
Inhaltsverzeichnis

1 EINLEITUNG	1
---------------------	----------

1.1

3 ELEMENTARE SPRACHELEMENTE	71
3.1 Einstieg	71

3.7 Anweisungen (zur Ablaufsteuerung)

4.7	Vertiefungen zum Thema Methoden
4.7.1	

195

Inhaltsverzeichnis

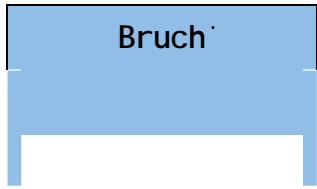
13.8 Timer

18.2

Inhaltsverzeichnis

Abschnitt 1.1 Beispiel für die objektorientierte Softwareentwicklung mit C#

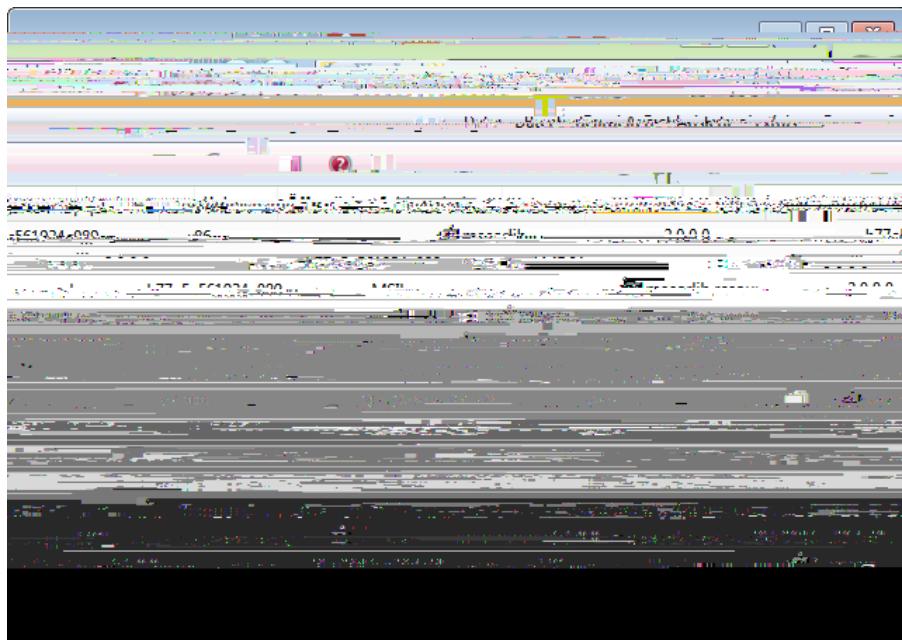
Abschnitt 1.1 Beispiel für die objektorientierte Softwareentwicklung mit C#



Bruch

Abschnitt 1.1 Beispiel für die objektorientierte Softwareentwicklung mit C#

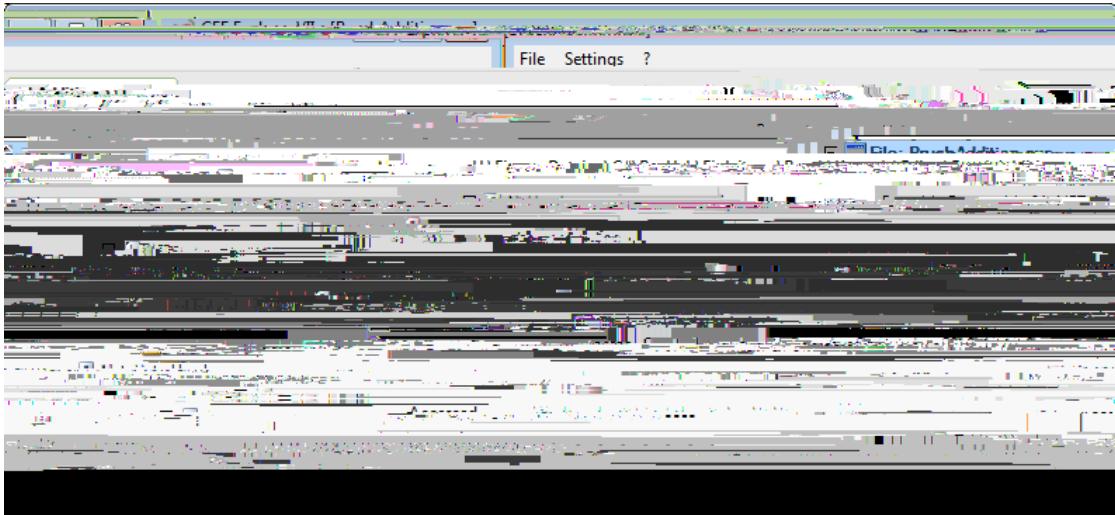
Die .NET – Version 4.0 bringt vor allem eine „Kantenglättung“ (Schwichtenberg 2010, S. 139) und Detailverbesserungen (z.B. bei der Unterstützung von Mehrkernprozessoren).



Eine Besonderheit bei GAC-Assemblies ist die erforderliche Signatur (siehe Spalte **Öffentlicher Schlüsseltoken**)

€ X86 (x86)

€



Beim Starten eines Assemblies auf einer 64-Bit – Maschine entscheidet die Laufzeitumgebung darüber, ob ein 32- oder ein 64-Bit – Prozess entsteht, wobei natürlich das Attribut **ProcessorArchitecture** eine wichtige Rolle spielt. Ein unter .NET 1.x erstelltes Assembly wird sicherheitshalber im 32-Bit – Modus ausgeführt, weil der damalige Compiler das Attribut **ProcessorArchitecture** noch nicht kannte.

