Department Wirtschaftsinformatik
Fachbereich Wirtschaftswissenschaft

Programmieren für Wirtschaftswissenschaftler

SS 2015

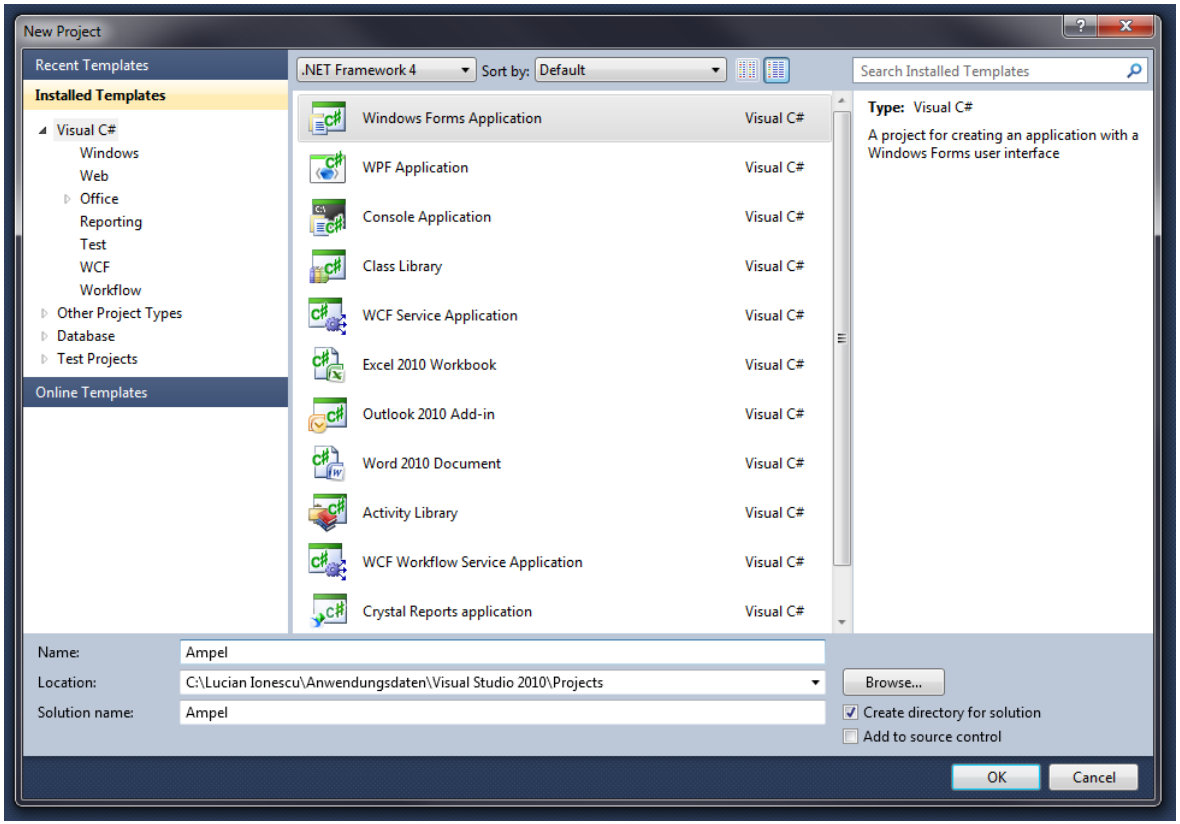
Lucian Ionescu

Blockveranstaltung 16.03–27.3.2015

7. Graphische Oberflächen

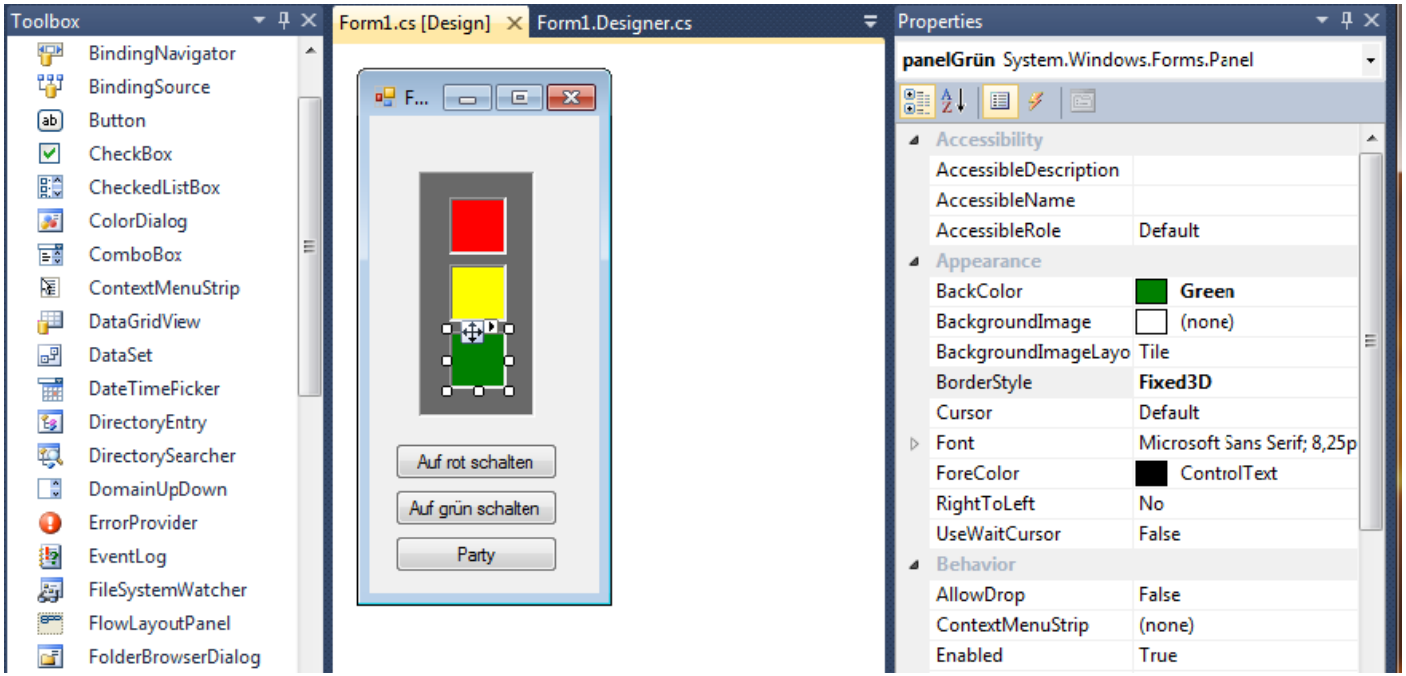
Ein neues Projekt anlegen

- Welche Projektart?
  - Windows Forms Application oder WPF Application
  - eine Form ist ein Objekt, genau wie dessen Inhalte, z.B. Buttons, Textfelder, Tabs, …



Graphische Oberflächen designen

- Über die **Toolbox** können Objekte per Drag & Drop auf die Form gezogen werden
- Alle Objekte haben Eigenschaften, die geändert werden können
  - Beispiel: Hintergrundfarbe der Panels *panel3* ist grün
  - Änderungen können im Code vorgenommen werden (panel3.BackColor = …) oder im Fenster **Eigenschaften**
  - insbesondere: Name ändern (button1, panel3, … sind nicht aussagekräftig!)



Ablauf graphischer Programme

- Die Main-Methode bildet wie in der Konsolen-Applikation den Startpunkt der Anwendung
  - Start der graphischen Oberfläche von hier aus
  - aber: Benutzeraktionen (wie Klicken auf Buttons, Mausbewegungen, Texteingaben) können danach ebenso Aktionen starten
  - dadurch erheblich flexibler als eine normale Konsolen-Anwendung
  - die Konsole lässt sich imitieren, indem man Textfelder erstellt, die man zur Ausgabe nutzen kann (Textboxes oder Captions)

```
namespace Ampel
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Unterschiedliche Auslöser für Aktionen

- Doppelklick auf Buttons erstellt/öffnet Methode, die beim Klick ausgeführt wird
- Auch andere Auslöser (**Events**) möglich (MouseHover, MouseDoubleClick, …)

