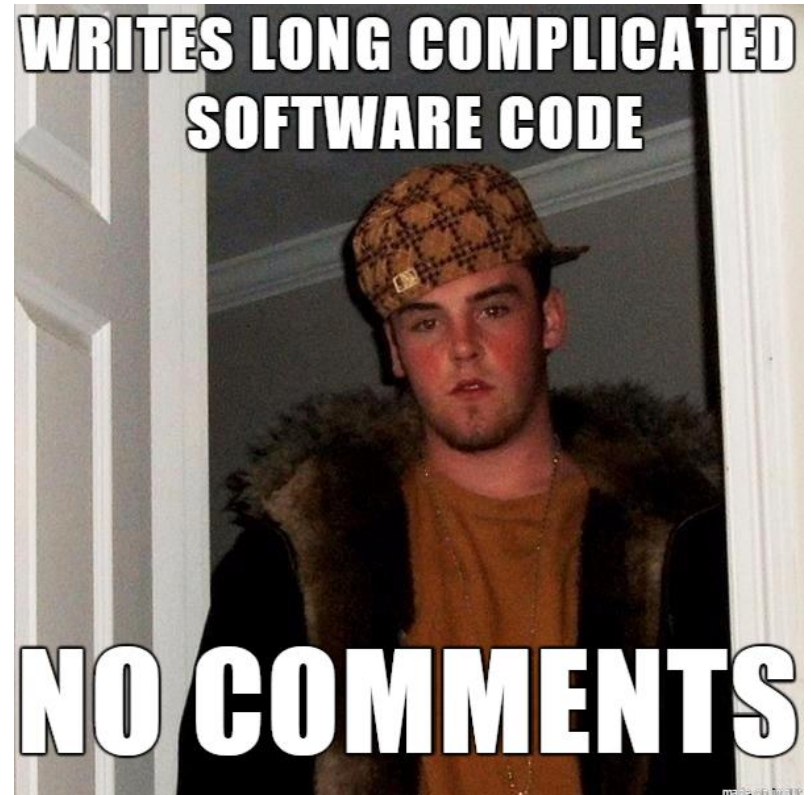
```
/// <summary>
/// Beschreibung der Testklasse
/// </summary>
class Test {

    /// <summary>
    /// Beschreibung der Testklasse
    /// </summary>
    /// <param name=„eingabe“>Beschreibung des
        Eingabeparameters</param>
    /// <returns>Beschreibung des Rückgabewertes
        </returns>

    public int testMethode(double eingabe) {
        double ergebnis = 0.0;
        [...]
        return ergebnis
    }

}
```

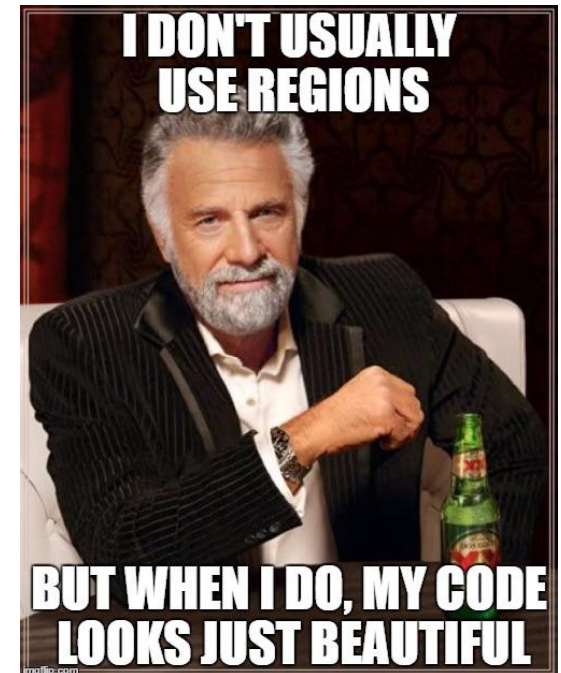
Automatisch erstellt
bei Eingabe von “///”



Hinzufügen von Regions

- Zur Organisation des Quellcodes
- Trennung von Variablen, Konstruktoren, globalen und privaten Methoden
- Regions lassen sich ein- und ausklappen
- Durch die Befehle `#region` und `#endregion`

```
class Test {  
    #region meineGlobalenVariablen  
    private int meinInteger;  
    private string meinString;  
    [...]  
    #endregion  
  
    meineMethoden  
}
```



Prinzipien für sauberen Code (1)

- SRP – Single Responsibility Principle
 - Jede Klasse sollte nur eine Aufgabe erfüllen
- SoC – Separation of Concerns
 - Programm in unterschiedliche Komponenten unterteilen, die jeweils eine Aufgabe haben (z.B. Trennung von Darstellungsebene und Anwendungslogik)
 - Keine Programmlogik in Forms
- YAGNI – You ain't gonna need it
 - Nur das programmieren, was man wirklich braucht

Prinzipien für sauberen Code (2)

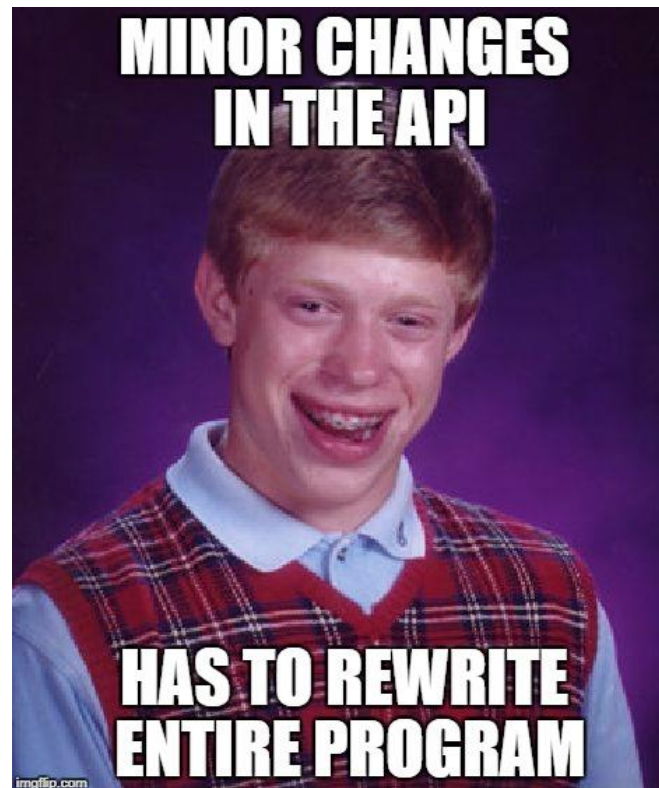
- DRY – Don't repeat yourself
 - Jedes mal wenn Strg + C gedrückt wird, sollten die Alarmglocken läuten
 - Lieber kleine schlanke Methoden, die nur eine Aufgabe haben, als „Spaghetti-Methoden“, die mehrere Dinge erledigen
- KISS – Keep it simple, stupid
 - Einfache Lösungen sind zu bevorzugen



2. Importieren von Projekten



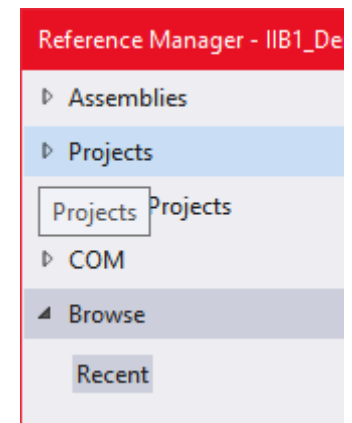
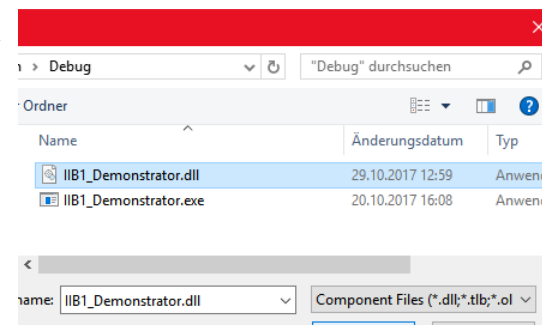
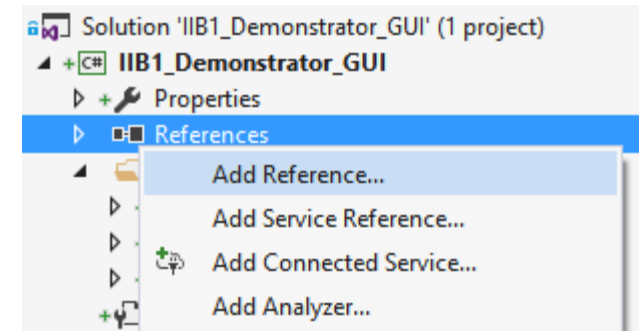
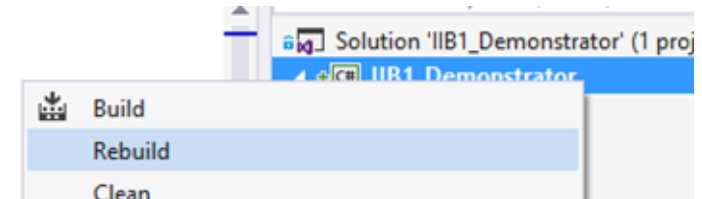
TECHNISCHE
UNIVERSITÄT
DARMSTADT



- Modularer Aufbau des Programms
 - Klassenstruktur könnte auch in anderen Programmen genutzt werden
 - Bei Änderung der angebundenen Software (z.B. Verwendung einer anderen Modellierungssoftware oder eines neuen Autodesk Revit Release) muss nur das für diese Software implementierte Programmstück geändert werden

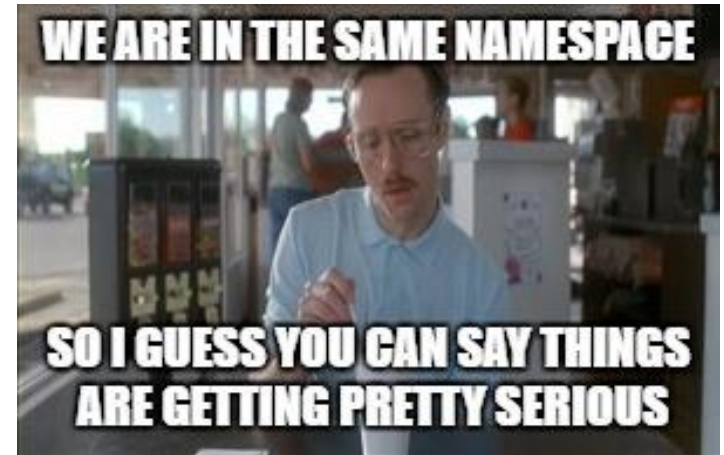
Vorgehen zum Importieren eines Projektes