In Abschnitt 2.2 wird sich zeigen, dass die im Kurs bevorzugte Entwicklungsumgebung für .NET – Programme *Visual Studio 2010* unter Windows 64 bei neuen EXE - Projekten in C# die Compiler- Voreinstellung MSIL bzgl. der Zielpf

2 Werkzeuge zum Entwickeln von C# - Programmen

In diesem Abschnitt werden kostenlos verfügbare Werkzeuge zum Entwickeln von .NET - Applika-

Kapitel 2: Werkzeuge zum En

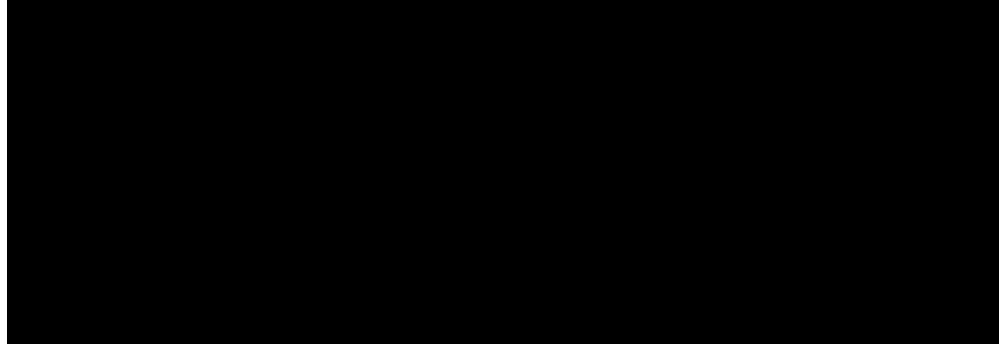
Sind *mehrere* Quellcodedateien in ein Assembly zu übersetzen, gibt man sie beim **csc**-Aufruf hintereinander an, z.B.

```
csc Bruch.cs BruchAddition.cs
```

Dabei sind auch Jokerzeichen erlaubt, z.B.:

```
csc *.cs
```

Das entstehende Assembly erbt seinen Namen von der Startklasse (mit der **Main()**-Methode):



Über die Befehlszeilenoption **out** kann der Assembly-Name aber angepasst werden, z.B.

Abschnitt 2.2 Microsoft Visual Studio 2010

Das TextBox

Kapitel 2: Werkzeuge zum En

Abschnitt 2.2 Microsoft Visual Studio 2010

Kapitel 2: Werkzeuge zum En

Nun kann in folgender Dialogbox auf der Registerkarte

3 Elementare Sprachelemente

In Kapitel 1 wurde anhand eines halbwegs realistischen Beispiels versucht, einen ersten Eindruck von der objektorientierten Softwareentwicklung mit C# zu vermitteln. Nun erarbeiten wir uns die Details der Programmiersprache C# und beginnen dabei mit elementaren Sprachelementen. Diese dienen zur Realisation von Algorithmen innerhalb von Methoden und sehen bei C# nicht wesentlich

Abschnitt 3.1 Einstieg

€

Abschnitt 3.2 Ausgabe bei Konsolenanwendungen

Die beschriebenen Formatierungstechniken sind nicht nur bei Konsolenausgaben zu gebrauchen.

Abschnitt 3.3 Variablen und Datentypen

Abschnitt 3.3 Variablen und Datentypen

3.4 Einfache Techniken für Benutzereingaben

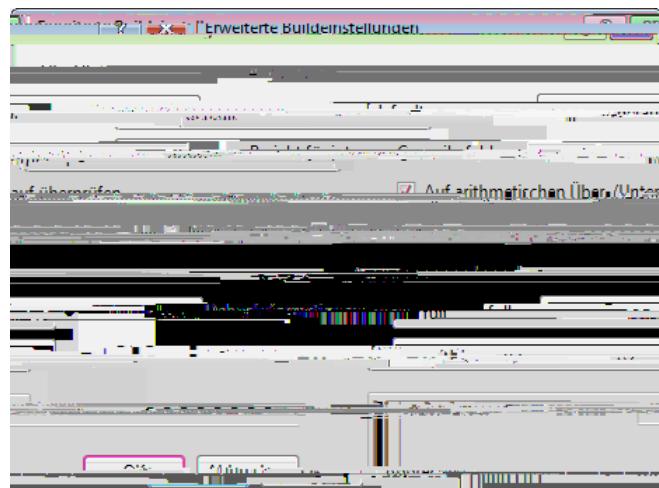
Quellcode

- € Die Arbeitsweise einer Methode kann von Argumenten (Parametern) abhängen.
- €


```
· (1 << i & cbit) != 0
```

Dieser logische Ausdruck wird bei einem Schleifendurchgang genau dann wahr, wenn das zum aktuellen im *i*-Wert korrespondierende Bit in der Binärdarstellung des untersuchten Zeichens den Wert Eins hat.