```
namespace Ampel
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            panelRot.BackColor = Color.Black;
            panelGelb.BackColor = Color.Black;
            panelGrün.BackColor = Color.Black;
        }

        private void RotButton_Click(object sender, EventArgs e)
        {
            panelGrün.BackColor = Color.Black;
            panelGelb.BackColor = Color.Yellow;

            //Timer
            Application.DoEvents();
            System.Threading.Thread.Sleep(1000);

            panelGelb.BackColor = Color.Black;
            panelRot.BackColor = Color.Red;
        }
```

Ampel soll auf Rot geschaltet werden:
- Erst wird grün ausgeschaltet und Gelb aktiviert
- Nach Ablauf eines Timers („Gelbphase“) wird dann gelb deaktiviert und Rot aktiviert

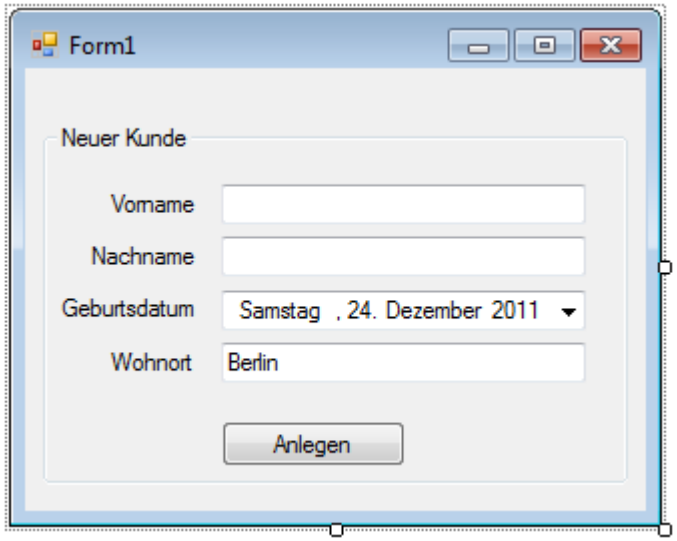
Verknüpfung mit Logik-Elementen

- Beispiel: In der Main-Methode oder im Konstruktor der Form können weitere Objekte erstellt werden (z.B. eine Ampelsteuerung)
- Forms sind normale Klassen und können Daten- und Funktionsmember enthalten!
- Es ist jedoch sinnvoll, die graphische Oberfläche von der Logik dahinter zu trennen (MVC-Prinzip: Model, View, Controller)

- Bsp.: Graphische Oberfläche für Konten-Beispiel?
  - Klassen Kunde und Konto werden angelegt
  - graphische Oberfläche kann zusätzlich erstellt werden, wird aber NICHT in die Logik-Klassen integriert!
  - Logik-Klassen sollten also unabhängig von der graphischen Oberfläche funktionieren und auch so angelegt werden!

Beispiel: Texteingabe

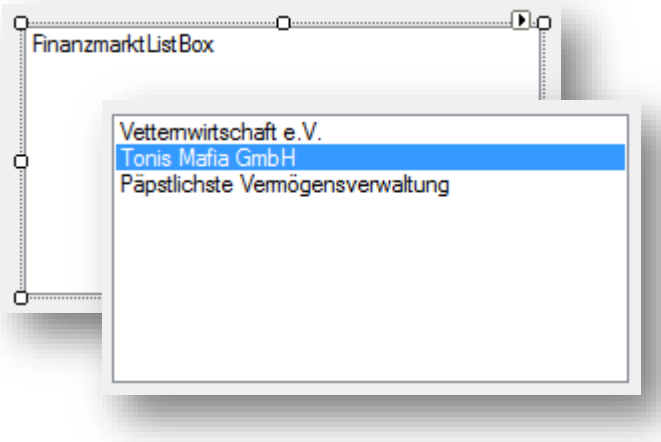
- Eine Bank benötigt ein System, mit dem Kunden in ihre Datenbank eingetragen werden können
- Benötigte graphische Objekte?
  - TextBox
  - Label
  - Button
  - evtl. Panel oder GroupBox?
  - TabControl?



- Klick auf Anlegen erstellt ein neues Objekt der Klasse „Person“ und speichert dieses z.B. in der Bank (Konsistenz der Eingaben testen?)
- Wichtig: Felder löschen, nachdem sie ausgelesen wurden
- Analog können Überweisungen getätigt werden (mit Suchfunktion?)

Graphische Oberflächen vs. Logische Strukturen

- Graphische Elemente bieten Strukturen zur Nutzung der vom Nutzer erstellten Klassen an
- Aber graphische Oberflächen sollten nur Logik beinhalten,
  - die zur Anzeige notwendig sind
  - die der Kommunikation mit den Logik-Klassen dienen



- Beispiel Listbox
  - besitzt die Property *Items* vom Typ *List<Object>*
    - die Liste kann nur Elemente beinhalten, die vom Typ *Object* sind
    - Und das Gute daran ist– jede Klasse ist auch vom Typ *Object*!
    - daher können Elemente einer beliebigen Klasse in der Liste gespeichert werden
    - hier sinnvoll: **List<Bank>!**
  - aber: *Items*–Listen der Listbox nicht zur Verwaltung der Kunden nutzen, dort haben sie nichts verloren, sondern eigene Listen erstellen!

Graphische Oberflächen vs. Logische Strukturen

- Beispiel ListBox

  Wie werden die Listeneinträge angezeigt?

  – Object-Klasse hat eine Methode *ToString()*

  – Diese Methode muss man überschreiben!

    - Beispielsweise muss die Klasse Bank dann diese Methode beinhalten:

```
public override string ToString()
{
    return name;
}
```

  – Achtung: wenn die Methode nicht überschrieben wird, gibt sie einfach nur den **Objekttyp** als String zurück!

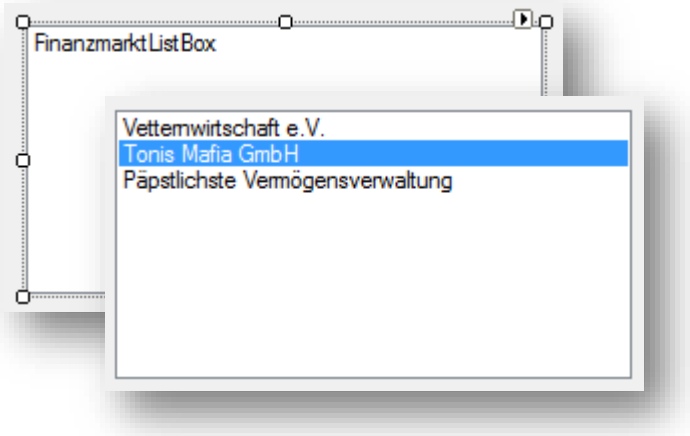
  – Dann steht dort etwas wie „Program.Bank“ anstatt ein bestimmter String, z.B. aus der Variablen „name“

Graphische Oberflächen vs. Logische Strukturen

- Beispiel Listbox (fortgeführt)
  - Wie werden die Listeneinträge wieder ausgelesen?
    - Listbox.SelectedItem ist das gewählte Element, aber vom Typ **Object**
    - Wir wissen aber, dass es vom Typ **Bank** ist, also können wir es explizit *casten*

```
private void KontenAusgeben_Click(object sender, EventArgs e)
{
    Bank bank = (Bank) FinanzmarktListBox.SelectedItem;
    TextBoxOutput.Text = bank.KontenAusgeben();
}
```

    - Nach dem Casten können wir wie gehabt auf alle nach außen sichtbaren Methoden und Datenfelder der Bank zugreifen



Das nur als ersten Einblick…

- Es gibt viele Online-Tutorials, die ausführlicher auf das Thema eingehen
- Nicht zu unterschätzen dabei Youtube-Tutorials! (z.B. mal *C# Windows Forms* suchen)