1. Klassenbibliothek erstellen (Build/Rebuild)
2. GUI Anwendung öffnen
3. Neue Referenz hinzufügen
4. „Projekte“ auswählen
5. „Browse“ → Debug-Ordner der Klassenbibliothek
6. DLL Datei auswählen → „Hinzufügen“
7. Namespace in den Klassen über „using“ Befehl einbinden



C# Namespaces (1)

- Ähnlich zu Java-Packages
- Gruppierung von Klassen
- Strukturierung des Quellcodes und Verhinderung von Namenskonflikten
- Einbindung:
 - `using System.Collections.Generic;`
- Verwendung in Klassenstruktur:
 - `namespace IIBTestNamespace { ... }`



C# Namespaces (2)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

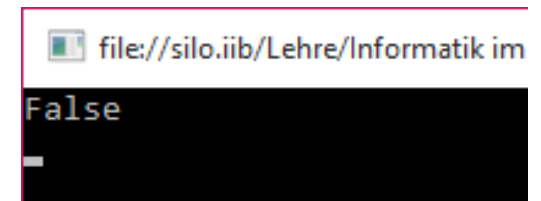
```
using IIBTestNamespace0;

namespace IIBTestNamespace0 {
    class X { }
}

namespace IIBTestNamespace1 {
    class X { }

    class Test {
        static void Main(string[] args) {
            IIBTestNamespace0.X x1 = new IIBTestNamespace0.X();
            X x2 = new X();

            Console.WriteLine(x1 is X);
            Console.Read();
        }
    }
}
```



3 Erstellen eines Graphischen Nutzerinterfaces (GUI)

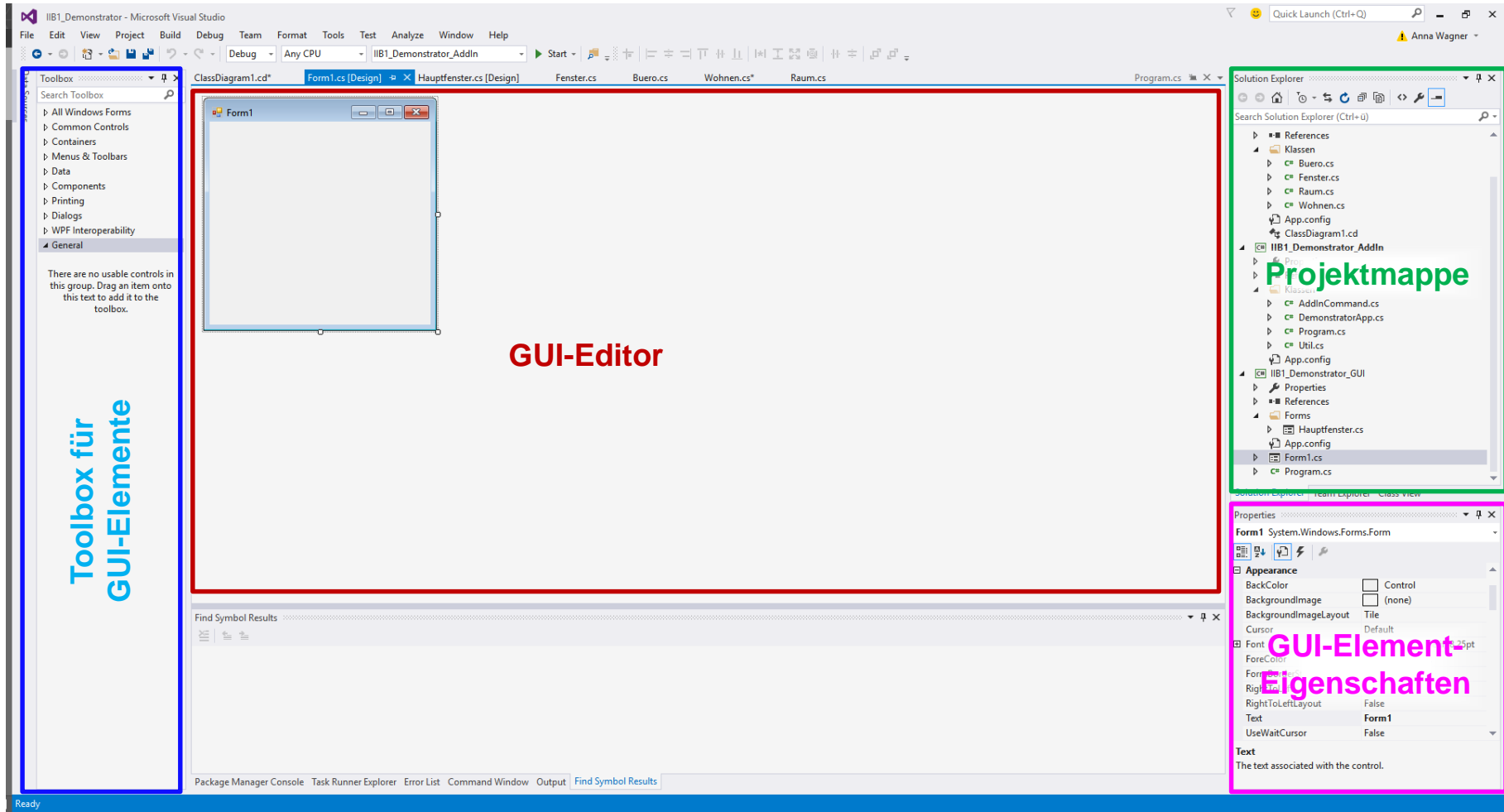


TECHNISCHE
UNIVERSITÄT
DARMSTADT

**A user interface is like a joke.
If you have to explain it,
it's not that good.**

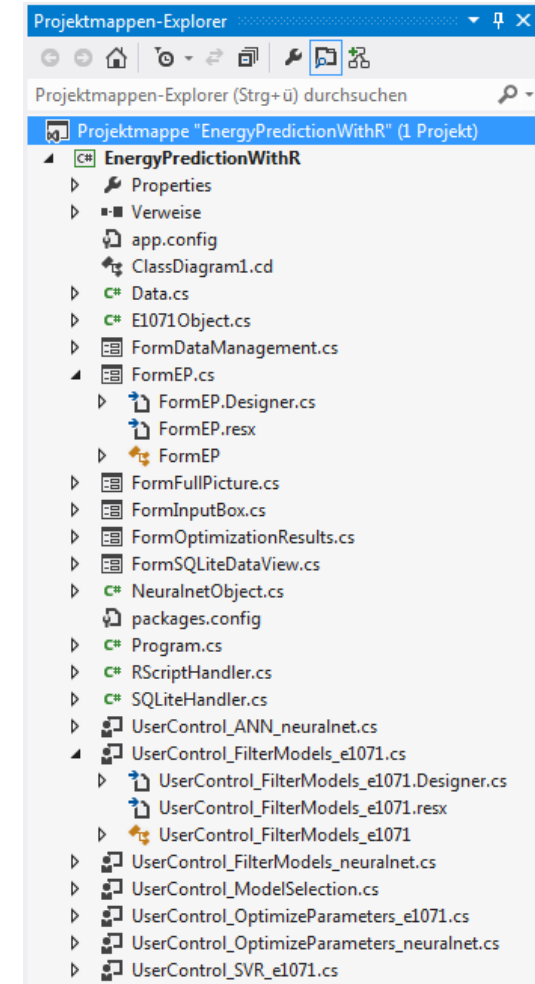
DigitalSynopsis.com

Designansicht



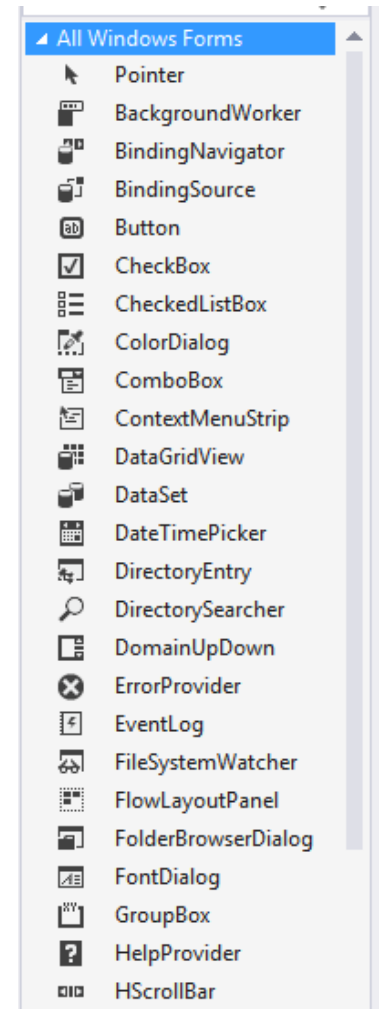
Projektmappen-Explorer

- Enthält Projektdaten
 - Code
 - Konfigurationsdateien
 - Daten
 - UML-(Klassen)Diagramme



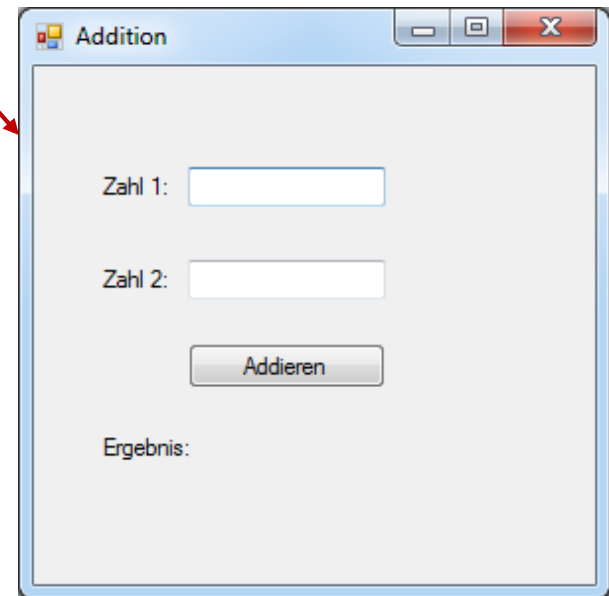
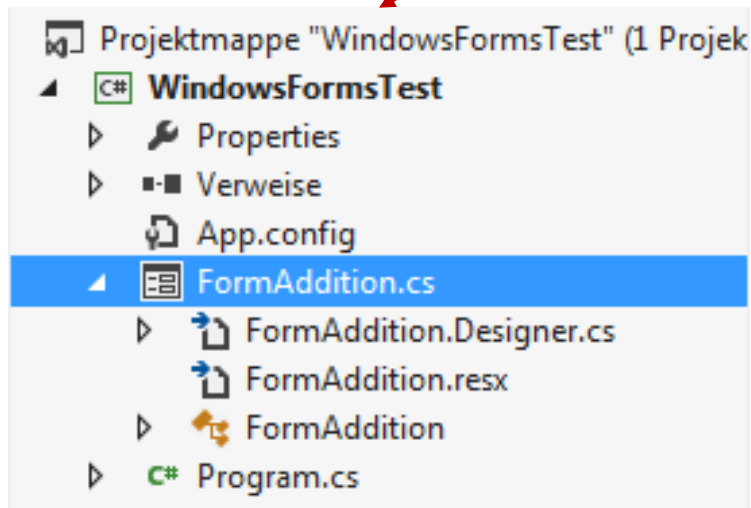
Werkzeugkasten (Toolbox)