3.5.7



Der Vollständigkeit halber soll

wird mit folgender Meldung abgebrochen:

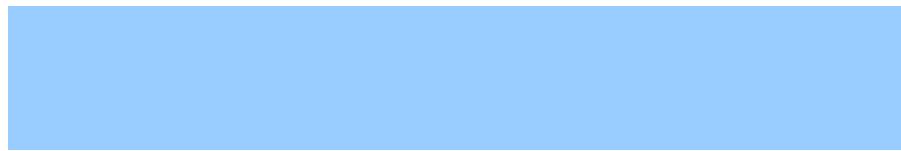
Console-Methode

Oft möchte man jedoch z.B.

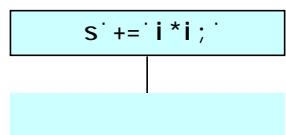
- die Ausführung einer Anweisung (eines Anweisungsblocks) von einer *Bedingung* abhängig machen
- oder eine Anweisung (einen Anweisungsblock) *wiederholt* ausführen lassen.

Abschnitt 3.7 Anweisungen (zur Ablaufsteuerung)

Quellcode	Ausgabe



```
s := i * i ;
```



A diagram illustrating a code snippet. The code `s := i * i ;` is enclosed in a light blue rectangular box. A vertical line points downwards from the bottom of this box to a second, longer light blue horizontal bar below it.

Quellcode	Ausgabe
using	


```
    ...  
    for (i = 2; i <
```

Abschnitt 3.7 Anweisungen (zur Ablaufsteuerung)

Der **FullName**

Abschnitt 4.1 Überblick, historische Wurzeln, Beispiel

Zwei Bemerkungen zum Kopf einer Klassendefinition:

- € Im Beispiel ist die Klasse Bruch als **public** definiert, damit sie uneingeschränkt von anderen Klassen (aus beliebigen Assemb

4.2.1 Sichtbarkeitsbereich, Existenz und Ablage im Hauptspeicher

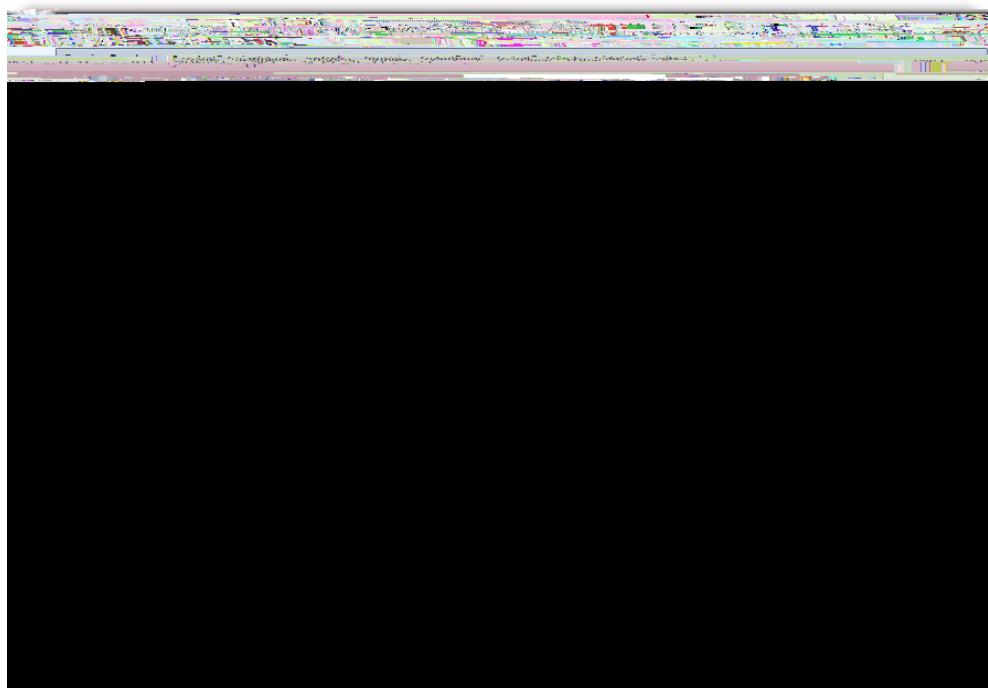
Von den lokalen Variablen unterscheiden sich die Instanzvariablen (Felder) einer Klasse vor allem bei der *Zuordnung*

Abschnitt 4.2 Instanzvariablen

Ziel einer typischen Klassendefini

Abschnitt 4.3 Instanzmethoden

€ Die Bruch-Instanzmethode Addieren()



- € Die Namen sind identisch.
- €

Abschnitt 4.4 Objekte

Abschnitt 4.4 Objekte

Abschnitt 4.4 Objekte

Die neue Option wurde unter der Bezeichnung **Objektinitialisierer**

immerhin die Tätigkeit des Garbage Collectors zum Programmende

Im **set**

```
Console.WriteLine("Bisher wurden " +
```



```
public Bruch() {  
    zaehler = zaehlerVoreinst;  
    nenner =
```



```
public class Sohn {  
    Famille f;  
    string name;  
    int alter;  
  
    public
```



```
<Window x:Class="BruchKuerzenGUI.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="
```


Nach der neugierigen Beschäftigung mit den Bedienmöglichkeiten des Eigenschaftenfensters kehren wir nun den Entwicklungsaufgaben im aktuellen Projekt zurück und ändern ...

€

Die Eigenschaften von mehreren, gleichzeitig markierten Formular-Objekten lassen sich in einem Arbeitsgang ändern.

