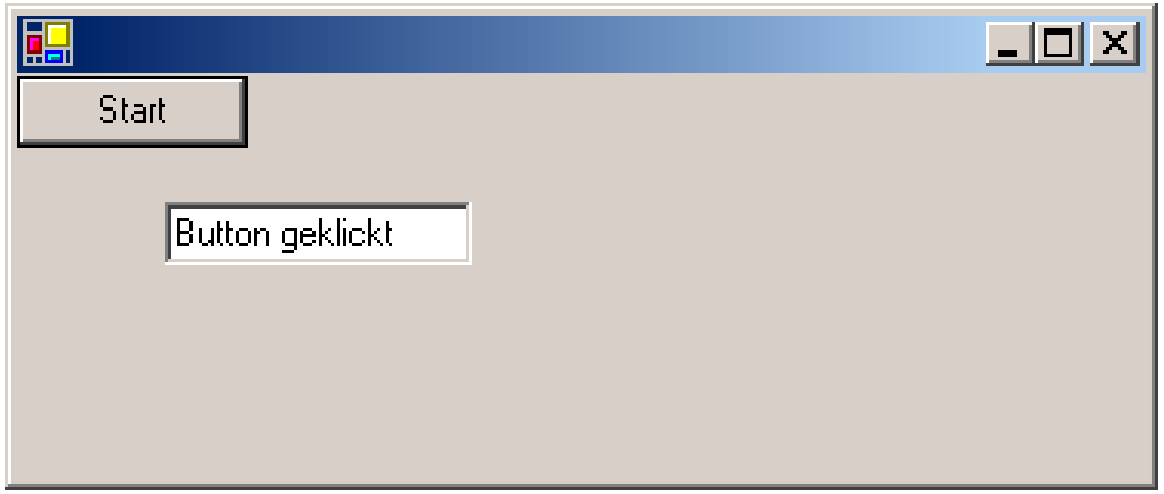
Windows Forms

Ausgangssituation

Wir wollen eine einfache Windowsanwendung mit einem Fenster erstellen. Dieses Fenster enthält (Container!) einen Button der beim klicken das Schreiben in ein Textfeld bewirkt.



Was ist zu tun ?

- **Programmlogik**
  1. **Aufbau der Präsentationsschicht (Ein- Ausgabe)**
  2. **Hinzufügen der Logik (hier: klicken mit Aktion)**
- **Programmerstellung mit Notepad bzw. mit Visual Studio**

Windows Forms

Aufbau der Präsentationsschicht (Ein- Ausgabe)

- **wir benötigen ein Formular (Klasse Form), in das wir unsere (Steuer-) Elemente einbetten**

```
public class Form1 : System.Windows.Forms.Form
```

- **einzubettende Steuerelemente:**
  - **Button:**
```
System.Windows.Forms.Button button1;
button1 = new System.Windows.Forms.Button();
```
  - **Textfeld:**
```
System.Windows.Forms.TextBox tb;
tb = new System.Windows.Forms.TextBox();
```

- **Steuerelemente dem Container (Form) hinzufügen**

```
this.Controls.Add(button1); // this zeigt auf Formobjekt
this.Controls.Add(tb);
```

- **Formobjekt anlegen und Anwendung starten**

```
System.Windows.Forms.Application.Run(new Form1())
```