- Ansicht → Werkzeugkasten
- Vielzahl visueller Steuerelemente mit unterschiedlicher Komplexität
- Dialoge
- Container/Fenster
- Diagrammelemente (Chart)
- Per Drag & Drop dem Formular hinzufügen



Form (1)

- Besteht aus drei Dateien und einem Fenster:

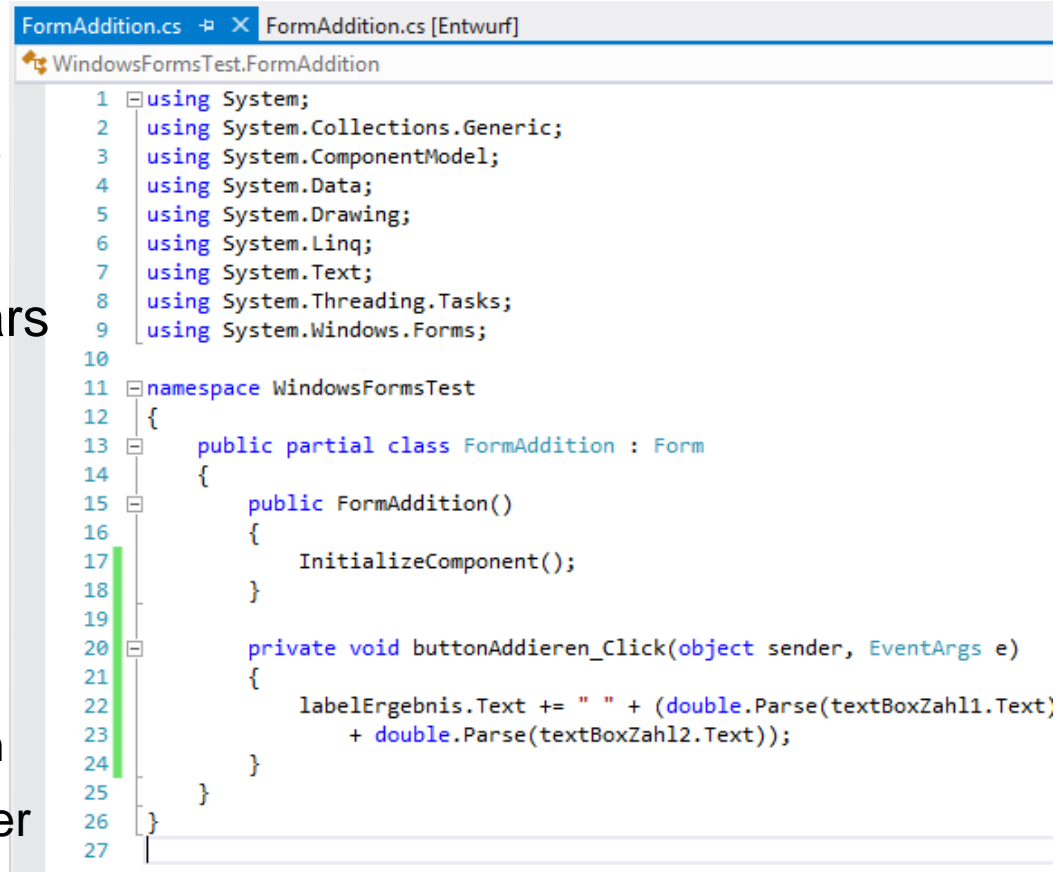


- *.cs[Design]

partielle Klasse, Sicht auf den Graphische Entwurf

Form (2)

- *.CS:
 - partielle Klasse, Sicht auf die Ereignisbehandlung (Programmlogik des Formulars und seiner Steuerelemente)
- C#-Schlüsselwörter blau eingefärbt
- Wechsel von der Entwurfs- in die Quellcodeansicht: F7, oder Rechtsklick → „Quellcode anzeigen“



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace WindowsFormsTest
12 {
13     public partial class FormAddition : Form
14     {
15         public FormAddition()
16         {
17             InitializeComponent();
18         }
19
20         private void buttonAddieren_Click(object sender, EventArgs e)
21         {
22             labelErgebnis.Text += " " + (double.Parse(textBoxZahl1.Text)
23                                     + double.Parse(textBoxZahl2.Text));
24         }
25     }
26 }
27
```


Form (3)



- *.Designer.cs:

partielle Klasse, Sicht auf den generierten Design-Code des Formulars und aller Steuerelemente

- *.resx:

Ressourcen-Datei (XML), meistens nicht relevant für Programmierung



```
private void InitializeComponent()

    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.textBoxZahl1 = new System.Windows.Forms.TextBox();
    this.textBoxZahl2 = new System.Windows.Forms.TextBox();
    this.buttonAddieren = new System.Windows.Forms.Button();
    this.label3 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(37, 50);
    this.label1.Name = "label1";
    this.label1.Size = new System.Drawing.Size(52, 17);
    this.label1.TabIndex = 0;
    this.label1.Text = "Zahl 1:";
    //
    // label2
    //
    this.label2.AutoSize = true;
    this.label2.Location = new System.Drawing.Point(37, 92);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(52, 17);
```

Form (4)



- Jedes Formular erbt von der Klasse Form
- Kann wie jedes Objekt erzeugt werden:

```
FormShowHide f1 = new FormShowHide();
```

- Anzeigen:

```
ShowDialog();
```

```
Show();
```

- Ausblenden:

```
Hide();
```

- Schließen:

```
Close();
```

```
11 namespace WindowsFormsTest
12 {
13     public partial class FormShowHide : Form
14     {
15         public FormShowHide()
16         {
17             InitializeComponent();
18         }
19
20         private void buttonNeuesFormular_Click(object sender, EventArgs e)
21         {
22             FormShowHide f1 = new FormShowHide();
23             f1.ShowDialog();
24         }
25
26         private void buttonHideMe_Click(object sender, EventArgs e)
27         {
28             this.Hide();
29         }
30     }
31 }
```

Show() vs. ShowDialog()

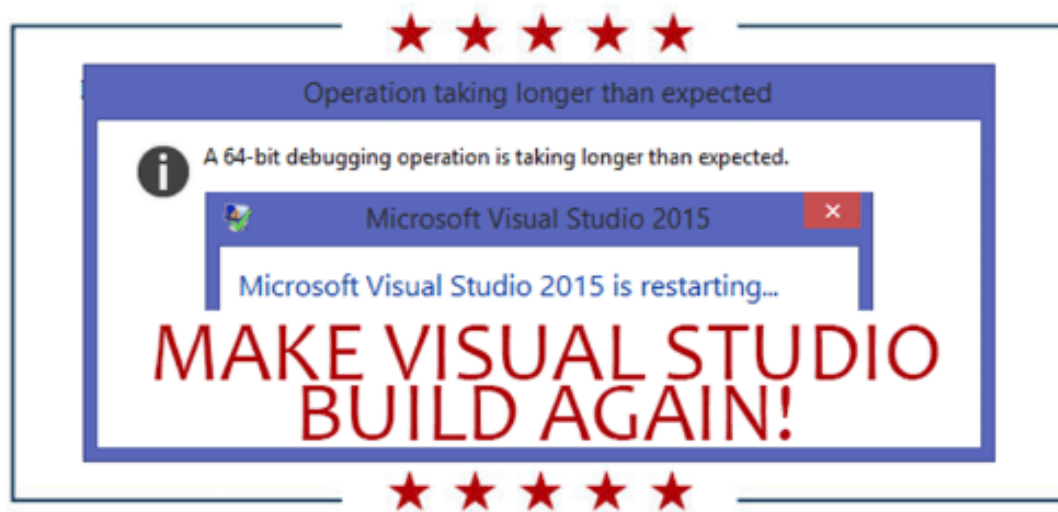
- Formulare / Dialoge sind entweder „*modal*“ oder „*modeless*“
- „modal“ bedeutet, dass das aufgerufene Formular geschlossen oder ausgeblendet werden muss, bevor mit der restlichen Applikation (z.B. dem ursprünglichen Hauptfenster) weitergearbeitet werden kann
- „modeless“ bedeutet, dass beliebig zwischen dem Formular und einem anderen Formular gewechselt werden kann, ohne es vorher schließen zu müssen
- **Show() = modeless**
- **ShowDialog() = modal**

4. Visual Studio und Graphical User Interfaces (GUI)

Implementierung

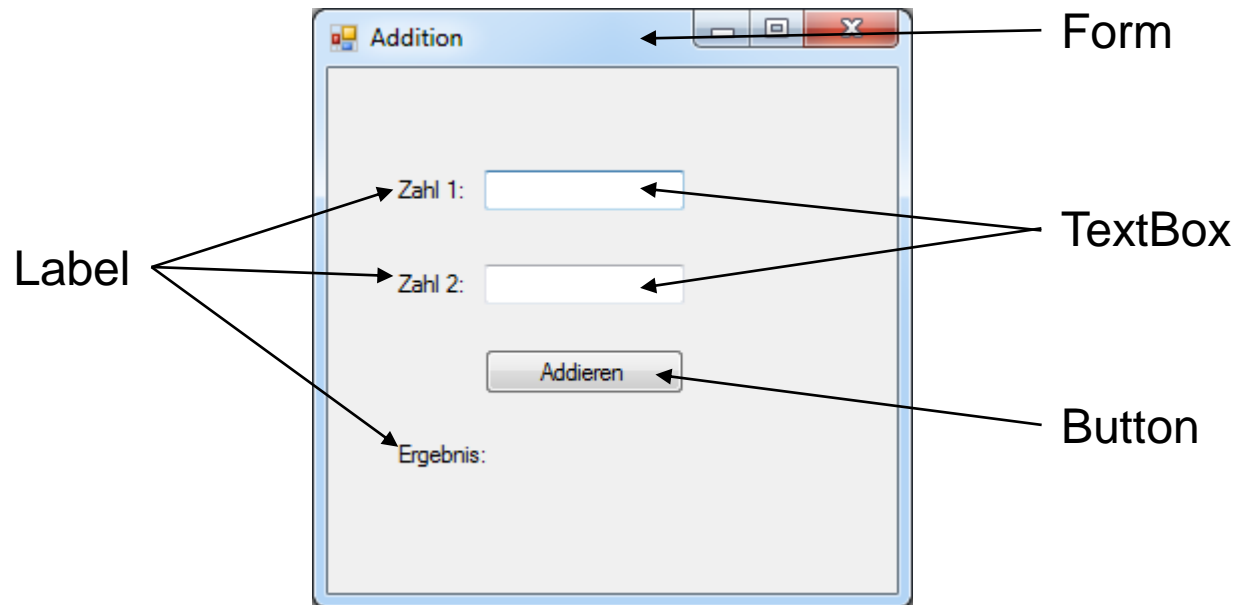


TECHNISCHE
UNIVERSITÄT
DARMSTADT

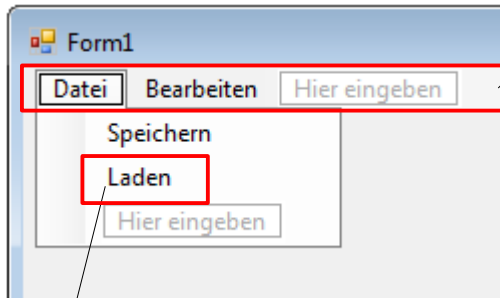


Make Visual Studio Build Again!

4.1 Steuerelemente – (1)

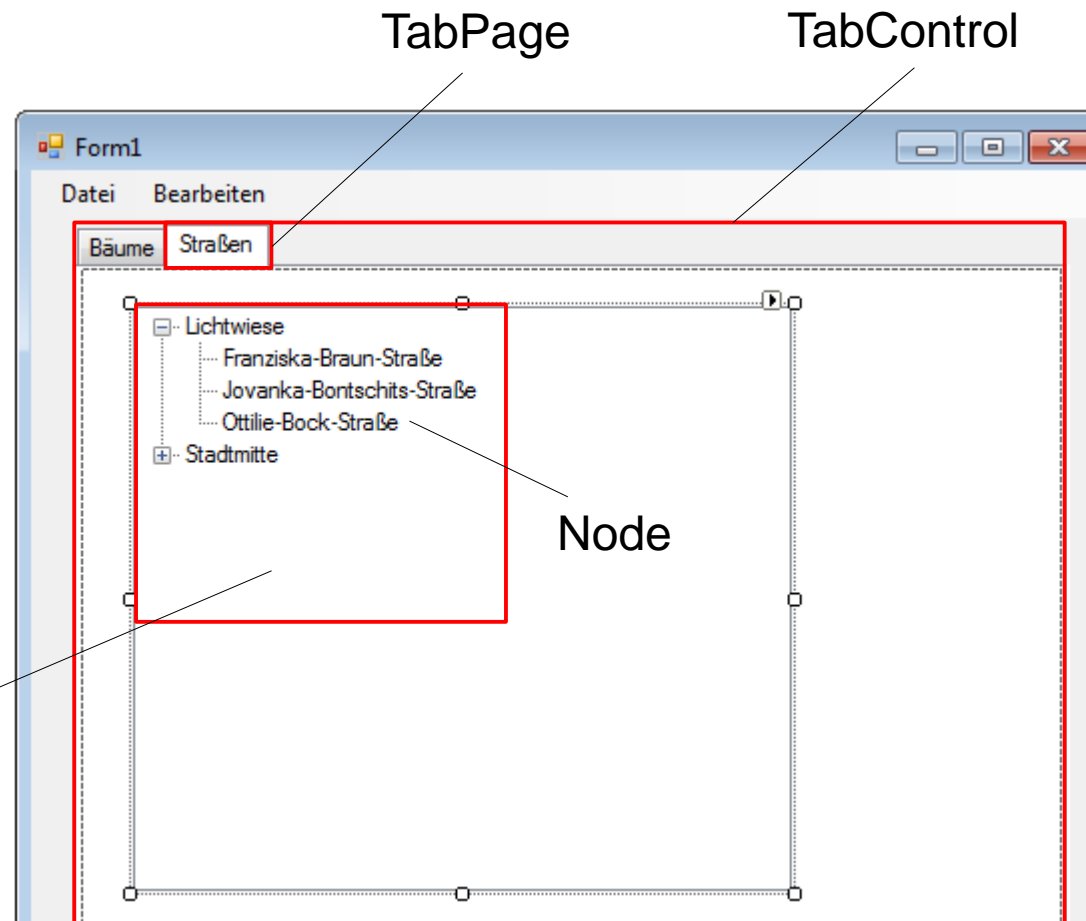


Steuerelemente – (2)



MenuStrip

ToolStripMenuItem
(zum Datei-Öffnen
beispielsweise
OpenFileDialog
hinzufügen)



TabPage

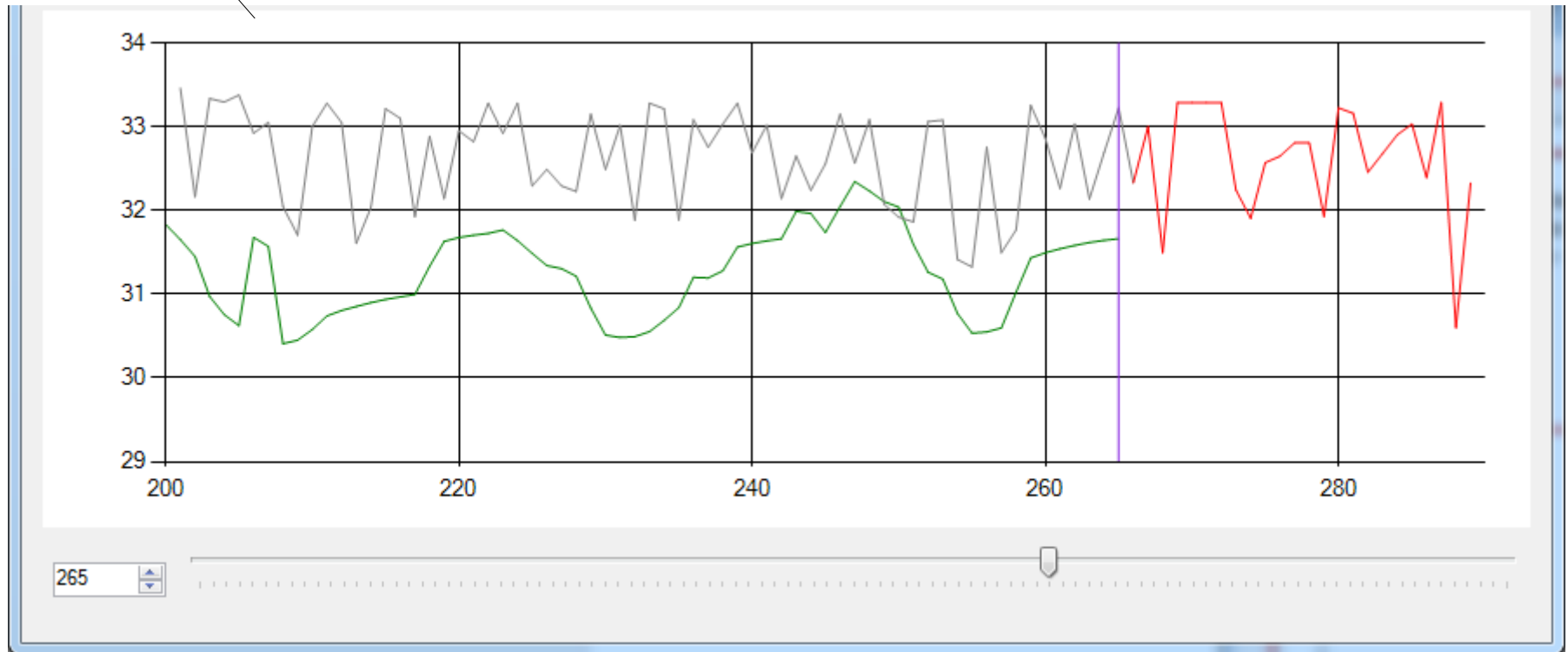
TabControl

Node

TreeView

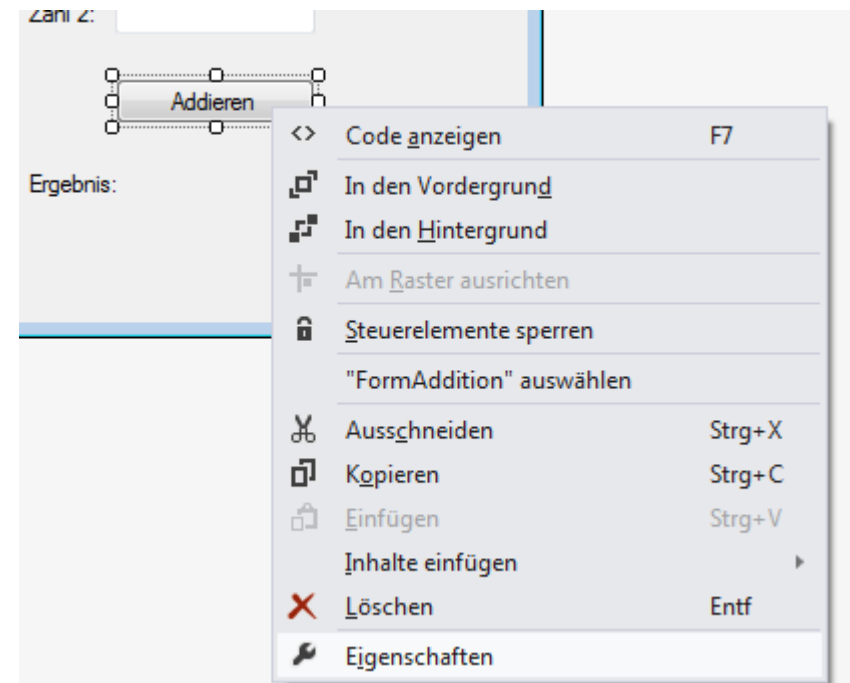
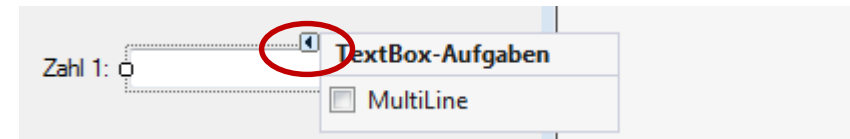
Steuerelemente – (3)

Chart



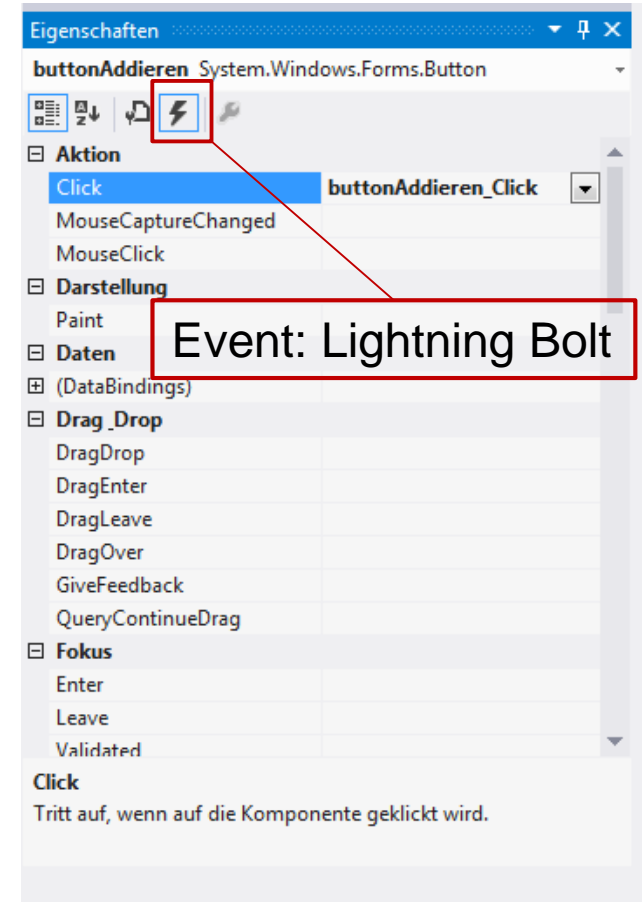