Zwar ist eine Eigenschaftsmodifikation zur

publ i c

5 Weitere .NETte Typen

Im folgenden Beispielprogramm wird beim Ganzzahlliteral 13 (vom Strukturtyp **Int32**) über die von **System.Object** geerbte Methode **GetType()** erfolgreich der Datentyp erfragt:

Quellcode	Ausgabe
using	

Anschließend wird die Methode **Console.WriteLine()** mit Aktualparametern vom Typ **String** bzw. **Object** aufgerufen. Auch zur Ausführung des **GetType()**-Aufrufs in der folgenden Anweisung

```
Console.WriteLine(13.GetType());
```

wird per **box**-OpCode ein Objekt erzeugt.

5.3 Arrays

Ein Array ist ein Objekt, das als Instanzvariablen eine feste Anzahl von Elementen desselben Da-

Abschnitt 5.3 Arrays

Im Unterschied zum **int[,]** - Objekt **matrix x**

Quellcode	Ausgabe
using	

Abschnitt 5.4 Klassen für Zeichenketten

Für den lesenden oder schreibenden Zugriff auf das i

Abschnitt 5.7 Übungsaufgaben zu Kapitel 5

6 Vererbung und Polymorphie

Im Manuskript war schon mehrfach davon die Rede, dass die .NET k/TT0 1 Tf6 Dat

Abschnitt 6.5 Erbstücke durch spezialisierte Varianten verdecken

Quellcode	Ausgabe
-----------	---------

```
class Prog {
    static void Main() { static
```


Abschnitt 6.7 Polymorphie (Methoden überschreiben)

Abschnitt 6.8 Abstrakte Methoden und Klassen

Unterklassenmethode (Verdecken und Überschreiben, vgl. Abschnitt 6.7) kommt bei einer abstrakten Basisklassenmethode nur das Überschreiben in Frage, wobei das zugehörige Schlüsselwort **override** anzugeben ist.

7 Typgenerizität und Kollektionen

In C# haben die Felder von Klassen und Strukturen sowie die Parameter von Methoden einen festen Datentyp, so dass der Compiler für Typsicherheit sorgen, d.h. die Zuweisung ungeeigneter Werte

Abschnitt 7.2 Generische Klassen

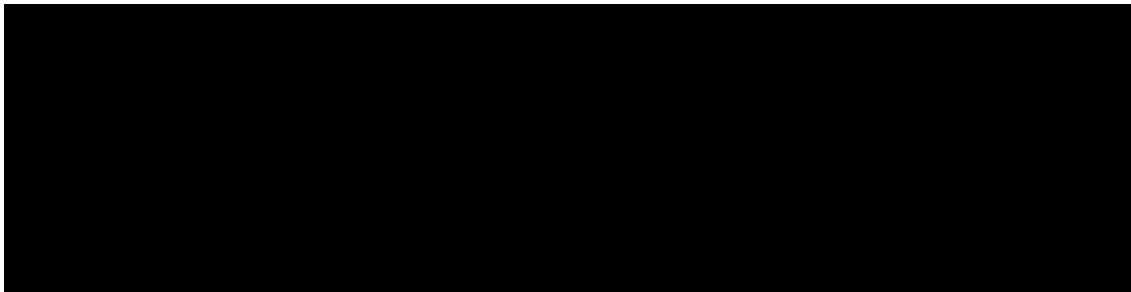
Abschnitt 7.5 Weitere Klassen aus dem Namensraum System.Collections.Generic

£ **public void IntersectWith(IEnumerable<T> kollektion**



using

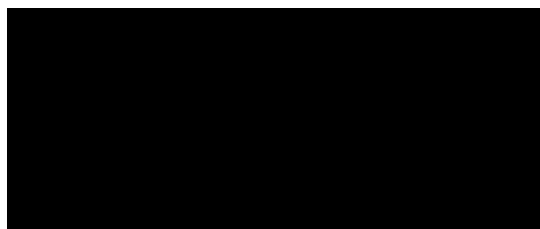
Betrachten wir zur Illustration eine Konsolen- und eine GUI-Anwendung zum Kürzen von Brüchen.
Bei der Konsolenanwendung (vgl. Abschnitt 4.1.3)



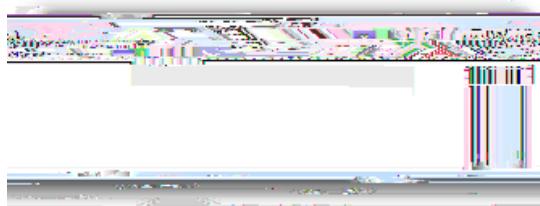
wird der gesamte Ablauf vom Programm diktiert:

- € Es fragt nach dem Zähler.
- € Es fragt nach dem Nenner.
- € Es schreibt das Erge

Abschnitt 9.2 Elementare Bausteine einer WPF-Anwendung



SingleBorderWindow



ToolWindow

Ein **Dele**

+ =

Abschnitt 9.3 Delegaten und CLR-Ereignisse

Abschnitt 9.3 Delegaten und CLR-Ereignisse

```
class EventConsumer : Window {
```

Abschnitt 9.3 Delegaten und CLR-Ereignisse

Abschnitt 9.4 Die Extended Application Markup Language (XAML)