Windows Forms

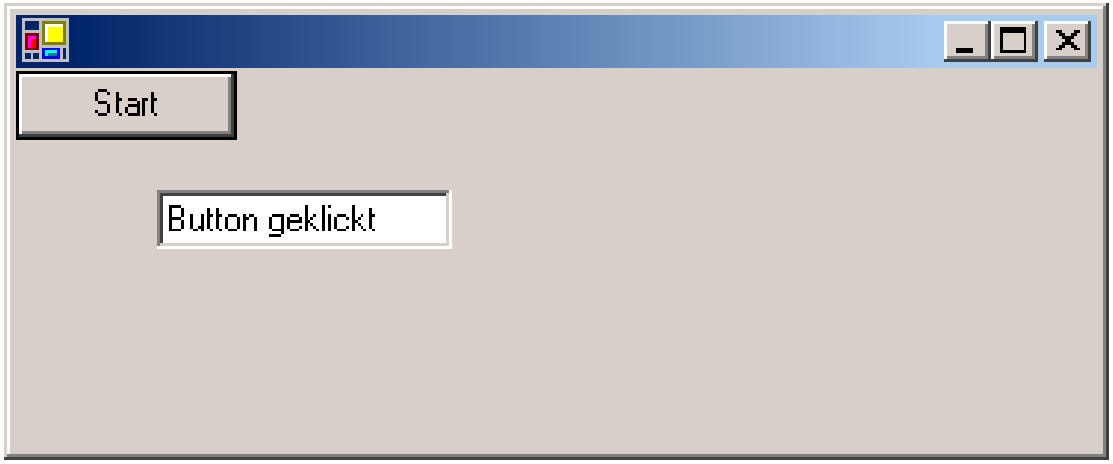
Programmerstellung mit Notepad

```
public class Form1 : System.Windows.Forms.Form //ohne Event
{
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.TextBox tb;
    public Form1() // Container für Button u. Textfeld
    {
        button1 = new System.Windows.Forms.Button();
        button1.Text = "Start";
        tb = new System.Windows.Forms.TextBox();
        tb.Location = new System.Drawing.Point(48, 40);
        this.Controls.Add(button1);
        this.Controls.Add(tb);
    }
    static void Main()
    {System.Windows.Forms.Application.Run(new Form1());}
}
```

Windows Forms

Hinzufügen der Logik (hier: klicken mit Aktion)

- Anforderung:
  Beim Mouseclick
  auf dem Button
  soll in einem Feld
  „Button geklickt“
  ausgegeben werden.



- Lösung: Eventhandling auf Basis des Beobachtermusters

Entwurfsmuster Beobachter (Publisher-Subscriber)

Windows Forms

Beobachtermuster im Beispiel

- Publisher: Button
  (hat Multicast-Delegate)
- Subscriber: Alle, die vom Button informiert werden wollen, wenn dieser geklickt wird
  Hierzu:
  Diese müssen den Button kennen und sich bei dem Button anmelden (unterschreiben = subscribe)
  (eigene Methode dem Multicast-Delegaten des Button hinzufügen)
- Auslöser: Benutzer der den Button klickt
  => Button informiert alle Subscriber
  (Multicast-Delegate wird ausgeführt)

Eventhandling in .Net basiert auf beschriebenem Konzept

Windows Forms

Eventhandling

1. **Definitonsphase**
   Publisher: Button „Start“
   Methoden (fkt1,.., fktN hier: btClick) unterschreiben beim Publisher d. h. an Multicastdelegaten (button1.Click) binden (+=)


```
button1.Click += new System.EventHandler(this.btClick);
```

2. **Ablaufphase**
   - **Komplette Interaktion erfolgt über Events (Warteschleife:** `Application.Run(new Form1());`**)**
   - **Event (z. B. Click) wird am Button „Start“ durch Benutzer ausgelöst**
     => Mit dem Event verbundener Delegate (Namenskonvention) wird ausgeführt, d. h. alle Methoden werden aufgerufen (hier: nur btClick)

Windows Forms

```
public class Form1 : System.Windows.Forms.Form   // mit Event
{
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.TextBox tb;
    public Form1(){
        button1 = new System.Windows.Forms.Button();
        button1.Text = "Start";
        tb = new System.Windows.Forms.TextBox();
        tb.Location = new System.Drawing.Point(48, 40);
        this.Controls.Add(button1);
        this.Controls.Add(tb);
        button1.Click += new System.EventHandler(this.btClick);
    }
    static void Main()
    {System.Windows.Forms.Application.Run(new Form1());}
    private void btClick(object sender, System.EventArgs e)
                                {tb.Text = "klick";}
}
```

Windows Forms

Die Nachrichtenschleife

- wird durch Aufruf der statischen Methode Application.Run(..) gestartet
- jeder Thread kann nur eine Nachrichtenschleife verarbeiten (d. h. Run nur einmal aufrufbar im Thread)
- Exit bzw. ExitThread der Klasse Application beendet Nachrichtenschleife (z. B. Schließen der Form)
- Tritt Event ein, wird EventHandler, d.h. die damit verbundenen Methoden aufgerufen

  Problem:

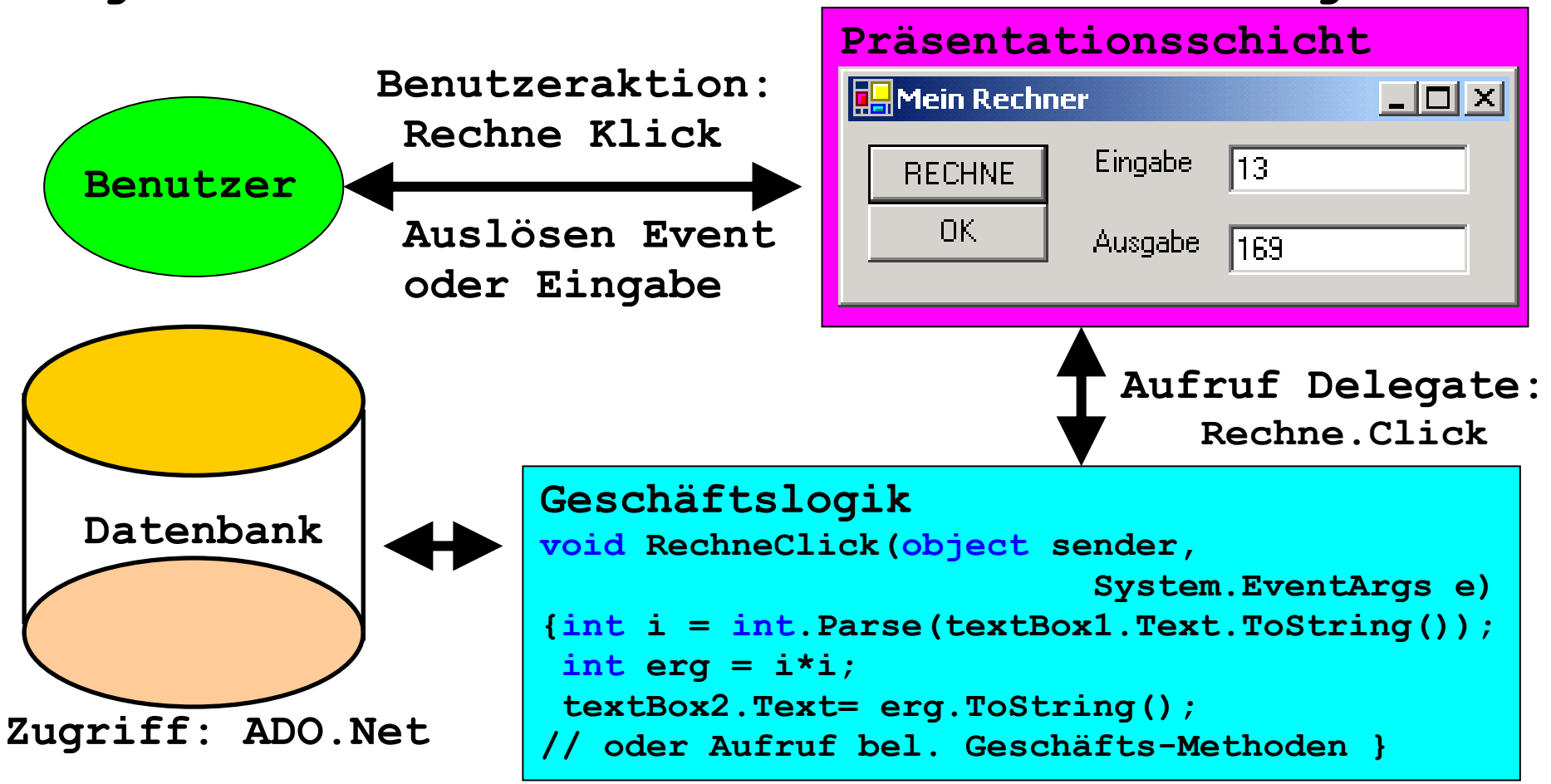
  => Programm wird erst nach dem Beenden dieser Methoden fortgesetzt

  Lösung:

  Statische Methode Application.DoEvents() ermöglicht die Inanspruchnahme der CPU durch andere Eventhandler

Windows Forms

Allgemeiner Aufbau einer Windows-Anwendung



Windows Forms

Nachteile der obigen Implementierung

- **Definition des Designs der Oberfläche und der Funktionalität sind in der Klasse**

```
public class Form1 : System.Windows.Forms.Form
```

  verknüpft (vermischt)
  => Ziel: Trennung von Design und Implementierung

- **Design der Oberfläche erfolgt in objektorientierter Form in der Programmiersprache (hier: C#)**
  => Ziel: Design durch beschreibende (deklarative) Form ermöglichen
  Lösung in .Net: Windows Presentation Foundation XAML