Bearbeiten von Steuerelementen

- Smart Tags:
Ermöglichen den Zugriff auf die „wichtigsten“ Einstellungen von Steuerelementen
- Kontextmenü mit Einstellungen für Controls (Rechte Maustaste)



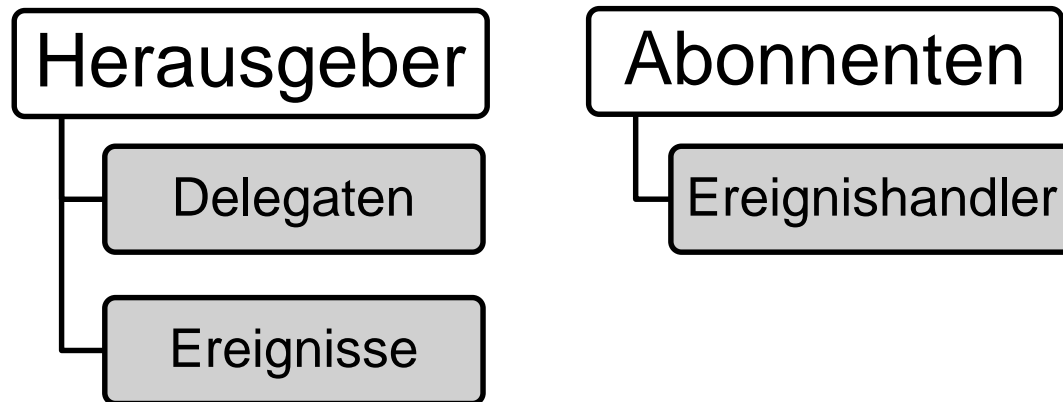
4.2 Events von Steuerelementen

- Event = Ereignis, welches bei einer bestimmten Nutzerinteraktion ausgeführt wird, z.B.:
 - Maus fährt über Element
 - Klick auf Element
 - Scrollen
 - Tastendruck
 - ...
- Verfügbare Ereignisse für ein Steuerelement sind im Eigenschaftsfenster sichtbar



Events (Ereignisse)

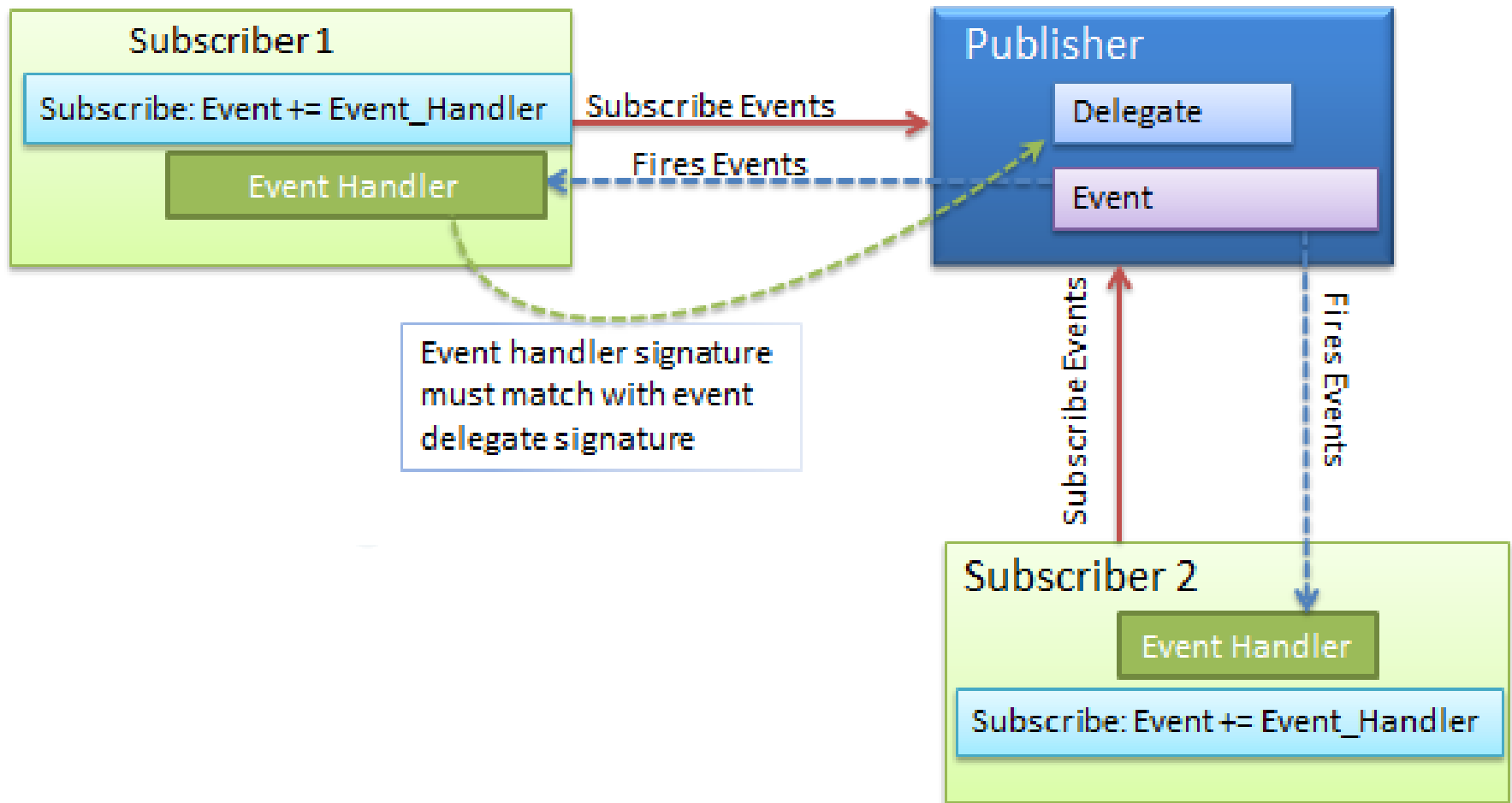
- Wird in GUI intensive verwendet, um Benutzeraktionen wie Mausklicks oder Menüauswahlen zu signalisieren
- Verwendung auch in Benutzerdefinierte Klasse
- Bestandteile:



(Normalerweise sind Herausgeber u. Abonnent zwei unterschiedliche Klassen)

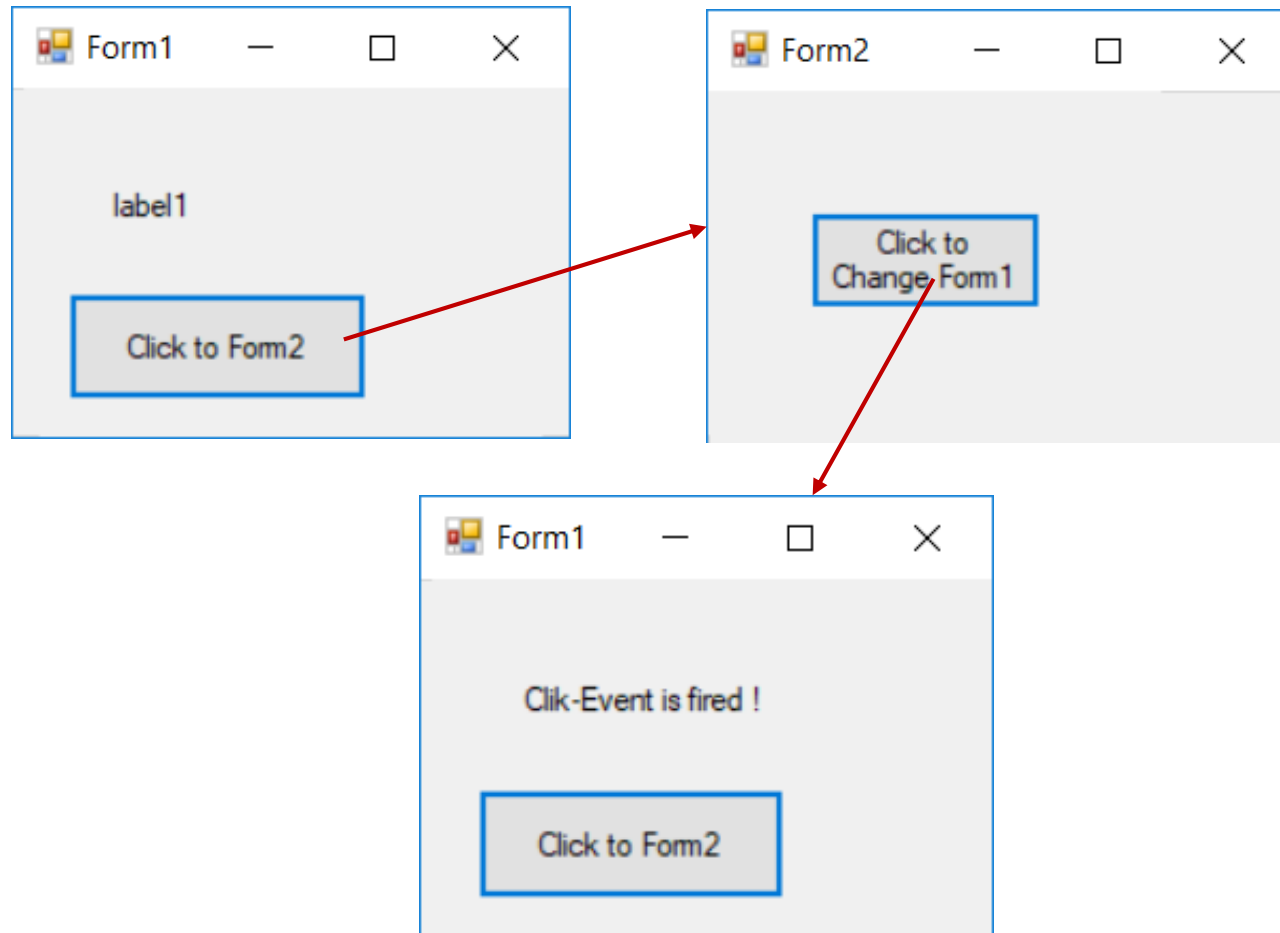
- Herausgeber (Publisher): Deklarieren Ereignisse. Beim Auslösen der Ereignisse werden ihren Abonnenten benachrichtigt.
 - Delegat: Ein Referenztyp für Methode. Er ermöglicht, die Methoden (ein Ereignishandler) als Übergabeparameter weiterzugeben. Delegat verbindet ein Ereignis mit dem Handler durch Definieren einer gemeinsamen Signatur.
 - Ereignis: Methode, die bei einer bestimmten Aktion ausgeführt wird.
- Abonnenten (Subscriber): Abonnieren Ereignisse und bestimmen entsprechende Reaktion auf das Ereignis.
 - Ereignishandler: Methode (hat gleiche Signatur wie Delegat) die durch Delegaten aufgerufen werden. Es bestimmt die Reaktion wenn Ereignis ausgeführt ist.

Events - Aufbau



Quelle: <http://www.tutorialsteacher.com/csharp/csharp-event>

Events- Beispiel (1)



Events- Beispiel (2)