9.4.2.1 Wurzelement

Eine XAML-Datei einhält nur *ein* Wurzelement, wobei für uns

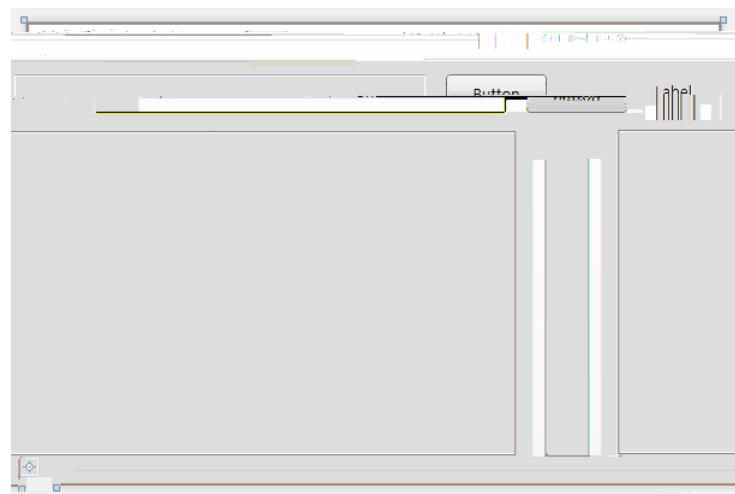
Abschnitt 9.4 Die Extended Application Markup Language (XAML)

<

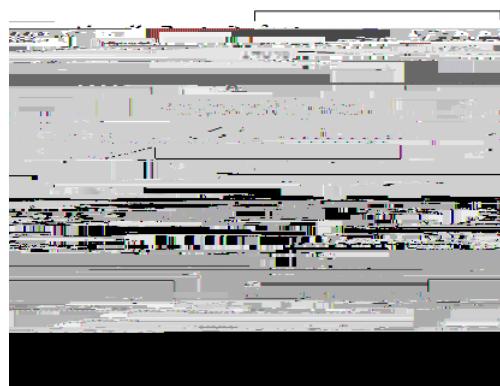
Abschnitt 9.4 Die Extended Application Markup Language (XAML)

Abschnitt 9.4 Die Extended Application Markup Language (XAML)

publ i c



Zur Erleichterung der Arbeit lässt sich mit dem **Zoom-Werkzeug** des WPF-Designers (oben links) eine passende Ansicht einstellen:



Wählen Sie zunächst für das Ha

Außerdem werden im Beispiel horizontale Abstände angezeigt.

Man kann also z.B. so vorgehen, um die Steuerelemente für den RSS-Feed - Reader anzurichten:

Eine weitergehende Analyse des automatisch erstellten Quellcodes sparen wir uns.

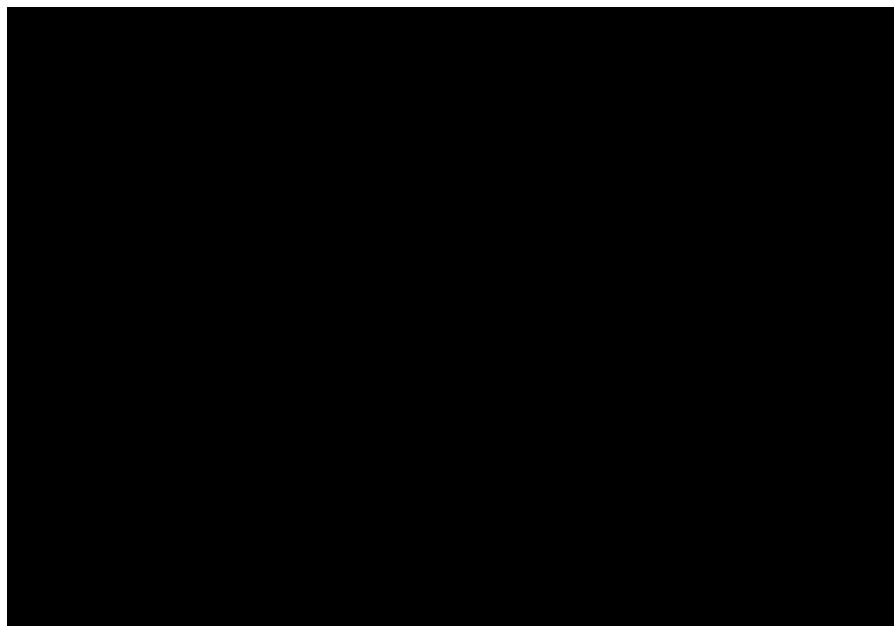
9.5.6 Click-Ereignisbehandlung zum Befehlsschalter (Teil 1)

Wir erstellen eine Ereignisbehandlungsmethode, die durch das Betätigen des Befehlsschalters (per Mausklick oder **Enter**-Taste) ausgelöst wird. Sie soll folgende Leistungen erbringen:

- € Unter Verwendung der **Text**

Um ein **ListBox**-Steuerelement zu füllen, kann man seiner Eigenschaft **ItemsSource** ein Objekt aus einer Klasse zuweisen, welche die Schnittstelle **IEnumerable** im Namensraum

Abschnitt 9.6 Routingereignisse



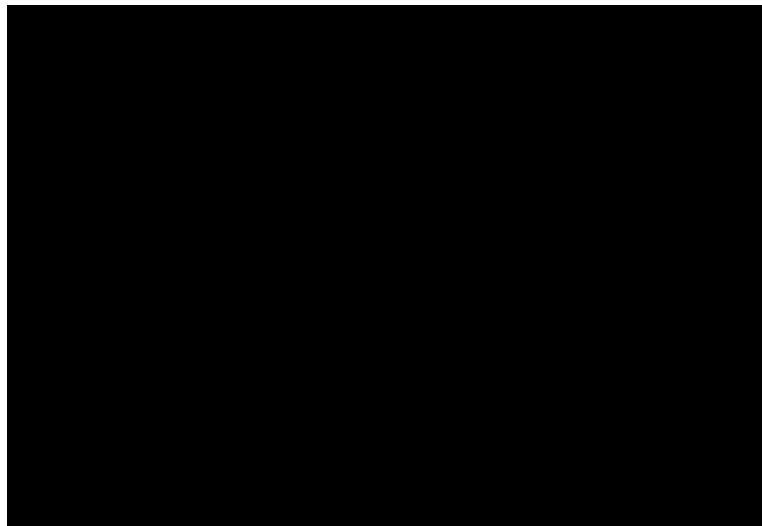
<Grid>

Abschnitt 9.7 Layoutcontainer

Abschnitt 9.8 Basiswissen über Steuerelemente

bleiben andere Ereignisbehandl

Abschnitt 9.8 Basiswissen über Steuerelemente



€ Es resultiert der XAML-Code:

```
<Button Name="count" Width="75" Grid.Column="1" . . . IsDefault
```


.....<StackPanel>

Eine GUI-Anwendung wartet die meiste Zeit darauf, dass eine Ihrer Methoden über ein (meist vom Benutzer ausgelöste)

10 Ausnahmebehandlung

€

Abschnitt 10.2 Ausnahmen abfangen

try

- € Bessere Fehlerinformationen für den Aufrufer
Über ein

Die `ToString()`



12 Ein- und Ausgabe über Datenströme

Praktisch jedes Programm muss Daten aus externen Quellen einlesen und/oder Verarbeitungsergebnisse in externe Senken schreiben. Wir haben uns bisher auf die Eingabe per Tastatur sowie die Ausgabe per Bildschirm beschränkt und müssen a

Abschnitt 12.1 Datenströme aus Bytes

Nun steht der `Close()`-Aufruf im **finally**-Block der vom Compiler aus der **using**-Anweisung erstellten **try-finally** – Anweisung.

Bei den Beispielen im weiteren Verlauf von Kapitel 12 werden wir der Einfachheit halber meist auf

12.1.6.1 Öffnungsmodus

Beim Erstellen eines **FileStream**-Objekts kann man den Öffnungsmodus über einen Wert des Enumerationstyps **FileMode** wählen:

In Abschnitt 12.2.3 beschäftigen wir uns mit dem Schreiben und Lesen von kompletten Objekten (oder auch Strukturinstanzen), wobei eine Binärdatei oder eine

13 Multithreading

Wir sind längst daran gewöhnt, dass moderne Betriebssysteme mehrere Programme (Prozesse) parallel betreiben können, so dass z.B. ein längere

Abschnitt 13.2 Threads synchronisieren

einen ausreichenden Vorrat antrifft, findet sein **Wait()** - Aufruf in einer **while**

Abschnitt 13.2 Threads synchronisieren

Abschnitt 13.2 Threads synchronisieren

13.3 *Threads stoppen*

Abschnitt 13.4 Thread-Lebensläufe

Anstelle eines Threadpool - Auftrags ist ein eigener (gem. Abschnitt 13.1 erstellter) Thread z.B.

Aktuelle Zufallssumme: 494947456

Der Arbeits Thread ist fertig

Der primäre Thread ist aufgewacht.

13.7 Asynchronous Programming Model

In diesem Abschnitt wird das **APM**–Entwurfsmuster (*Asynchronous Programming Model*) vorge-


```
for (int i = 0; i < anz; i++)  
    ThreadPool.QueueUserWorkItem(RandomSumTP);
```

kann die TPL im aktuellen Beispiel allerdings *nicht* punkten.

Soll eine Aufgabe bzw. die zugrunde liegende Methode einen **Rückgabewert** liefern, erzeugt man ein Objekt der generischen Klasse **Task<TResult>**, die von der Klasse **Task** abstammt und für den Rückgabetyp einen Typparameter besitzt. Im folgenden Programm

```
using System;
using System.Threading.Tasks;

class TaskDemo {
    static int sg = 10000;

    static double SampleMean() {
        Random zzg = new Random();
        double erg = 0.0;
        for (int j = 0; j < sg; j++)
            erg += 5.0 + zzg.NextDouble() * 10.0;
        return erg / sg;
    }

    static void Main() {
        Task<double> task = Task.Factory.StartNew<double>(SampleMean);
        Console.WriteLine("Stichprobenmittel = " + task.Result);
    }
}
```

entsteht aus der Methode `SampleMean()`, die aus einer Stichprobe mit `sg` Zufallszahlen den Mittelwert berechnet.

dazu verwendet werden,

Abschnitt 13.10 Übungsaufgaben zu Kapitel 13

Abschnitt 14.2 Internet - Ressourcen per Request/Response – Modell nutzen

14.2.3.4 POST

Beim **POST**-Verfahren, das man im **form** - Tag einer HTML-Seite durch die Angabe

```
method="post"
```