```
namespace WindowsFormsApp3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Form2 f2 = new Form2();
            // Abonieren "Form2.changeForm1Clicked" Ereignis mit Handler
            "OnF2ButtonClicked"
            f2.changeForm1Clicked += new EventHandler(OnF2ButtonClicked);
            f2.Show();
        }
        // Bestimmen Ereignishandler mit gleicher Signatur wie Delegat
        private void OnF2ButtonClicked(object sender, EventArgs e)
        {
            this.label1.Text = "Clik-Event is fired !";
        }
    }
}
```


Events- Beispiel (3)



```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
    }

    // EventHandler: Ein Delegate-Typ in C# mit Signatur: void (object sender, EventArgs e)
    // sender: Die Quelle des Ereignisses.
    // e: Ein Objekt, das keine Ereignisdaten enthält.
    // Vererbung: Objekt - Delegat - EventHandler
    // Deklariert ein Ereignis, seine Abonnenten gleiche Signatur wie EventHandler haben.
    public event EventHandler changeForm1Clicked;

    private void buttonChangeForm1_Click(object sender, EventArgs e)
    {
        if (changeForm1Clicked != null)
        {
            // Gleiche Signatur wie Delegat EventHandler
            changeForm1Clicked(this, e);
        }
    }
}
```

- Ein Ereignis kann mehrere Abonnenten haben.
- Ein Abonnent kann mehrere Ereignisse von mehreren Herausgebern behandeln.
- Ereignisse, die keine Abonnenten haben, werden nie ausgelöst.
- In der .NET Framework -Klassenbibliothek basieren die Ereignisse auf dem EventHandler -Delegaten und der EventArgs -Basisklasse.
- In WindowsForm sind alle Schritte bis zu den Inhalte vom Ereignishandler automatisch erstellt -> Minimal Aufwand für Entwickler!

Tutorial

<https://www.youtube.com/watch?v=jQgwEsJISy0&list=WL&index=2&t=1157s>

Löschen eines Events

NICHT die Methode in der Form-Klasse löschen

→ Wird aus dem Designer aufgerufen

Richtig:

- Eigenschaften des Objekts (label1) aufrufen
- Auf Eventliste wechseln
- Inhalt des ungewollten Events löschen

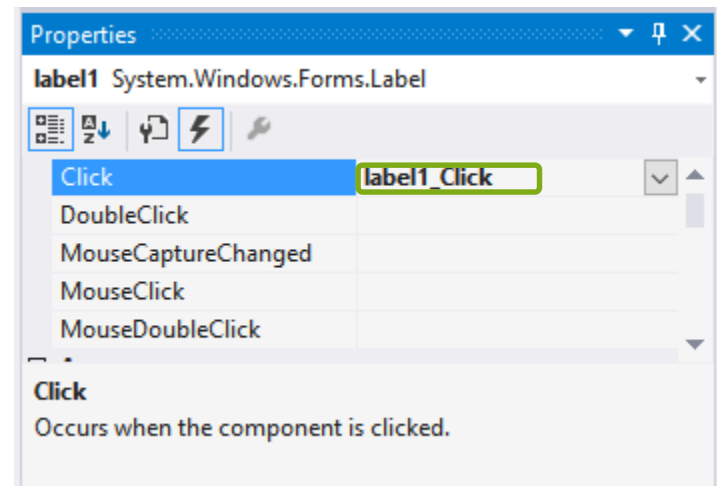
In der Form Klasse

```
private void label1_Click(object sender, EventArgs e)
{
    |
}
```

Designer-Ansicht

```
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(12, 9);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(143, 13);
this.label1.TabIndex = 2;
this.label1.Text = "Bitte wählen Sie einen Raum";
this.label1.Click += new System.EventHandler(this label1_Click);
```

Eigenschaften (Eventliste) des Objekts



4.3 Füllen von Steuerelementen

- In Methode extrahieren (Steuerelemente müssen während der Laufzeit häufiger aktualisiert werden)
- TextBoxen: `textBox.Text`
- ComboBoxen: `comboBox.DataSource`
`comboBox.DisplayMember` } Inhalte eingeben
`comboBox.SelectedItem` → Gewählter Inhalt auslesen
- ListBoxen: `listBox.DataSource`
`listBox.DisplayMember` } Inhalte eingeben
`listBox.SelectedItem` → Gewählter Inhalt auslesen

Füllen von Steuerelementen (2)

- TreeView

`treeView.SelectedNode`

`treeView.Nodes.Add()`

`treeView.Nodes.Clear()`

`treeView.Refresh()`

→ Gewählter Inhalt
auslesen

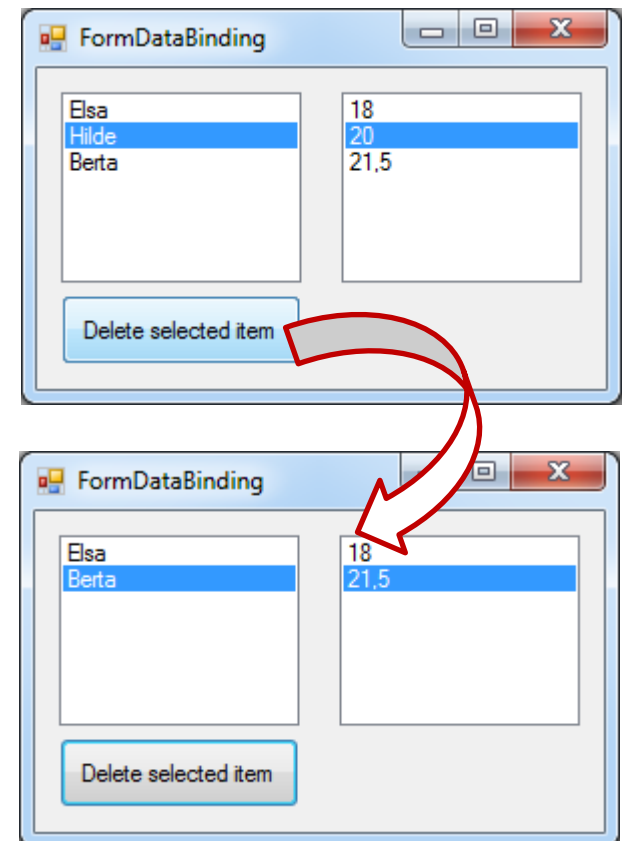
} Inhalte verwalten

Bsp. ListBox: GUI-Elemente mit Datenbindung

- Die Klasse `BindingList<T>` ermöglicht eine bidirektionale Datenbindung zu beispielsweise einer ComboBox / ListBox

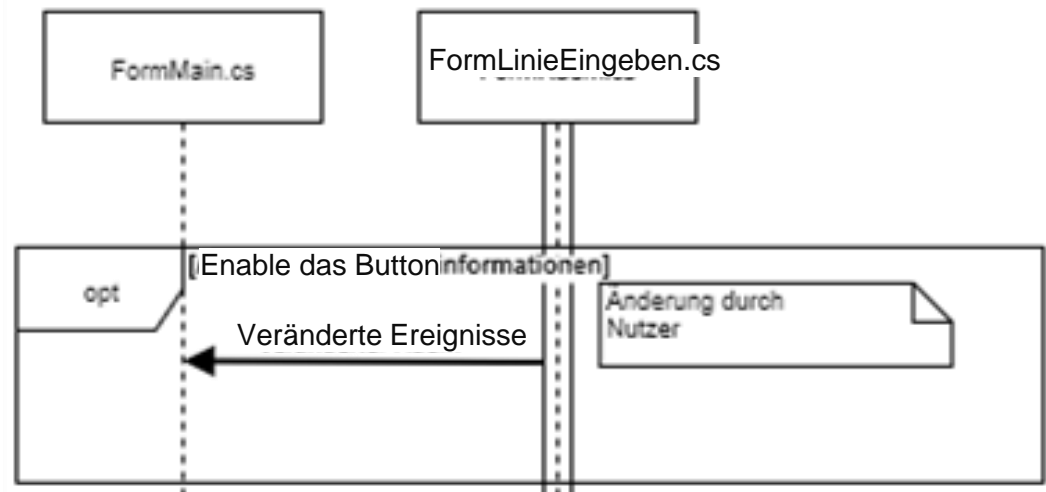
```
public partial class FormDataBinding : Form{
    BindingList<Kuh> listOfKuehe;
    public FormDataBinding(){
        InitializeComponent();
        initializeListOfKuehe();
        listBox1.DataSource = listOfKuehe;
        listBox1.DisplayMember = "Name";
        listBox2.DataSource = listOfKuehe;
        listBox2.DisplayMember = "Milchleistung";
    }
    void initializeListOfKuehe(){
        listOfKuehe = new BindingList<Kuh>();
        listOfKuehe.AllowNew = true;
        listOfKuehe.AllowEdit = false;
        listOfKuehe.AllowRemove = true;
        listOfKuehe.RaiseListChangedEvents = true;
        listOfKuehe.Add(new Kuh("Elsa", 18, Color.Brown));
        listOfKuehe.Add(new Kuh("Hilde", 20, Color.White));
        listOfKuehe.Add(new Kuh("Berta", 21.5, Color.Chocolate));
    }