- € eingehende Nachrichten protokollieren, z.B. mit einem mehrzeiligen **TextBox**-


```
    int alter;
```


Abschnitt 15.5 View-Objekt zu einer Kollektion

```
    public MainWindow() {
        InitializeComponent();
        personen
```

Rückgabewert vom Typ **bool** zeigt. Diese Methode wird für jedes Listenelement aufgerufen und entscheidet darüber, ob ein Element im Datenbindungsziel erscheint (Rückgabe **true**) oder nicht (Rückgabe **e**)

0 . 1 3 9 9 9

16.1.2 Modaler und nichtmodaler Auftritt

€

Abschnitt 16.2 Modale Dialogfenster

Abschnitt 16.3 Nicht-modale Dialogfenster

Schalter **Abbrechen** angeboten, der aber (anders als beim modalen Dialog) mit einer **Click**-Behandlungsmethode ausgestattet werden muss (siehe unten)

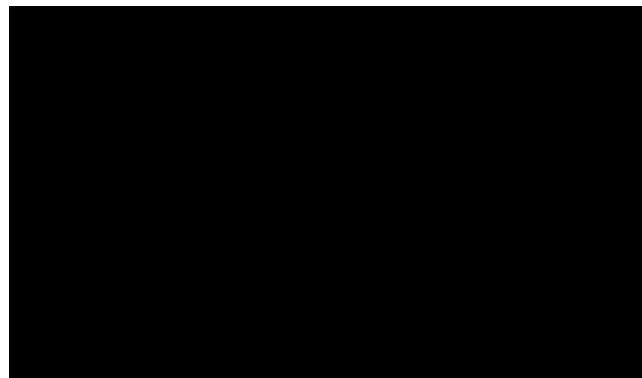
Mit dem Ereignis FindNext



Abschnitt 17.1 Konfigurationsdateien im XML-Format

namespace





Im XAML-Code werden Ausdehnung und Positionierung der Ellipse über die vertrauten Eigenschaften bzw. Attribute geregelt:

```
<Window x
```


Abschnitt 18.1 Vektorgrafik mit Objekten aus der Shape-Klassenhierarchie

Hier wird vor einem schwarzen Hintergrund ein Text in Weiß und dann (über die Eigenschaften **X** und **Y** des **TranslateTransform**-Objekts) leicht verschoben nochm

19 Text darstellen und drucken

Wir behandeln zunächst die Klassen für eine flexible Bildschirmausgabe von Fließtexten und beschäftigen uns dann mit der Druckausgabe mit Hilfe des XPS-Seitenformats (*XML Paper Specification*).

19.1 Die Klasse *TextBlock* für kurze Fließtexte

Für kurze Textausgaben im Rahmen einer Bedienoberfläche bietet

Abschnitt 19.4 XPS-basierte Druckausgabe

Für eine *mehrseitige* Druckausgabe per

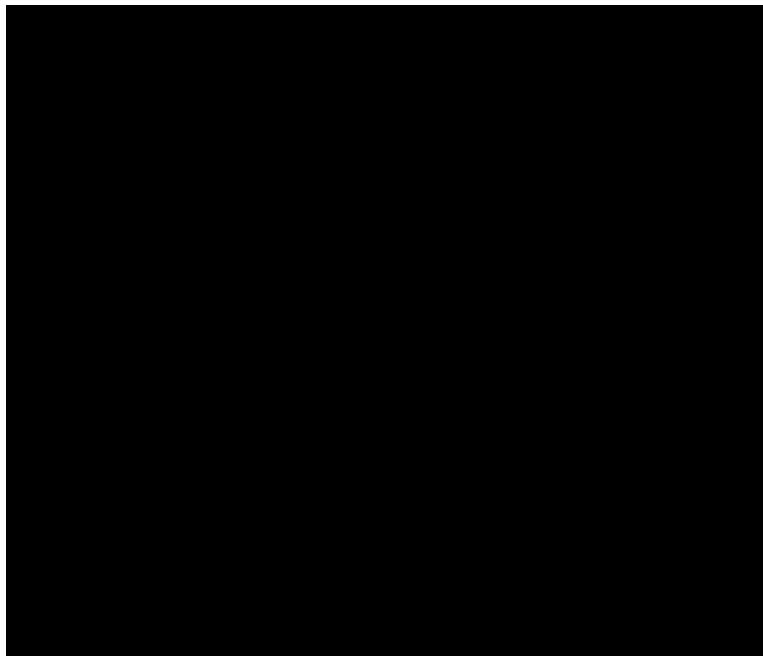
Abschnitt 20.2 Relationale Datenbanken

Folglich kann man z.B. beim Auflisten von Bestel

Abschnitt 20.3 SQL

Kapitel 20: Datenbankpr





Der *logische Datenbankname* (hier: **Northwind**) wird per Voreinstellung aus der **mdf**-Datei übernommen und kann bei Bedarf im Management Studio geändert werden.

20.4.3 Konfiguration

20.4.3.1 Rechteverwaltung

Zum Provider **SqlClient**, den wir diesem Kurs ausschließlich einsetzen werden, gehören die fol-

Abschnitt 20.5 ADO.NET

- € Es führt mit Hilfe von **Command**-Objekten **SELECT**- und DML-Abfragen durch und vermittelt dabei zwischen den lokal (in einem

Kapitel 20: Datenbankpr

- € Für jede Wertveränderungen erhält man vier Ereignisse in folgender Reihenfolge:
ColumnChanging, **ColumnChanged**, **RowChanging** und **RowChanged**.
- € Bei der **ColumnChanging**-Behandlung ist der vorgeschlage

DataRowState	Beschreibung
Unchanged	Der Datensatz wurde nicht verändert.
Added	Der Datensatz wurde hinzugefügt.
Deleted	Der Datensatz wurde gelöscht.
Modified	Der Datensatz wurde geändert.


```
// Das Explizite Öffnen und Schließen der Datenbankverbindung
```


Zur Konkretisierung des Typparameters der generischen Basisklasse

Abschnitt 20.6 Typisierte DataSets

Kapitel 20: Datenbankpr

Kapitel 20: Datenbankpr

21 LINQ

Seit C# 3.0 bzw. .NET 3.5 ist eine einheitliche,

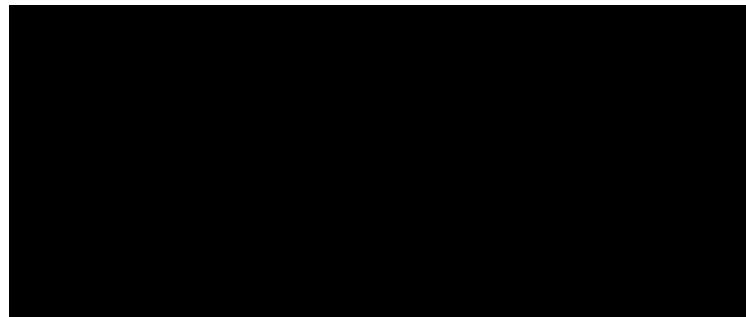
Eine zentrale Rolle bei LINQ-Abfragen spielen diejenigen Erweiterungsmethoden, welche die statische Klasse **Enumerable** im Namensraum **System.Linq** für *alle* Typen realisiert, welche die Schnittstelle

Quellcode-Segment	Ausgabe
<pre>Console.WriteLine(customers[6].CustomerID);</pre>	

22.1 Entity Data Model

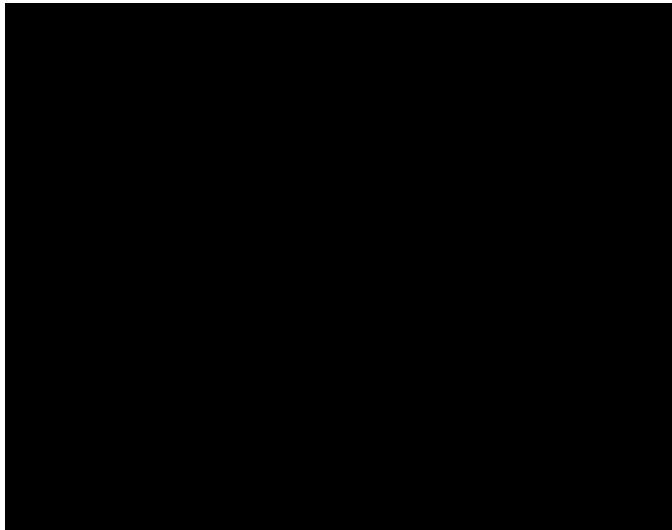
Anwendungsentwickler treffen oft auf eine komplexe und durch Normalisierung optimierte Datenbank, welche die Ein- und Ausgabedaten des geplanten Programms (z.B. Kunden, Aufträge) per-

In Abhängigkeit vom Typ der Datenbankverbindung wird nachgefragt, ob die Datenbankdatei in den Projektordner kopiert werden soll:



Im Beispiel wird nach einer Zustimmung in der **Entitätszeichenfolge** der feste Pfad zur Datenbankdatei durch eine relative

Abschnitt 22.1 Entity Data Model



String

beiden ORM-Lösungen aus dem Hause Microsoft eindeutig das Entity Framework die besseren Zukunftschancen hat als der Konkurrent LINQ to SQL.

Aufgabe 2**1. Falsch**

Plattformunabhängigkeit setzt allerdings voraus, dass ein Assembly für die Platform MSIL übersetzt wurde.

2.

Abschnitt 23.2 Lösungsvorschläge zu den Übungsaufgaben

Abschnitt 23.2 Lösungsvorschläge zu den Übungsaufgaben

Abschnitt 23.2 Lösungsvorschläge zu den Übungsaufgaben

Abschnitt 23.2 Lösungsvu8

MacBeth, G. S. (2004). *C#. Programmer's Handbook*. New York: Springer.

Datenkapselung 151, 164

Datenstrom 429

Datentyp 84

Datentypen

elementare 87

Stichwortregister

override 269, 270
Owner 536, 544

P

PageContent 600
Panel 360
PAP 132
Parallel 483
Parametrisierte Abfragen 642
params 171
partial 214, 330
Pascal 155
Pascal Casing 78
PasswordBox 388
Passwörter 388
PathSegment 576
Peek() 443
Pfadname 449

UploadValues 508
Uri 333
URI 498
URL 498
UrlEncode() 507
URL-Kodierung 505
User Datagram Protocol 496

Z

- Zahlenkreis 125
- Zeichenketten 242
- Zeichenkettenliterale 101
- Zeilenumbruch 255