    private void buttonDeleteItem_Click(object sender, EventArgs e){
        listOfKuehe.RemoveAt(listBox1.SelectedIndex);
    }
}
```



4.4 Interaktion zwischen Forms

- Zugriff auf Daten und Steuerelemente (mit ihren Eigenschaften)
- Notwendig, um konsistenten Datenfluss zu ermöglichen
- Gemeinsame Datengrundlage (gleiche Objektklassen)



Beobachtungsauftrag:
Wie könnte die Datenübergabe
eines veränderten Raumes durch
die vorgestellten Methoden
realisiert werden?

Zugriff auf Controls aus einer anderen Form

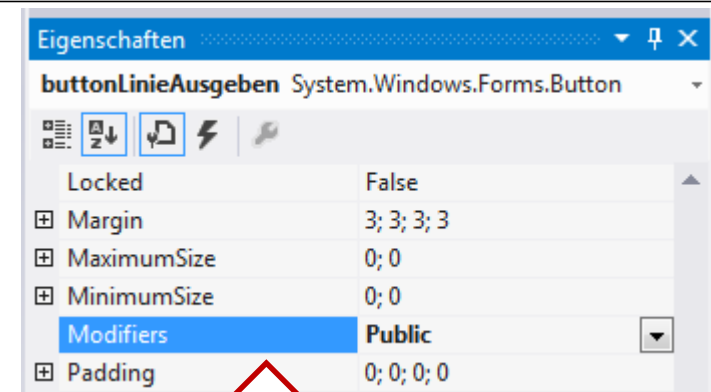
(1) Properties

Mit einer Property:

```
public partial class FormStart : Form
{
    //...
    public Boolean ButtonLinieAusgebenEnabled
    {
        get { return buttonLinieAusgeben.Enabled; }
        set { buttonLinieAusgeben.Enabled = value; }
    }
    //...
    FormLinieEingeben.Show(this)
    //...
}
```

Superklasse als
Übergabeparameter

```
public partial class FormLinieEingeben : Form
{
    //...
    private void buttonAbschicken_Click(object sender, EventArgs e)
    {
        FormStart fStart = (FormStart)this.Owner;
        fStart.ButtonLinieAusgebenEnabled = true;
        //...
    }
}
```



Geht zwar, ist aber unflexibel
Besser: Properties / Events
verwenden

Nur die Eigenschaften sichtbar/zugreifbar
machen, die auch benötigt werden.

Zugriff auf Controls aus anderem Form (2)

Eventbasiert

Mit einem Event:

FormLinieEingeben

Erstellen eines Events

EventHandler hinzufügen
zum Button-Event
(statt durch Klicken
im Form Fenster)

Bei Button-Event: „feuern“
unseres neuen Events

```
public partial class FormLinieEingeben : Form
{
    public event EventHandler FLEButtonClicked;

    public FormLinieEingeben(){
        InitializeComponent();
        buttonAbschicken.Click += new EventHandler(OnButtonClicked);
    }

    private void OnButtonClicked(object sender, EventArgs e){
        if (FLEButtonClicked != null)
            FLEButtonClicked(this, e);
        //...
    }
}
```

FormStart

EventHandler für das
Event zuweisen

Event behandeln

```
public partial class FormStart: Form
{
    //...
    fEingeben = new FormLinieEingeben();
    fEingeben.FLEButtonClicked += new EventHandler(OnFLEButtonClick);

    //...
    private void OnFLEButtonClick(object sender, EventArgs e)
    {
        buttonLinieAusgeben.Enabled = true;
    }
}
```

Zugriff auf Objekt aus anderem Form

1. Variante: Übergabe der Daten selbst in Form von Objekten/Objekt-

Referenzen

```
//Klasse FormStart
fInfos = new FormLinienInfos(linie);

//Klasse FormLinienInfos
public FormLinienInfos(Linie2D linie){ /*...*/ }
```

2. Variante: Referenzieren eines Objekts, welches Zugriff auf die Daten bietet (z.B. durch Properties).

```
//Klasse FormStart
private Linie2D linie;
public Linie2D Linie{ set { linie = value; } }
//...
fEingeben = new FormLinieEingeben();
fEingeben.Owner = this;

//Klasse FormLinieEingeben
FormStart fStart = (FormStart)this.Owner;
fStart.Linie = new Linie2D(new Punkt2D(x1, y1), new Punkt2D(x2, y2));
```

- Wer darf Daten wie ändern?
- Geht es um den Transfer oder das Teilen von Daten?

- Einfaches Nachrichtenfenster
- Verschiedene Formen (z.B. Buttons)
- Bsp:

```
string message = "You did not enter a server name. Cancel this  
operation?";
```

```
string caption = "Error Detected in Input";
```

```
MessageBoxButtons buttons = MessageBoxButtons.YesNo;
```

```
DialogResult result;
```

```
// Displays the MessageBox.
```

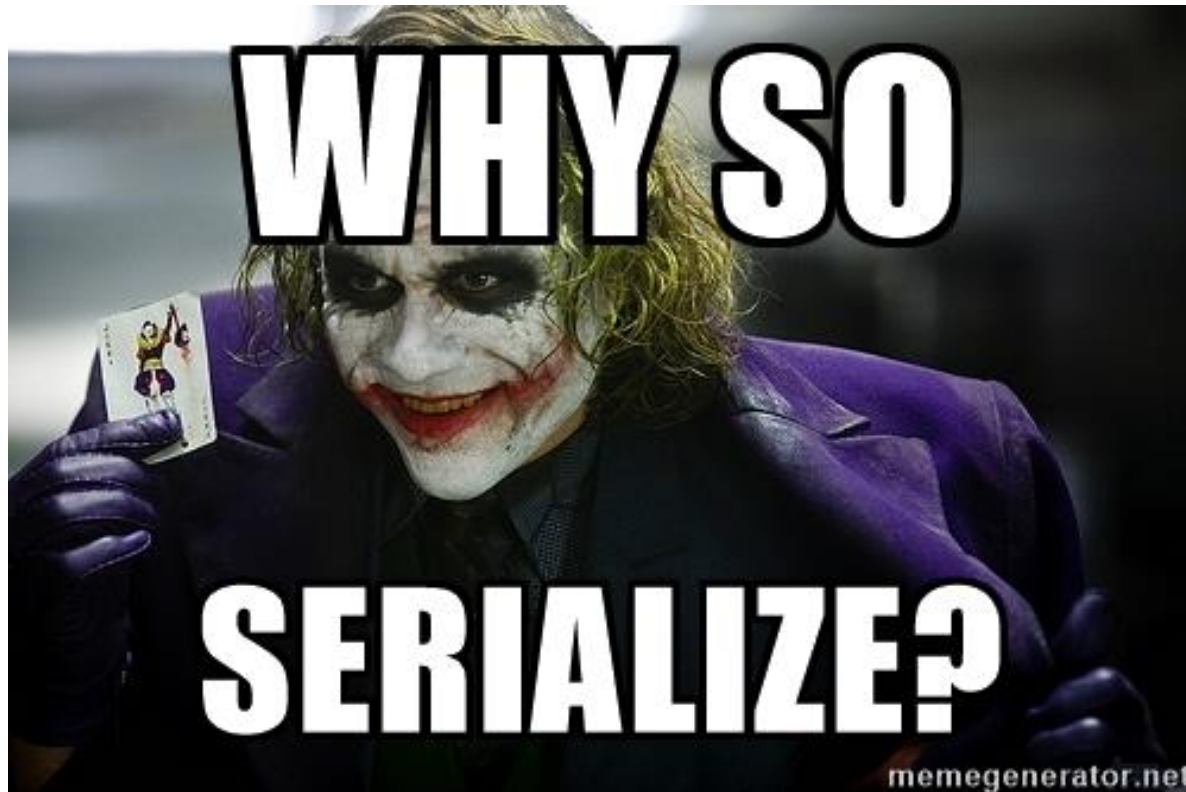
```
result = MessageBox.Show(message, caption, buttons);
```

Wenn keine Buttons
angegeben werden, gibt es
nur einen „OK“ Button, der
die MessageBox schließt

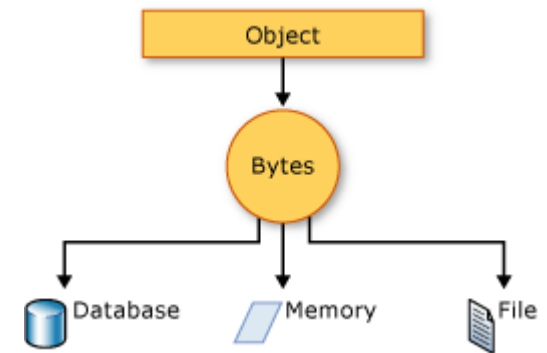
4. Serialisieren



TECHNISCHE
UNIVERSITÄT
DARMSTADT



- Serialisierung: Instanzen im Hauptspeicher werden in einen Datenstrom geschrieben und z.B. in einer Datei auf der Festplatte gespeichert
- Basic Serialization (Binary):
 - Klasse muss durch das `SerializableAttribute` gekennzeichnet sein → `[Serializable]`
 - Einzelne Felder können durch `[NonSerializable]` von der Serialisierung ausgeschlossen werden
 - Schreiben in eine / Lesen aus einer Datei erfordert einen `FileStream`
 - Serialisierung im Binärformat wird durch einen `BinaryFormatter` erreicht



Siehe auch:

- XML Serialization
- SOAP Serialization
- Custom Serialization

Objekt in Datei Speichern (Serialisierung)

```
public void speichern(string dateipfad, MyObject obj)
{
    FileStream fs = new FileStream(dateipfad, FileMode.Create);
    BinaryFormatter bf = new BinaryFormatter();
    bf.Serialize(fs, obj);
    fs.Close();
}
```

Streams schließen! Sonst
blockieren sie spätere
Zugriffe auf die Datei.

Hier wird das Objekt
übergeben, das
gespeichert werden soll.

Erzeugt eine neue Datei,
oder überschreibt sie
wenn sie bereits existiert.

Alternativ:
`FileMode.CreateNew`
Erzeugt eine neue Datei
oder löst eine
Fehlermeldung aus, falls
sie bereits existiert.

Objekt aus Datei lesen (Deserialisierung)

```
public MyObject laden(string dateipfad)
{
    FileStream fs = new FileStream(dateipfad, FileMode.Open);
    BinaryFormatter bf = new BinaryFormatter();
    MyObject obj = (MyObject) bf.Deserialize(fs);
    fs.Close();
    return obj;
}
```

↑
Casting!

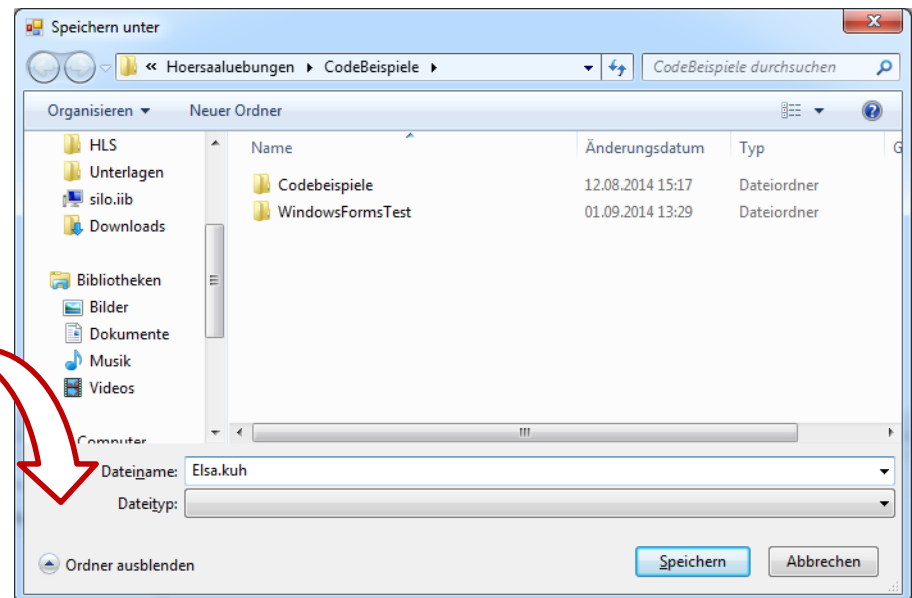
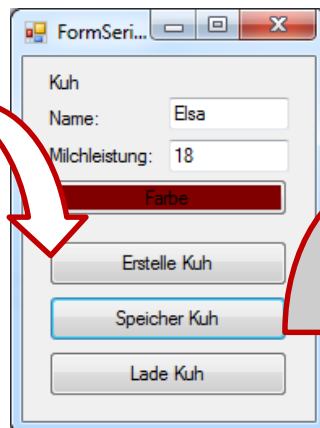
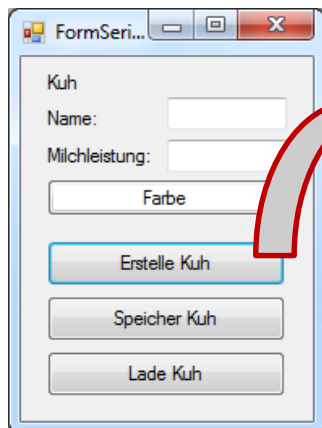
Serialisierbare Klasse (Beispiel)



```
[Serializable]
class Kuh
{
    /* ... */
    public static Kuh laden(string dateipfad){
        FileStream fs = new FileStream(dateipfad, FileMode.Open);
        BinaryFormatter bf = new BinaryFormatter();
        Kuh kuh = (Kuh)bf.Deserialize(fs);
        fs.Close();
        return kuh;
    }
    public void speichern(string dateipfad){
        FileStream fs = new FileStream(dateipfad, FileMode.Create);
        BinaryFormatter bf = new BinaryFormatter();
        bf.Serialize(fs, this);
        fs.Close();
    }
}
```


Serialisierung + SaveFileDialog (Beispiel)

```
public partial class FormSerialisierung : Form {  
    Kuh kuh;  
    /* ... */  
    private void buttonSerialisieren_Click(object sender, EventArgs e){  
        SaveFileDialog sfd = new SaveFileDialog();  
        if (sfd.ShowDialog() == DialogResult.OK) {  
            kuh.speichern(sfd.FileName);  
        }  
    }  
}
```



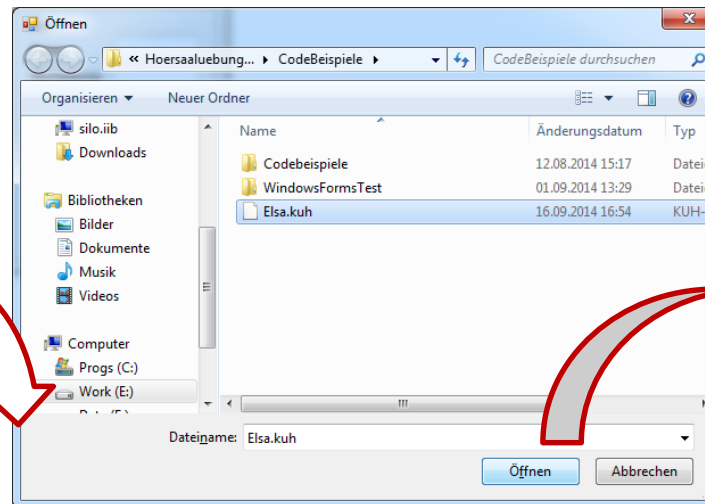
Deserialisierung + OpenFileDialog (Beispiel)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
public partial class FormSerialisierung : Form {  
    Kuh kuh;  
    /* ... */  
    private void buttonDeserialisieren_Click(object sender, EventArgs e) {  
        OpenFileDialog ofd = new OpenFileDialog();  
        if (ofd.ShowDialog() == DialogResult.OK) {  
            kuh = Kuh.laden(ofd.FileName);  
            textBoxName.Text = kuh.Name;  
            textBoxMilchleistung.Text = kuh.Milchleistung.ToString();  
            buttonColorPicker.BackColor = kuh.Farbe;  
        }  
    }  
}
```

Kuh
Name:
Milchleistung:
Farbe:
Erstelle Kuh
Speicher Kuh
Lade Kuh



Kuh
Name: Elsa
Milchleistung: 18
Farbe:
Erstelle Kuh
Speicher Kuh
Lade Kuh

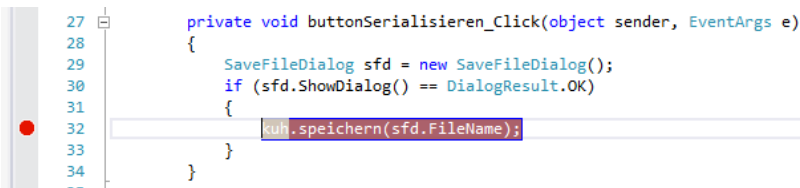
5. Debugging



TECHNISCHE
UNIVERSITÄT
DARMSTADT

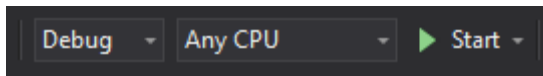


- Setzen eines Haltepunkts (Breakpoint)

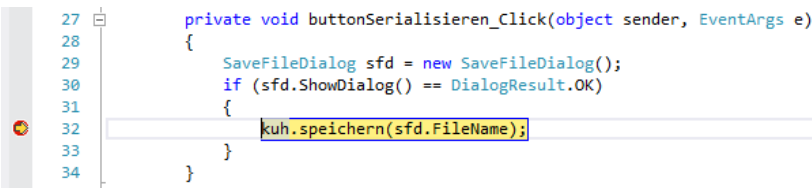


```
27 private void buttonSerialisieren_Click(object sender, EventArgs e)
28 {
29     SaveFileDialog sfd = new SaveFileDialog();
30     if (sfd.ShowDialog() == DialogResult.OK)
31     {
32         kuh.speichern(sfd.FileName);
33     }
34 }
```

- Debugger starten

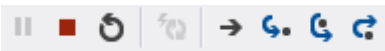


- Programm stoppt am Haltepunkt

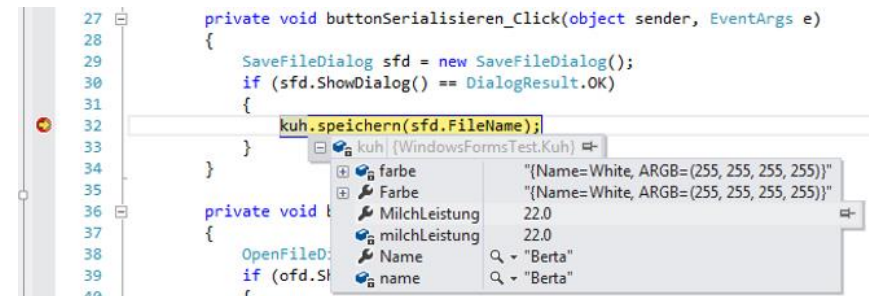


```
27 private void buttonSerialisieren_Click(object sender, EventArgs e)
28 {
29     SaveFileDialog sfd = new SaveFileDialog();
30     if (sfd.ShowDialog() == DialogResult.OK)
31     {
32         kuh.speichern(sfd.FileName);
33     }
34 }
```

- Zeilenweises Durchlaufen

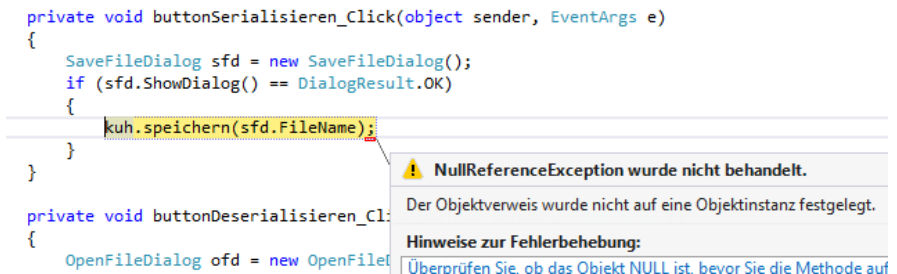


- Zustände von Variablen, Instanzen überwachen...



```
27 private void buttonSerialisieren_Click(object sender, EventArgs e)
28 {
29     SaveFileDialog sfd = new SaveFileDialog();
30     if (sfd.ShowDialog() == DialogResult.OK)
31     {
32         kuh.speichern(sfd.FileName);
33     }
34 }
35
36 private void buttonDeserialisieren_Click(object sender, EventArgs e)
37 {
38     OpenFileDialog ofd = new OpenFileDialog();
39     if (ofd.ShowDialog() == DialogResult.OK)
40     {
```

- Integrierte Hilfe bei „Fehlern“



```
private void buttonSerialisieren_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        kuh.speichern(sfd.FileName);
    }
}

private void buttonDeserialisieren_Click(object sender, EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog();
    if (ofd.ShowDialog() == DialogResult.OK)
    {
```

6. Demo



TECHNISCHE
UNIVERSITÄT
DARMSTADT

